FROM QUERIES TO DECISIONS: COMPUTATION, LEARNING, OPTIMIZATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Matthew T. Ford

August 2026

© 2026 Matthew T. Ford
ALL RIGHTS RESERVED

FROM QUERIES TO DECISIONS: COMPUTATION, LEARNING, OPTIMIZATION

Matthew T. Ford, Ph.D.

Cornell University 2026

Many computational decision systems begin by asking a question. A simulation is run, an example is sent for labeling, or a person or language model is asked to compare alternatives. The response must then be estimated, represented numerically, and incorporated into a model that guides an action. Across simulation optimization, active learning, exam scheduling, and preference elicitation, this dissertation studies that pipeline from queries to decisions. Its central finding is that decision quality depends not only on the final optimizer but also on how its inputs are computed and whether they support the interpretation assigned to them.

Simulation optimization estimates performance by repeatedly running a stochastic computer model. Improving the system often requires a gradient, but deriving and implementing gradient estimators by hand can be difficult. This dissertation evaluates automatic differentiation, which obtains derivatives directly from program code, for constructing simulation-gradient estimates. Experiments identify runtime, memory, compilation, and implementation tradeoffs relative to finite differences and manual derivatives. Automatic differentiation exactly differentiates a realized simulation, but this alone does not guarantee an unbiased estimate of the derivative of expected performance.

In active learning, an algorithm chooses which examples should be labeled next. Selecting from a fixed pool can exclude potentially more informative examples absent from that pool, while synthesizing examples requires searching a high-dimensional

space without producing unnatural queries. This dissertation formulates joint batch search in the lower-dimensional space of a learned generative decoder, which may yield more realistic or natural-looking examples than unconstrained search over raw inputs. Batches are scored by expected information gain: how much observing their labels would reduce uncertainty about the classification model. A new Monte Carlo estimator evaluates related terms using the same random draws so that their fluctuations can cancel. In fixed-batch experiments, it substantially reduces variance while retaining the standard estimator's expected value and finite-sample bias. A remaining challenge is optimization: differentiating a sampled score while holding its sampled labels fixed ignores how their distribution changes with the proposed examples and can therefore produce a biased gradient.

University exam scheduling illustrates how a useful objective can still be computationally difficult to optimize. A schedule should avoid direct conflicts while limiting undesirable patterns such as back-to-back exams or several exams within a short period. On three public Cornell semester datasets, a heuristic developed in this dissertation constructs conflict-free schedules in about two minutes whose weighted penalties are competitive with those of a published Cornell benchmark method. An exact decomposition separates the overall timetable from student-level penalty calculations. Representing each student's schedule by only its occupied exam slots makes those calculations inexpensive, but the overall timetable problem becomes slow as constraints accumulate and fails to improve its provable lower bound. The exact runs count penalties only for students taking at most five exams and thus understate the full objective; even so, their best values are worse than the heuristic's full-population scores. These experiments identify both an effective practical method and the main obstacle to an effective exact method.

The penalties used in exam scheduling implicitly assign one set of preferences

to the entire student population. A choice survey of 981 Cornell students tests that simplification. Models allowing preferences to differ by student modestly improve prediction of responses not used to fit the model relative to a model that assumes everyone shares the same preferences. More importantly, the differences are concentrated in a consequential tradeoff: some students prioritize spacing exams apart, whereas others prioritize finishing the exam period earlier. Changing a schedule along this dimension therefore shifts burden between preference types rather than benefiting everyone equally.

Large language models offer a convenient way to translate natural-language preferences into numerical inputs. If one underlying dollar-valued utility explains these inputs, then valuing items separately, comparing priced offers directly, and combining comparisons along a path should agree after accounting for price. Statistical tests across six language models and three controlled domains reject global self-consistency for every model. Even after assigning a separate unrestricted value to every item, the best-fitting utility can leave errors that are large relative to the prices of the compared offers. Direct item-versus-offer comparisons frequently show statistically supported disagreement, and some imply opposite choices between the offers. Changing the query format can therefore materially change both the estimated preference and the resulting decision.

Together, these studies show that optimization cannot validate its own inputs. Decision methods should verify estimator validity, represent consequential individual variation, and test whether information obtained in different ways supports the common model assumed downstream.

BIOGRAPHICAL SKETCH

Matthew T. Ford earned a Bachelor of Science in Statistics and Analytics and a Bachelor of Arts in Mathematics from the University of North Carolina at Chapel Hill in May 2020. He began his graduate studies in Operations Research and Information Engineering at Cornell University in August 2020 and received his Master of Science in December 2023.

Within Cornell's Ph.D. program in Operations Research, Matthew pursued the concentration in Applied Probability and Statistics, with graduate minors in Applied Mathematics and Data Science. His doctoral research spans simulation optimization, Bayesian active learning, discrete optimization, and preference elicitation.

ACKNOWLEDGEMENTS

I am grateful to the members of my special committee: my advisor, Shane G. Henderson; Peter I. Frazier, with whom I worked closely; and Siddhartha Banerjee.

I am especially grateful to Shane for his intellectual support and for the many ways he supported me beyond my research during our time together. He knew how to wait patiently before turning the rudder at just the right moment, giving me the freedom to explore ideas for myself while guiding me toward fruitful paths. His example has shaped how I approach research, collaboration, and relationships more broadly.

Peter was extraordinarily generous with his time and in sharing his ways of thinking. Our conversations repeatedly sharpened my thinking, particularly by helping me translate personal intuitions into targeted questions and insights that strengthened my professional work. I am grateful for his candid feedback, enthusiasm for science, and close mentorship across several projects.

I would also like to thank my many collaborators, mentees, and students for everything we shared. You helped me grow and made this journey so much more enjoyable.

Finally, I would like to thank my parents, grandparents, and friends (you know who you are) for being with me along the way.

This work was supported in part by National Science Foundation grants OAC-2410950 and DMS-2230023 and by the Office of Naval Research grant N00014-22-1-2763.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Acknowledgements . . . iv
Table of Contents . . . v
List of Tables . . . ix
List of Figures . . . xi

1 Overview **1**
1.1 From Queries to Decisions . . . 1
1.2 Main Themes . . . 2
1.3 Chapters of the Dissertation . . . 3

2 Automatic Differentiation for Gradient Estimators in Simulation **8**
2.1 Introduction . . . 8
2.2 Gradient Estimation and Automatic Differentiation . . . 10
2.3 Considerations for Automatic-Differentiation Tools in Simulation . . . 13
2.3.1 Autograd . . . 15
2.3.2 JAX . . . 16
2.4 Adapting Simulation Code for Autograd and JAX . . . 18
2.4.1 Pseudo-Random Number Generators . . . 18
2.4.2 Object-Oriented vs. Functional Programming . . . 19
2.4.3 Event Lists . . . 19
2.4.4 Loops and Conditionals . . . 20
2.4.5 An Example . . . 20
2.5 Incorporating Automatic Differentiation into SimOpt . . . 22
2.6 Experiments . . . 26
2.6.1 Stochastic Activity Network . . . 27
2.6.2 Ambulance System . . . 28
2.7 Discussion and Conclusion . . . 29

3 Batch Synthesis in High-Dimensional Classification Active Learning via Latent Gradient Optimization **31**
3.1 Introduction . . . 31
3.1.1 Contributions . . . 33
3.1.2 Organization of Paper . . . 35
3.2 Background and Assumptions . . . 36
3.2.1 Bayesian-Like Classification Models . . . 36
3.2.2 BatchBALD Acquisition Function . . . 37
3.2.3 Generative Models . . . 38
3.3 Methodology . . . 38
3.3.1 LGO-BatchBALD: QS via LGO . . . 38
3.3.2 Estimating BatchBALD and Its Gradient . . . 39
3.3.3 Complexity . . . 44

3.4 Computational Experiments 46
3.4.1 Estimator Validation 46
3.5 Limitations and Future Work 47
3.6 Conclusions 49
3.6.1 Theory for the Proposed Estimator 50
3.6.2 Additional Experimental Details 53

4 Optimization Algorithms for Student-Aware Exam Timetabling 55
4.1 Introduction 55
4.2 Related Work 57
4.3 Empirical Overview 60
4.3.1 Semester Instances 61
4.3.2 Headline Comparison 62
4.3.3 Evaluation Notation 64
4.3.4 Bad-Event Metrics and Operational Rules 65
4.4 Primal Scheduling Heuristic 68
4.4.1 Representation and Search Objective 69
4.4.2 Configuration Used for the Reported Benchmark 70
4.4.3 Constructive Scheduling Phase 72
4.4.4 Insertion of Unscheduled Exams 74
4.4.5 Local Search 74
4.4.6 Construction and Local-Search Improvement 78
4.4.7 Cornell-Weighted Comparison 79
4.5 Heuristic with Front-Loading 79
4.6 Attempts to Limit Daily Student Participation 82
4.7 Motivation for Exact Optimization 84
4.8 Full-Space Mixed-Integer Formulation 84
4.9 Benders Decomposition 87
4.9.1 Master Problem 89
4.9.2 Student Subproblem 90
4.9.3 Benders Algorithm 92
4.9.4 Why the Cuts Can Be Weak for the Binary Master 93
4.9.5 Combination-Based Student Subproblem 94
4.9.6 Occupied-Slot Student Reformulation 95
4.9.7 Implementation of the Reported Benders Runs 98
4.9.8 Formulation Size 100
4.10 Implied Formulation Sizes for the Semester Instances 101
4.11 Benders Performance and Bottleneck Analysis 104
4.11.1 Occupied-Pattern Precomputation 105
4.11.2 Five-Iteration Cornell-Objective Runs 106
4.12 Future Work 110
4.13 Conclusion 111

5 Student Preferences in Exam Scheduling: Heterogeneity in the Tradeoff Between Spacing and Finishing Early 113
5.1 Introduction . . . 113
5.1.1 Research questions . . . 114
5.2 Background and Related Work . . . 116
5.3 Methods: Survey Design and Participants . . . 119
5.3.1 Schedule representation . . . 119
5.3.2 Schedule features . . . 119
5.3.3 Questions . . . 120
5.3.4 Deployment and participants . . . 121
5.4 Analytic Approach and Validation . . . 123
5.4.1 Model overview . . . 123
5.4.2 Utility model . . . 124
5.4.3 Question temperatures and ranking likelihood . . . 125
5.4.4 Priors and estimation . . . 126
5.4.5 How the anchor questions map to model parameters . . . 127
5.4.6 Predictive-validation metrics . . . 128
5.5 Results . . . 130
5.5.1 Descriptive pattern: Students agree on bunching but are split over spacing versus finishing early . . . 130
5.5.2 RQ1: The main individual heterogeneity is the spacing/finish tradeoff . . . 131
5.5.3 Validation: The anchor questions elicit sharper choices than the application block . . . 132
5.5.4 RQ1: Individual preferences generalize to realistic schedules 133
5.5.5 RQ2: College of Engineering students are more often spacing-preferring . . . 137
5.5.6 Scale and identification checks . . . 144
5.5.7 RQ3: Population-level weights for timetabling . . . 145
5.6 Discussion and Implications . . . 148
5.6.1 Implications for exam-scheduling policy . . . 148
5.6.2 Implications for engineering education . . . 150
5.6.3 Limitations and future work . . . 151
5.7 Conclusion . . . 151
5.8 Survey ranking questions . . . 153
5.9 Survey sample and response quality . . . 156
5.10 Model details and supplementary analyses . . . 159
5.10.1 Prior specification . . . 159
5.10.2 Sampling diagnostics . . . 159

6 LLM-Derived Preference Judgments Are Not Self-Consistent 166
6.1 Introduction . . . 166
6.2 Related Work . . . 170
6.3 Utilities and LLM Estimates . . . 172

6.3.1 Items, Offers, and Utility . . . 172
6.3.2 Two Query Types . . . 173
6.3.3 Self-Consistency Hypothesis . . . 174
6.3.4 Global Test for Self-Consistency . . . 176
6.3.5 Local Tests for Self-Consistency . . . 176
6.3.6 Statistical Evidence and Practical Magnitude . . . 178
6.4 Audit Protocol . . . 179
6.4.1 Domains and Preference Utterances . . . 179
6.4.2 Queries, Models, and Sampling . . . 180
6.4.3 Inference and Reporting . . . 181
6.5 Audit Results . . . 182
6.5.1 Does One Utility Fit the Query Means? . . . 182
6.5.2 Where Does Self-Consistency Fail? . . . 183
6.6 Implications for Preference Learning . . . 184
6.7 Conclusion . . . 185
6.8 Formal basis of the audit . . . 188
6.8.1 Why the queries estimate item utilities and offer-utility differences . . . 188
6.8.2 Why P1 and P2 follow from self-consistency . . . 189
6.9 Experimental reproducibility details . . . 190
6.9.1 Domains, items, and query counts . . . 190
6.9.2 Audited preference utterances . . . 191
6.9.3 Prompts and exact query specification . . . 192
6.9.4 Model and API configuration . . . 193
6.10 Supplementary nonfinite-output diagnostics . . . 195
6.11 Supplementary path-length stress check . . . 195
6.12 P2 results by domain . . . 196
6.13 Statistical details . . . 199
6.13.1 Audit-group self-consistency tests . . . 199
6.13.2 Local residual uncertainty . . . 204

LIST OF TABLES

2.1 SAN timing with AD applied per replication 27
2.2 Ambulance-system timing results 28

3.1 Comparison of query-synthesis formulations 33
3.2 Fixed-batch MNIST estimator comparison 47

4.1 Exam enrollment instances by semester 61
4.2 Students by number of exams 62
4.3 Unique exam sets by number of exams 62
4.4 Headline three-semester method comparison 64
4.5 Headline heuristic configuration 71
4.6 Original heuristic construction and search results 80
4.7 Comparison under Cornell penalty ratios 80
4.8 Front-loading rule variant comparison 81
4.9 Daily-cap repair results 83
4.10 Approximate formulation sizes 100
4.11 Slot-combination and assignment counts by number of exams . . . 101
4.12 Semester-scale master and restricted workload 103
4.13 Occupied-pattern cost and dual precomputation 105
4.14 SP24 Benders iteration trace 107
4.15 Best restricted and full objectives from Benders runs 107
4.16 Benders timing and conservative heuristic comparison 108
4.17 Cross-semester Benders timing and progress 108

5.1 Survey response rates by COE status 122
5.2 Out-of-sample predictor performance 135
5.3 Normalized spacing-preference estimates by COE status 140
5.4 Relative population-level bunching penalties 147
5.5 Learned and operational bunching-weight comparison 147
5.6 College and graduation-year distribution 158
5.7 Engineering-major composition of the COE subsample 158
5.8 Raw population-level bunching penalties 160
5.9 Posterior block temperatures 161
5.10 In-sample predictor performance 162
5.11 Paired out-of-sample log-score tests 163
5.12 Leave-one-question-out validation results 164
5.13 COE effect on scheduling-horizon preference 164
5.14 Posterior covariate effects on individual parameters 165

6.1 LLM preference audit design 180
6.2 Best-fitting utility RMSE by model and domain 183
6.3 Local P1 and P2 self-consistency results 184
6.4 Audit design by domain 191

6.5 Model and API configuration 193
6.6 Numerical-output parseability 195
6.7 Path-length stress-check summary 197
6.8 P2 self-consistency results by domain 198
6.9 Audit-group self-consistency tests 204
6.10 Audit-group self-consistency tests, continued 205

LIST OF FIGURES

2.1 Event-list implementation without JIT 21
2.2 JAX event-list implementation with JIT 23
2.3 Stochastic activity network and aggregate-AD timing 27

3.1 Latent gradient optimization for query synthesis 33
3.2 LGO-BatchBALD workflow 38
3.3 Exact two-state estimator variance ratios 54

5.1 COE differences in spacing and bunching preferences 138
5.2 Posterior spacing-type probability by COE status 142
5.3 Six survey ranking questions 154
5.4 Sample composition by graduation year and college 157
5.5 Attention-check pass rates 157
5.6 COE covariate effects on bunching penalties 161
5.7 Posterior block-temperature intervals 162
5.8 Distribution of normalized spacing preference by COE status . . . 163
5.9 Time-of-day preference by COE status 165

6.1 Representative LLM-assisted preference learning 167
6.2 Local implications of self-consistency 179
6.3 P1 and P2 disagreement threshold curves 187
6.4 Path length and P1 disagreement 196

CHAPTER 1

OVERVIEW

1.1 From Queries to Decisions

Many computational decision systems begin with a query. A simulation is evaluated at a chosen input, an oracle is asked to label an example, a person is asked to rank alternatives, or a language model is prompted to express a preference. The response is rarely a decision by itself. It must be estimated, represented, and incorporated into a model whose output guides an action. The resulting pipeline runs from queries, through information and computation, to decisions.

This dissertation studies several points along that pipeline. Its settings include simulation optimization, active learning, exam timetabling, human preference measurement, and large-language-model (LLM) preference elicitation. Across these settings, the central question is the same: what information does a decision procedure require, and how can that information be obtained and used without losing sight of what it represents? The answer depends not only on the final optimizer, but also on the query, estimator, representation, and assumptions connecting the resulting numbers to the intended decision.

Chapters 2 and 3 study how to compute simulation gradients and synthesize informative classification queries. Chapters 4 and 5 examine two sides of exam scheduling: optimizing with a shared bad-event objective and testing how well a population-level model represents individuals. Chapter 6 audits whether LLMs can coherently translate an individual's natural-language preferences into numerical inputs.

1.2 Main Themes

Information is produced through computation. A query response is not an abstract, costless fact. Computing a simulation gradient requires a differentiable program and conditions under which a pathwise derivative estimates the desired gradient. Evaluating an information-based active-learning objective requires Monte Carlo estimation. Measuring preferences requires a survey or prompting design and an observation model. Thus, what a system can learn is partly determined by how its information is computed. This makes estimator correctness a recurring concern. Automatic differentiation can exactly differentiate a realized computation while the pathwise derivative remains biased for the derivative of an expectation. Likewise, differentiating a realized Monte Carlo acquisition estimate while holding sampled labels fixed can omit how their distribution changes with the query. Computational availability does not establish that a quantity has the meaning an optimizer assigns to it.

Structure enables computation but introduces assumptions. Computational graphs enable automatic differentiation; a generative decoder restricts high-dimensional query synthesis; occupied-slot patterns expose student-level scheduling structure; hierarchical choice models separate population and individual preferences; and quasi-linear utility yields testable restrictions on monetary LLM judgments. These representations reduce complexity, but none is neutral. A decoder does not guarantee realistic queries, a valid decomposition need not be a practical algorithm, a population utility can conceal individual variation, and quasi-linearity is meaningful only with an appropriate numeraire. Structure is useful when its assumptions are explicit and its consequences are checked.

Optimization cannot validate its own inputs. Optimization can precisely solve an objective whose inputs do not represent every individual. The survey shows that individual preference models modestly improve held-out prediction and that the important heterogeneity is concentrated in one policy tradeoff. LLMs may conveniently translate personal descriptions into numerical inputs, but optimization cannot establish that answers to different prompts measure one stable utility. Fitting one objective to biased, heterogeneous, or inconsistent information resolves disagreement through modeling choices rather than making it disappear. The dissertation therefore reports limitations and negative results alongside successful methods, identifying where new estimators, representations, or calibration data are needed.

1.3 Chapters of the Dissertation

Chapter 2: automatic differentiation for simulation gradients. Simulation optimization can query a simulator without directly observing the expected objective or its gradient. Chapter 2 evaluates automatic differentiation (AD) tools for constructing pathwise gradient estimators from simulation code. Experiments on a stochastic activity network and an ambulance simulation show that AD can avoid manual derivative coding with runtime competitive with finite differences, although manual derivatives are usually fastest and JAX compilation requires substantial code restructuring. Applying AD separately to each replication also reduces memory and permits variance estimation. AD differentiates the realized program, but unbiasedness for expected performance still requires valid interchange of differentiation and expectation. This chapter appeared as Ford et al. (2022). I was first author. Henderson proposed the topic and advised the work. I developed the software, designed and conducted all experiments, analyzed the results, and led

the writing.

Chapter 3: high-dimensional active-learning queries. Chapter 3 moves from evaluating fixed inputs to synthesizing informative classification queries. It formulates latent gradient optimization for jointly synthesizing batches under a generative constraint and evaluating them with BatchBALD, which values a query by its expected information about uncertain model parameters (Kirsch et al., 2019). A proposed Monte Carlo estimator has the same finite-sample expectation as the standard estimator and lower variance in the validated fixed-batch experiments, although the reduction is not universal. The readily available derivative that holds sampled labels fixed omits their changing probabilities and can be biased, so a validated end-to-end optimizer remains future work. This unpublished manuscript is by Matthew T. Ford, Simiao Ren, Jordan M. Malof, and Shane G. Henderson. I conceived and conducted the methodological work, derived the estimator and its theoretical results, implemented all code, designed and conducted all experiments, analyzed the results, and led the writing. An anonymous reviewer identified an issue with an earlier estimator, which motivated the current construction; I developed the replacement and its analysis. Ren and Malof contributed to the literature review and writing, while Henderson advised the work and contributed to its framing and revision.

Chapter 4: student-aware exam-timetabling decisions. Chapter 4 studies exam scheduling when student costs depend on conflicts, closely spaced exams, and operational restrictions on large exams and daily student participation. It develops a fast heuristic and an exact Benders formulation whose occupied-slot representation permits precomputation of pattern costs and solver-returned LP duals. On the three Cornell semesters with conflict-free published comparison schedules, one heuristic

replication per semester produces a complete conflict-free schedule in about two minutes with a score competitive with Cornell Hybrid under its published objective weights. The Benders experiments restrict the objective to groups taking at most five exams. Although precomputation makes their evaluation negligible, the master consumes nearly all measured loop time and the lower bound remains zero; even a conservative objective comparison strongly favors the heuristic. This unpublished work was advised by Shane G. Henderson and Peter I. Frazier. I developed and implemented the heuristic. Henderson and Frazier conceptualized the Benders formulation; I implemented it, developed the occupied-slot reformulation and solver-derived dual precomputation, and conducted the computational experiments.

Chapter 5: measuring student preferences. Chapter 5 tests how well Chapter 4's common objective weights represent individual students. It analyzes rankings from 981 Cornell students using hierarchical Plackett–Luce models. Individual effects improve held-out prediction, showing preference variation omitted by a population objective, although the top-choice gain is only about two percentage points. The main heterogeneity is the tradeoff between greater exam spacing and finishing earlier; students agree more strongly that bunching is undesirable. The chapter also estimates population bunching weights, but its larger contribution is making visible how any one schedule distributes burden between preference types. This working paper by Matthew T. Ford, Peter I. Frazier, Shane G. Henderson, and Jessica Rush Leeker is planned for the *Journal of Engineering Education*. I designed and implemented the survey and statistical model, collected the data, managed the human-subjects process, conducted the analysis, and wrote the manuscript. Frazier and Henderson advised the work, and Rush Leeker contributed to its positioning and narrative.

Chapter 6: auditing LLM-derived preference information. Chapter 5 shows the value of individual rather than only population preference information. Systems increasingly use LLM responses to natural-language preferences as structured data (Jovine et al., 2025; Kobalczyk et al., 2025; Austin et al., 2024). Chapter 6 tests whether answers from item queries, offer-pair queries, and path-based compositions can be represented by one quasi-linear, dollar-denominated utility. It develops global and local statistical diagnostics and demonstrates them across six models and three controlled domains. Every model rejects the global self-consistency claim. These rejections are accompanied by decision-relevant departures, although their magnitude varies by domain. Even after assigning one unrestricted dollar value to every audited item, the best-fitting utility leaves average root mean squared errors of 18.9–44.9% of the mean prices of the compared offers for hotels, compared with 1.6–6.1% for flights and apartments. In the direct comparison between item and offer-pair queries, 41.7–87.5% of residual confidence intervals exclude zero across models, and 2.1–12.5% of comparisons support opposite offer choices. Consequently, the query format can materially change both an estimated utility difference and the resulting decision. This audits internal coherence, not accuracy for a person. The planned submission is by Matthew T. Ford, Francis Bahk, Jingjing Wang, Adam S. Jovine, Tinghan Ye, David B. Shmoys, and Peter I. Frazier. I conceived the project, formalized the audit, designed the prompts and experiments, implemented the code, collected and analyzed the model responses, developed the statistical procedures, and wrote the manuscript and literature review. Bahk designed the hotel items and query paths after I had developed the other domains. All coauthors provided feedback on the experiment design, figures, presentation of results, and writing, and Frazier advised the work throughout. This chapter also reuses limited infrastructure from the earlier LISTEN codebase, to which Bahk, Wang, Jovine,

and Ye contributed.

Together, these chapters show that moving from queries to decisions requires more than an effective optimizer. It requires computationally valid ways to obtain information, representations that preserve consequential individual variation, and evidence that translated numerical inputs measure the quantities the final optimization assumes.

CHAPTER 2

AUTOMATIC DIFFERENTIATION FOR GRADIENT ESTIMATORS IN SIMULATION

2.1 INTRODUCTION

Gradient estimation is important in both sensitivity analysis and optimization in the practice of simulation. We emphasize the simulation-optimization context for concreteness, but the ideas we present also apply to sensitivity analysis. Consider the problem of minimizing $f(x) = \mathbb{E}h(x, Y)$ where $x \in \mathbb{R}^d$ is a vector of decision variables, Y is a random element capturing all stochastic primitives in the simulation, $h(x, Y)$ represents the logic of the simulation model returning some real-valued performance measure, and $f\colon \mathbb{R}^d \to \mathbb{R}$ is the objective function. We assume that x is a *structural* parameter, i.e., the distribution of Y does not depend on x. This is essentially without loss of generality since x can include any finite set of parameters that are needed to map Uniform$(0, 1)$ random variables, which are then represented as Y, to the simulation output.

Assuming appropriate differentiability conditions, first-order optimization algorithms use information about the gradient of the objective function at a solution x, $\nabla f(x)$, to try to find a better neighboring solution x' such that $f(x') < f(x)$. In the simulation-optimization setting, we do not have direct access to $f(\cdot)$ or $\nabla f(\cdot)$. However, we have the ability to sample Y and then compute $h(x, Y)$ for any x. We may also be able to compute the gradient of $h(\cdot, Y)$ at x, $\nabla h(x, Y)$, through analytical derivation and manual coding, or using automatic-differentiation (AD) software. In that event, $\nabla h(x, Y)$ can be used to construct estimators of $\nabla f(x)$ to be used by first-order optimization algorithms.

We discuss considerations when selecting and using AD software for gradient

estimation. Many modern AD tools have been designed primarily for machine-learning (ML) problems, e.g., PyTorch (Paszke et al., 2017), JAX (Bradbury et al., 2018), Autograd (Maclaurin et al., 2015), and Tensorflow (Abadi et al., 2015), and thus are tailored to a different assumed program structure than what is typical in simulation. As we note in Section 2.2, there is a direct relationship between AD and infinitesimal perturbation analysis (IPA) gradient estimators. There we also discuss some general computational principles of AD. Section 2.3 reviews some of the main AD tools and their suitability for simulation modeling in general. Section 2.4 discusses adapting simulation code for use with two of those tools, and Section 2.5 explores their use in the Python library SimOpt (Eckman et al., 2020). Section 2.6 provides timing experiments for running the various tools on two relatively simple examples and Section 2.7 discusses the results of the experiments and provides some final perspectives.

The use of AD in simulation has been contemplated for a long time, e.g., Fu et al. (2000). One area of simulation in which AD has found success is finance, e.g., Giles and Glasserman (2006); Glasserman (2013); Fries (2017), and Fries (2019), perhaps because of the relative simplicity of pricing simulation models compared to discrete-event simulation models. Another area is at the intersection of Monte Carlo simulation and machine learning, such as in variational inference (Kucukelbir et al., 2017) or in modern implementations of policy gradient estimators in reinforcement learning, building upon Williams (1992). Relative to such work, we emphasize issues with adapting existing AD software tools to simulation models, using two simple examples to emphasize the key messages and gain insights into coding complexity and run times.

2.2 GRADIENT ESTIMATION AND AUTOMATIC DIFFERENTIATION

When x is a structural parameter we can use infinitesimal perturbation analysis (IPA) for gradient estimation Glasserman (1991). The value of the IPA gradient estimator $\nabla h(x, Y)$ can be obtained by differentiating (with respect to x) the source code describing the simulation logic h and evaluating the result at (x, Y).

The IPA gradient estimator is unbiased at a solution x if the interchange of differentiation and expectation,

$$\nabla f(x) = \nabla \mathbb{E} h(x, Y) = \mathbb{E} \nabla h(x, Y), \tag{2.1}$$

holds Fu and Hu (1997). Cao (1985) gives conditions for when this interchange holds; see also Asmussen and Glynn (2007, VII.2). If these conditions do not hold, we can sometimes replace $h(x, Y)$ by a conditional expectation that is differentiable in x to obtain an unbiased gradient estimator; this technique is known as Smoothed Perturbation Analysis Gong and Ho (1987). Even if the IPA gradient estimator is biased, it may still be useful for optimization Eckman and Henderson (2020b).

Relative to other gradient estimation methods, IPA gradient estimators possess certain advantages. IPA gradient estimators can be obtained from a single simulation replication, i.e., one realization of Y and evaluation of $\nabla h(x, Y)$, while methods such as finite differences or weak derivatives typically require approximately $2d$ replications to obtain a single gradient estimate. In this sense, IPA is computationally efficient. The method of finite differences also suffers from needing to choose a step size that balances bias and variance; IPA requires no such choice. Often, likelihood ratio gradient estimators Glynn (1990) can also be obtained from a single replication, but they have been seen to suffer from higher variance than IPA (Cui et al., 2020). A further advantage of IPA gradient estimators relative to

their competitors is that they are the *exact* (up to numerical precision) gradients of the pathwise function, which ensures that the function and gradient estimates are consistent with one another. This consistency could be expected by solvers when, e.g., taking very small steps. The consistency does *not* hold for gradient estimates obtained using, e.g., the likelihood ratio method. Fu (2015) provides a recent survey of gradient estimation in simulation. In the rest of this paper we focus on IPA gradient estimators and their calculation using AD.

AD is a method to differentiate functions which are compositions of primitive functions whose gradients are computable at all arguments of interest Griewank (1989). The chain rule provides a formula to differentiate the composite function, which is $h(x, Y)$ for some fixed Y in our setting. Wikipedia contributors (2022) provides a highly accessible introduction to AD, including most of the topics discussed here.

It is now convenient to redefine $z = h(x, Y)$ to be an $\mathbb{R}^m$-valued vector of outputs as a function of the inputs x and random element Y. Thus, we now consider multiple outputs instead of the special case $m = 1$ assumed till now. We thus seek the Jacobian of h, giving a matrix of partial derivatives of each component of z with respect to each component of x. The Jacobian is important in simulation optimization, for example, when both the objective function and some constraint functions must be estimated.

The two most common methods for AD calculation of the Jacobian of a composite function are forward and reverse mode differentiation, both of which calculate the same numerical value as the IPA estimate but do so in different ways. Both methods work with the same code used to compute $h(x, Y)$. Forward mode applies the chain rule from the "inside out," calculating and carrying forward derivatives of each intermediate variable with respect to each component of x as the original

program runs. Reverse mode recursively works from the "outside in," i.e., from the final simulation output z back towards the simulation input x, using the chain rule after the initial program has finished execution, requiring access to the values of all intermediate variables from the forward pass. While it is possible to use a combination of forward and reverse mode in different parts of the program to reduce the amount of computational work, finding the approach that minimizes computation is an NP-complete problem Naumann (2008) and thus is rarely pursued in practice. When implemented well, AD of a function has a computational cost that is on the order of the cost to evaluate the function. Forward mode has constant memory complexity and computational complexity which scales with the number of inputs. On the other hand, reverse mode has memory complexity which scales with the number of intermediate variables and computational complexity which scales with the number of outputs (van Merriënboer et al., 2018). Thus forward mode is typically preferred to reverse mode when there are more outputs than inputs and vice versa. In the simulation-optimization setting where the number of outputs m is typically small or even 1, reverse mode is preferred.

Reverse mode must define a way to construct the chain rule to work backwards from the output to the input. We summarize the two main approaches which are discussed in more detail in Chapter 6 of Griewank and Walther (2008) and whose practical tradeoffs are discussed in van Merriënboer et al. (2018). The first is operator overloading, which overloads the functions in the original program to build an instance-dependent computational graph on the primitive functions used in the forward pass. This makes it easy to handle control-flow operations because the computational graph is static for a fixed instance, making the backward pass equivalent to there being no control-flow operations in the original program. A disadvantage of this approach is that building the computational graph each time

the function is called incurs overhead, and since the graph can change with each instance, it is hard or inefficient to optimize the code that executes the backwards pass. The other approach is source code transformation which, from the source code of the program, explicitly constructs another program, called the adjoint program, to perform the backward pass based purely on the intermediate values stored during the forward pass. The advantage of this approach is that the adjoint program only has to be constructed once and thus incurs no overhead at run time, other than potentially on the first replication. The program for the backward pass can also be constructed prior to run time and can be optimized and compiled prior to run time. The disadvantage is that implementing the transformation rules for certain function calls and control-flow operations to construct the adjoint program can be complex. This can also lead to long compilation times of the adjoint program.

2.3 CONSIDERATIONS FOR AUTOMATIC-DIFFERENTIATION TOOLS IN SIMULATION

Several factors impact the utility of AD tools for simulation:

- The tool must be able to automatically generate the derivatives of primitive functions, such as $\sin x$ and e^x, that are used in the code to compute $h(x, Y)$. Some tools, such as Autograd and JAX, have mimicking implementations of functions from popular libraries such as NumPy, so that code written in basic Python and NumPy can be differentiated with minimal changes to the code.
- Some AD tools require code to be restructured to address complications posed by control-flow operators such as `for` and `while` loops and `if` statements.
- Some AD tools restrict the types of data structures that can appear as inputs and outputs of the function to be differentiated.

- The run time of the AD tool can be important because simulations may require nontrivial computation.

The performance (run time) of AD tools depends on the structure of the software that is being differentiated. Certainly, ML programs differ in structure from simulation programs. The control flow of the forward pass of a neural network tends to be relatively simple. Deep neural networks can have millions of inputs but often only have a few outputs, which largely explains the popularity of a specialization of reverse mode to the neural-network setting known as back propagation. Furthermore, the majority of the computational effort of a forward or backward pass through a deep neural network is typically due to large matrix operations. Thus, the overhead incurred by constructing the per-instance computational graph or the adjoint program for the backwards pass can be negligible relative to the matrix computations. Differences in implementations of matrix operations can therefore account for much of the performance differences between AD software tools in ML. See Baydin et al. (2017) for a more thorough discussion of factors impacting the run times seen in practice of AD tools in machine learning applications. On the other hand, in discrete-event simulation, the program used to compute the function $h(x, Y)$ can have very complicated control flow with many conditional statements and loops. There are often a large number of sequential, non-vectorizable operations involved in performing a single replication of a simulation. Thus, the overhead of the AD software is more likely to have a noticeable impact on the total compute time of the program than in standard ML applications. In simulation, the ratio of inputs to outputs varies but is typically much smaller than in ML. These differences in the typical program structures in ML and discrete-event simulation may cause some AD software to perform better in one application than in another.

Some AD tools perform exceptionally well in specific situations. Zygote (Innes

et al., 2019) is a promising source-transformation-based AD tool for Julia that claims to be able to work with any data structure, trace through mutation, and thus differentiate any (differentiable) function in Julia, including those from other libraries not specially designed to work with Zygote. Enzyme Moses and Churavy (2020) is another source transformation AD tool that works with any language whose compiler compiles the source code to the LLVM intermediate representation including C, C++, Fortran, Julia, Rust, Swift, MLIR, and more. We now focus on Autograd and JAX due to their compatibility with Python and implementation of NumPy-like functions as primitives.

2.3.1 Autograd

Autograd is an AD library in Python that uses reverse mode via operator overloading. Primitive functions include basic Python functions like addition, NumPy-like functions like sin, its own AD functions, and the ability to define your own primitive functions. The `autograd.value_and_grad` function returns the value and gradient of the function to which it is applied, while `autograd.grad` returns just the gradient. The function to which `autograd.value_and_grad` or `autograd.grad` is applied must have a scalar output ($m = 1$) but is allowed to have multiple inputs ($d \geq 1$) with various data structures such as scalars, lists, arrays, and dictionaries supported. The `autograd.jacobian` function only supports taking Jacobians where the input x and output z are either scalars or arrays. Further, all inputs x for which derivatives in the Jacobian are desired must be passed into the first argument of the function being differentiated. Therefore, if x is multidimensional, then its values must be concatenated into a single array. Similarly, if the output z is multidimensional, then the function being differentiated should return a single array containing those outputs. There does not exist an `autograd.value_and_jacobian` function that

returns both the output and Jacobian in a single pass, but such a function can be easily written by modifying the code of `autograd.jacobian`. Another downside is that Autograd is not able to trace through certain operations such as in-place operations, commonly encountered when assigning new values to only part of an array, because these modify the intermediate variables. However, lists can be used freely and converted to arrays to perform calculations. More detail on what is supported and what workarounds exist are available on the tutorial page of the Autograd GitHub repository (Maclaurin et al., 2015).

2.3.2 JAX

JAX is another AD library in Python that combines an updated version of Autograd with the option of just-in-time (JIT, discussed below) compilation. JAX is capable of both forward and reverse mode. The implementation of reverse mode uses operator overloading. As primitives, JAX also has basic Python functions, NumPy like functions, its own AD functions, the ability to define custom primitives, and some additional utility functions. Like Autograd, JAX has `jax.grad`, `jax.value_and_grad`, `jax.jacfwd`, and `jax.jacrev` functions. The `jax.grad` and `jax.value_and_grad` functions are used in the same way as Autograd's versions. There are two versions of a Jacobian function to support forward (`jax.jacfwd`) and reverse mode (`jax.jacrev`) differentiation. These functions can be used the same way as `autograd.jacobian`, but they also allow inputs and outputs to be data structures other than scalars or arrays, such as lists and dictionaries. JAX provides a functional interface for array-element assignment and control flows, such as loops, to allow differentiation and JIT compilation as discussed in Section 2.4.

JAX stands for "just after execution," which refers to JAX being capable of JIT compilation. Our discussion of JIT compilation is mostly adapted from

Aycock (2003), to which we refer readers seeking more information. JIT compilers dynamically translate source code or bytecode to optimized machine code at run time. JIT compilers can provide the benefits of both static compilers and interpreters. While an overhead is paid at run time for the translation, the cost can be small relative to the cost of executing computationally heavy programs like long simulations or large numerical algebra tasks, and thus the speed can be similar to statically compiled programs. In contrast to static compilers, JIT compilers can recompile different pieces of code during run time if, for example, a new data type is encountered. This allows more flexibility in the source code than static compilers provide, yet not as much flexibility as interpreters. To JIT compile a function with JAX, the function must be pure-and-statically-composed, i.e., the function must only act on the inputs to the function and all changes must be passed as an output (as opposed to silently altering the state of an object) and must have a static computational graph on the primitive functions of JAX. A `while` loop is an example of a common component of simulation programs which violates this condition as the number of iterations, and thus the computational graph, varies with the input. This is why JAX provides a function which mimics the behavior of a `while` loop.

JIT compilation is not the only tool JAX provides to accelerate program execution. It also provides `vmap` and `pmap` functions which can somewhat automatically vectorize and parallelize code, respectively. Additionally, JAX provides GPU/TPU support providing further speedups in some cases. For more details regarding JAX, we refer the reader to the JAX GitHub repository (Bradbury et al., 2018).

2.4 ADAPTING SIMULATION CODE FOR AUTOGRAD AND JAX

Discrete-event simulation programs commonly feature pseudo-random number generation, custom written objects, event lists, loops and conditional statements. Depending on their implementation, these ingredients may cause issues when Autograd or JAX is used for AD. We discuss potential issues and solutions.

2.4.1 Pseudo-Random Number Generators

If x is a purely structural parameter, then there is no need to differentiate the code of the random number generator (RNG), but if any components of x are parameters of the distribution of Y, then we must differentiate the code of the RNG. Although both Autograd and JAX without JIT compilation (henceforth written JAXnoJIT) allow the use of any RNG, these software tools may not be able to differentiate through the RNG depending on the RNG's implementation. Indeed, many RNGs will not be compatible with JAX with JIT compilation (henceforth written JAXJIT) because the RNG may not be functionally pure. For example, an RNG object that outputs a number based on some internal state which gets updated each time the RNG is called would not be compatible with JAXJIT. This lack of compatibility arises because the output of the call to the RNG depends on the RNG's (internal) state which is neither static nor an input of the function being called. JAX's `jax.random` module revolves around a *key* object which acts as a state. This key object must be passed as an argument whenever a function is called to generate a random variate. For example, a sample from a Bernoulli(0.5) distribution could be drawn by calling `jax.random.bernoulli(key, p=0.5)`. The key must then be manually tracked and updated between its uses. Any generation

of random numbers used with JAXJIT must have a similar functional structure built around the use of a key or seed.

2.4.2 Object-Oriented vs. Functional Programming

As discussed earlier, Autograd and JAX can compute the Jacobian of a function $z = h(x, Y)$ for a fixed Y. In the code, it is important that x is an input and z is an output in the function signature. For example, in an object-oriented simulation program, a `model` object may contain the simulation logic in a method `model.simulate()` that does not take in any inputs, but instead internally accesses the simulation model's parameters via an attribute `model.parameters`. This code architecture makes it impossible for Autograd or JAX to differentiate `model.simulate()`, because the method has no explicit inputs. To make the code differentiable, the `model.simulate()` method would need to be rewritten to accept the model's parameters as an explicit input, e.g., `model.simulate(model.parameters)`.

2.4.3 Event Lists

Simulation models are often implemented using an event-driven world view. If the event list is implemented as a Python list, then retrieving, adding, and removing events from the event list are likely compatible with Autograd and JAXnoJIT without any modification. However, if the event list is implemented as a NumPy array, then the simplest way to get the event-list manipulations to be compatible with Autograd and JAXnoJIT is to switch it to a list. To use JAXJIT, the event list should be implemented as a JAX array. JAX arrays are also compatible with JAXnoJIT. When using a JAX array, it is necessary to use the functions JAX provides for retrieving and setting elements of the event list such as `arr.at[].get()` or `arr.at[].set()`. If the simulation requires taking a dynamic-sized slice of the

event list, i.e., the size depends on values not known until run time, one must instead use a functional approach yielding a static-sized slice. This is due to the requirements of the accelerated-linear-algebra compiler needing to know how much memory to allocate to the array regardless of the input. An example of a functional solution is given in Figure 2.2.

2.4.4 Loops and Conditionals

Standard Python loops and conditional statements are compatible with Autograd and JAXnoJIT. However, JAXJIT requires any loops or conditional statements to be replaced by functional equivalents, which can be found in the `jax.lax` module. Examples of converting a `for` loop and `if` statement are given in Figure 2.2. Additionally, if JAXJIT is used to differentiate a simulation containing the JAX functional equivalent of a `while` loop, then reverse mode will not be possible. This is again due to the compiler requiring a static bound on the size of the array needed to store the intermediate variables for the backwards pass. If a simulation requires a `while` loop and JAXJIT is used, forward mode is the only option.

2.4.5 An Example

Figure 2.1 gives code for a function that is used in an ambulance simulation in Section 2.6. This function takes in an event list and the number of events in the list and then returns the next event, the newly updated event list with the next event deleted, and the updated number of events in the list decremented by 1. The event list is unsorted, the first `num_events` elements are events in the list, and the rest of the list contains some dummy `place_holder_event`. Each event is itself a list containing information about the event such as the time at which it occurs and the type of the event. Thus, `event_list` is a list of lists. This code is compatible

```
import autograd.numpy as np
# alternatively, import jax.numpy as np if using JAX

def get_next_event(event_list, num_events):
    # initialize next event time to essentially infinity
    mini = SIM_LENGTH + 2.0
    # find the event with the earliest event time
    for i in range(num_events):
        if event_list[i][0] < mini:
            mini = event_list[i][0]
            next_event_index = i
    next_event = event_list[next_event_index]
    # delete next_event from event_list
    event_list[next_event_index:num_events+1] = (
        event_list[next_event_index+1:num_events+2]
    )
    num_events -= 1
    return event_list, num_events, next_event
```

Figure 2.1: Event list as an unordered list of lists, compatible with Autograd and JAX without JIT compilation.

with both Autograd and JAXnoJIT, but not with JAXJIT.

We now provide a modified version of the code, as seen in Figure 2.2, as an example of adapting code to be JAXJIT compatible. This modified code assumes that `event_list` is a two dimensional JAX array with each event itself being a JAX array.

The code in Figure 2.1, which is compatible with Autograd and JAXnoJIT, looks indistinguishable from typical code, except perhaps that we use a list of lists instead of an array or custom defined object for our event list. In contrast, in Figure 2.2, which is compatible with JAXJIT, the code had to be restructured:

1. The `for` loop had to be replaced with a functional equivalent.
2. Though not shown, we also have to write the `get_next_event_body` function that carries out the logic inside each iteration of the `for` loop to update `mini` and `next_event_index` as needed.
3. This function itself has an `if` statement which must be replaced by a functional

equivalent that, in turn, also requires the writing of two more functions to handle each Boolean value of the conditional.

4. Each of these functional equivalents require us to write additional functions to pass to them as arguments.
5. The `.at[]` and `.get()` notation is needed to index and retrieve values from arrays.
6. When updating `event_list` in Figure 2.1, we accessed a slice of the `event_list` that depended on the run time value `num_events`. In Figure 2.2 we use `np.where` instead.

2.5 INCORPORATING AUTOMATIC DIFFERENTIATION INTO SIMOPT

SimOpt is a Python library intended for benchmarking the finite-budget performance of simulation-optimization solvers and understanding their behavior on a variety of problems. In light of this goal, SimOpt retains a simple enough architecture so that researchers from varied backgrounds can easily contribute problems and solvers that are of interest to them. Furthermore, the motivation for incorporating AD into SimOpt is to reduce the burden on contributors by automating the implementation of IPA estimators. Thus, the decision of which AD tool to use in SimOpt is based on ease of use followed by reasonable (run time) performance. As SimOpt already heavily utilizes `NumPy`, it is convenient to choose an AD tool that includes NumPy-like function primitives. These considerations led us to consider Autograd and JAX and rule out other options such as PyTorch. We ruled out JAXJIT as an option because it would require an extensive restructuring of the SimOpt library.

```
from jax import jit
from jax.lax import cond, fori_loop
import jax.numpy as np

@jit
def get_next_event(event_list, num_events):
    mini = SIM_LENGTH + 2.0
    i = 0
    next_event_index = 0
    # loop below is to find index of next event
    _, _, next_event_index = fori_loop(
        0,
        num_events,
        get_next_event_body,
        (event_list, mini, next_event_index),
    )
    next_event = event_list.at[next_event_index].get()
    place_holder_event = (
        np.zeros(EVENT_SIZE) + SIM_LENGTH + 1
    )
    # deleting next_event from event_list
    event_list = np.where(
        np.arange(len(event_list)) < next_event_index,
        np.vstack([
            event_list[:-1], place_holder_event
        ]).T,
        np.vstack([
            event_list[1:], place_holder_event
        ]).T,
    ).T
    num_events -= 1
    return event_list, num_events, next_event
```

Figure 2.2: Event list as an unordered JAX array of JAX arrays, compatible with JAX with JIT compilation.

Between Autograd and JAXnoJIT, we chose to implement Autograd because the performance is better in two representative examples; see Section 2.6.

In the SimOpt architecture, a simulation-optimization problem is specified by two objects: the `Model` class which represents the simulation model and the `Problem` class which formulates the optimization problem built around the model. The parameters of the simulation model are stored in a dictionary attribute `Model.factors`. The `Model` class also has a `replicate()` method that returns dictionaries with stochastic values of `responses` and `gradients` as the outputs of the simulation. The `Model.replicate()` method does not have any explicit inputs but instead accesses the simulation model parameters internally through `Model.factors`. The `Problem` class defines decision variables as a function of `Model.factors` and a performance metric as a function of the `responses` returned from the `Model.replicate()` function. Currently, we have only implemented AD to calculate the `Model.replicate()` Jacobian associated with the `Model.factors` and `responses`. At this time, the Jacobian of the mapping from decision variables of `Problem` to `Model.factors` and the gradient of the mapping from `responses` to the final performance metric of `Problem` must be computed manually and multiplied with the automatically obtained Jacobian of `Model.replicate()` to complete the gradient calculation of the performance metric with respect to the decision variables. We intend to eventually extend AD to the `Problem` class so that the entire gradient calculation is handled automatically.

To enable automatic computation of the Jacobian of the `responses` with respect to the `factors`, we created a subclass of `Model` named `Auto_Model`, which must have an `Auto_model.inner_ replicate()` method. The `Auto_model.inner_replicate()` method is essentially the same as `Model.replicate()` but with a different signature and must follow the general

rules to be compatible with Autograd so that we can use Autograd to differentiate it. Thus, it requires little to no additional work for the programmer to write when compared to `Model.replicate()`. The `Auto_Model` constructor automatically creates an `Auto_model.replicate()` method by applying the `replicate_wrapper()` method that we wrote to `Auto_model.inner_replicate()`. The `replicate_wrapper()` method converts dictionaries to arrays for the inputs and vice versa for the outputs. It also calls the `value_and_jacobian()` method we wrote applied to `Auto_model.inner_replicate()` to get its output and Jacobian. The method `value_and_jacobian()` is not included in Autograd but can easily be written by modifying the code for `autograd.jacobian()` to return both the function output and Jacobian instead of just the Jacobian. We allow the calculation of a partial Jacobian with respect to only a subset of outputs, saving computation since `value_and_jacobian()` uses reverse mode whose run-time complexity scales with the number of outputs. A user can write a subclass of `Auto_Model` instead of just `Model`, requiring comparable amounts of work, to automatically get an IPA Jacobian estimate out of their simulation model when they call `replicate()` instead of having to manually code the IPA Jacobian estimate.

While incorporating Autograd into SimOpt, we inevitably encountered bugs. Most were caused by using functions or arrays from `numpy` instead of `autograd.numpy` inside of a function being differentiated. For example, the `Auto_model.inner_replicate()` function typically makes calls to SimOpt's custom RNG which is defined in a separate file from model definitions. It was necessary to change `import numpy as np` to `import autograd.numpy as np` in the file where the RNG was defined. Another source of bugs is in-place updating of arrays. These are easily fixed by using lists instead of arrays until all in-place operations are complete, at which point an array can be constructed from the list

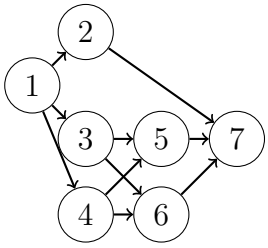
for computations if needed. Finally, it is possible for Autograd to unexpectedly return 0 as a value for a derivative when this is not the desired output. This is because Autograd may not be able to trace the output as being dependent on the inputs if the rules of compatibility with Autograd are broken. Therefore, if a value of 0 is encountered as a derivative value, it may be good practice to use finite differences to verify whether 0 is indeed the correct value.

2.6 EXPERIMENTS

We explore the run times of AD software on simulation models of a stochastic activity network (SAN) and an ambulance service. We time each method of obtaining an IPA gradient estimate using a varying number of replications by averaging over replications. AD can be applied at two places in the code: after each replication or after averaging over all replications, and we shall see that the run-time difference is large. For the SAN we include both methods, but for the ambulance system we perform only the latter. We ran JAXJIT twice for each number of replications to demonstrate the time spent on the initial program compilation versus executing the program after compilation. All AD tools used reverse mode except JAXJIT in the ambulance system where forward mode was required due to the inclusion of a `while` loop in the code. We also manually implemented IPA. For the manual implementations, we used reverse mode for the SAN and forward mode for the ambulance system. We also used forward finite differences (FFD) for both models. The timing experiments were performed in Google Colab using two Intel Xeon CPU 2.20GHz cores.

Table 2.1: SAN results, with AD applied after each replication then averaged. The entry N/A indicates that the run did not complete in reasonable time.

Reps	Manual (REV)	FFD	Autograd	JAX	JAX JIT first	JAX JIT second
1	996 μs	5.92 ms	3.46 ms	1.52 s	1.38 s	185 μs
10^2	9.27 ms	33.9 ms	283 ms	5.54 s	1.88 s	509 μs
10^4	762 ms	5.71 s	16.1 s	7 min 42 s	1.60 s	10.1 ms
10^6	33.5 s	4 min 1 s	19 min 55 s	N/A	1.48 s	646 ms



Reps	Autograd	JAX	JAX JIT 1st	JAX JIT 2nd
1	4.16 ms	2.23 s	1.71 s	6.38 ms
10^2	562 ms	8.67 s	2.81 s	7.81 ms
10^4	26.4 s	13 min 17 s	2.53 s	47.5 ms
10^6	N/A	N/A	3.26 s	680 ms

Figure 2.3: Left: The SAN we used in our experiments. Right: SAN results with AD applied after averaging all replications. N/A results are due to running out of memory.

2.6.1 Stochastic Activity Network

A SAN is a feed-forward network that models the precedence of a collection of subtasks for completing a task. Each arc represents a subtask which we assume takes an exponentially distributed amount of time to complete. Figure 2.3 depicts the specific SAN we used. The parameters of the simulation model, $x \in \mathbb{R}^{10}$, are the mean times of each subtask. The output of the simulation model is the time to complete the total task, which is the (random) length of the longest path from node 1 to node 7, plus a deterministic cost $\sum_{i=1}^{10} x_i^{-1}$ so that $h(x, Y) = Y + \sum_{i=1}^{10} x_i^{-1}$, where $Y = Y(x)$ is the length of the longest path from node 1 to 7. Table 2.1 gives the average per-replication time to obtain a derivative when AD is applied to each replication individually. Figure 2.3 gives the corresponding results when AD is applied once to the entire body of replications collectively.

Table 2.2: Ambulance system timing results, with AD applied after each replication and then averaged.

Reps	Manual (FWD)	FFD	Autograd	JAX	JAX JIT 1st (FWD)	JAX JIT 2nd (FWD)
1	24.2 ms	46.8 ms	61.6 ms	3.08 s	4.86 s	23 ms
10^1	100 ms	552 ms	643 ms	9.01 s	6.60 s	168 ms
10^2	837 ms	4.19 s	4.53 s	40.7 s	5.26 s	1.57 s
10^3	9.82 s	30.2 s	30.5 s	4 min 59 s	19.5 s	15.8 s

2.6.2 Ambulance System

We use the model described in Eckman and Henderson (2020b) but with modified parameters. The following model description is taken, almost verbatim, from that reference. Calls arrive according to a Poisson process in time at rate 1/2.5. Call locations are uniformly distributed in a square with side length 20. The closest free ambulance is dispatched to the call, and travels to the call in Manhattan fashion (traveling first in the x direction, followed by the y direction) at a constant speed of 1. The ambulance then spends a random amount of time at the scene of the call (exponentially distributed with mean 10) and afterwards returns to its base in Manhattan fashion. For simplicity, hospitals are not modeled, and ambulances returning to base are not diverted to new calls. Calls arising when all ambulances are busy are queued and answered in first-in-first-out order. Call arrivals, locations, and scene times are mutually independent. The simulation starts from both ambulances being available and at base, with no queued calls. The variable x specifies the locations of the two ambulance bases, which are located at $(15, 15)$ and $(5, 5)$; thus, $x = (15, 15, 5, 5)$. The output $h(x, Y)$ is the sum of the ambulance response times to calls over a time interval of length 100. This simulation uses an event list and has a while loop that keeps the simulation running until time 100. Timing results for AD on this ambulance system simulation are reported in Table 2.2.

2.7 DISCUSSION AND CONCLUSION

The SAN experiment suggests that it is beneficial to apply AD after each replication rather than after averaging all replications due to faster run times and lower memory requirements. Moreover, access to per-replication IPA estimates allows the computation of sample variances of the IPA estimates. The memory issue when applying AD after averaging all replications is expected, because the number of intermediate variables stored for the backwards pass then scales linearly with the number of replications. We did not run into memory issues when AD was applied to each replication. Autograd appears to run faster than JAXnoJIT. JAXJIT can provide substantial speedups over Autograd on larger scale tasks but requires significant effort and care to restructure the code to be fully functionally oriented with static-sized intermediate variables. JAXJIT is limited to forward mode when a while loop is present in the code, as in many event-list based simulations. This may narrow the run-time gap when compared to Autograd in such simulations as seen in the ambulance example. The compile time of JAXJIT dominates the actual program execution time for small programs. The run times when JAX is run a second time after already applying JAXJIT from the first run are faster than the first time because the overhead for JIT compilation is not repeated.

The manual method was faster than the AD methods in all comparisons, except with JAXJIT on the SAN example for large numbers of replications. JAXJIT was faster than FFD for large numbers of replications. Autograd was slower than FFD on the SAN problem, but typically on the same order of magnitude, whereas the two methods had similar run times on the ambulance problem. Our experiments were not designed to show the scaling of the run times with the dimension, d, however, FFD should scale linearly with d because $d + 1$ simulation runs are needed to estimate the gradient. We expect that reverse-mode manual and AD methods will

remain approximately constant in d. Thus, for larger d we expect that Autograd will be faster than FFD. We did not use GPU/TPU support or the `vmap` or `pmap` functions of JAX, which could significantly speed up JAX and JAXJIT for some models.

AD can give yield IPA estimates on non-trivial simulations without requiring manual derivation and coding of the IPA estimator, with a competitive run time compared to FFD. In addition to saving analyst effort, getting an IPA estimate from AD is likely less error prone than manually deriving and implementing the IPA estimator. The use of existing AD tools requires some care in coding, but for many models is straightforward to implement. Additionally, IPA has different bias and variance properties than FFD which may be advantageous in some settings. We look forward to developments in AD tools that improve their utility in simulation applications, perhaps through careful consideration of common flow structures in discrete-event simulation models.

CHAPTER 3

BATCH SYNTHESIS IN HIGH-DIMENSIONAL CLASSIFICATION ACTIVE LEARNING VIA LATENT GRADIENT OPTIMIZATION

3.1 INTRODUCTION

In this work we consider the problem of active learning (AL) for challenging high-dimensional classification problems (e.g., image, text, and audio classification). Recently, high-capacity models, especially deep neural networks (DNNs), have demonstrated tremendous success for such problems at the cost of much higher training-data complexity. This has made AL in this setting an important emerging area of research (e.g., Ren et al., 2021). The majority of existing AL methods for this setting are pool-based: they select data to label from a large but finite collection of *real-world* instances termed the pool. Although potentially effective, this approach restricts the search to samples within the pool, which may be extremely sparse in high-dimensional problems and severely limit the candidates available for labeling.

Query synthesis (QS) methods aim to overcome the limitations of pool-based AL by searching the entire input space and synthesizing whichever instances are most informative. Although QS offers significant potential advantages, especially in high-dimensional settings, it is also more challenging to implement. In general, AL begins by training the model on a small subset of labeled training data, after which an additional batch of data points is chosen for labeling according to an *acquisition function*. The acquisition function maps unlabeled instances to scalar values indicating their relative informativeness. Searching high-dimensional spaces for instances that maximize an arbitrary acquisition function is challenging; direct search can quickly become computationally intractable, necessitating approximations that may render the search ineffective. These problems are exacerbated

when synthesizing batches rather than individual instances, as is often needed for practical AL with neural networks. Another major obstacle to high-dimensional QS is that the data may lie on a manifold that is difficult to estimate, resulting in unrealistic or implausible synthesized queries (Baum and Lang, 1992).

Chen et al. (2021) jointly optimize batches with respect to an acquisition function in a learned latent space and decode the resulting representations into high-dimensional inputs. The decoder constrains synthesized points to its learned range and lowers the dimension of the search, but it does not guarantee realism or unambiguous labels. Their acquisition criterion is not mutual-information based. Schumann and Rehbein (2019) and Huijser and van Gemert (2017) also synthesize queries through learned latent representations, using non-information-based criteria and classifiers operating in latent space. Kong et al. (2019) train a generative adversarial network (Goodfellow et al., 2014) to balance data modeling and informativeness rather than directly optimizing a fixed batch acquisition objective. Bayesian generative active deep learning combines pool-based mutual-information acquisition with generative augmentation of selected observations (Tran et al., 2019).

Mutual-information acquisition can also be viewed as Bayesian experimental design. Foster et al. (2020) develop gradient estimators for information-based experimental-design objectives, but not a high-dimensional batch query-synthesis method constrained by a generative decoder. The intended LGO-BatchBALD framework combines three elements: a joint batch formulation, a generative decoder constraint, and the BatchBALD mutual-information objective for a classifier operating in data space. Table 3.1 summarizes this positioning; it describes the intended formulations rather than validating every optimization implementation.

	Batch	Decoder	Direct opt.	Info.	Data-space DNN
LGO-BatchBALD	yes	yes	yes	yes	yes
Foster et al. (2020)	no	no	yes	yes	yes
Tran et al. (2019)	no	yes	no	yes	yes
Schumann and Rehbein (2019)	no	yes	yes	no	no
Huijser and van Gemert (2017)	no	yes	yes	no	no
Chen et al. (2021)	yes	yes	yes	no	yes

Table 3.1: Comparison of query-synthesis formulations. "Decoder" indicates synthesis through a learned generative representation, not guaranteed realism. "Direct opt." indicates direct optimization of the stated acquisition objective, and "Info." indicates a mutual-information objective. The LGO-BatchBALD row describes the intended framework.

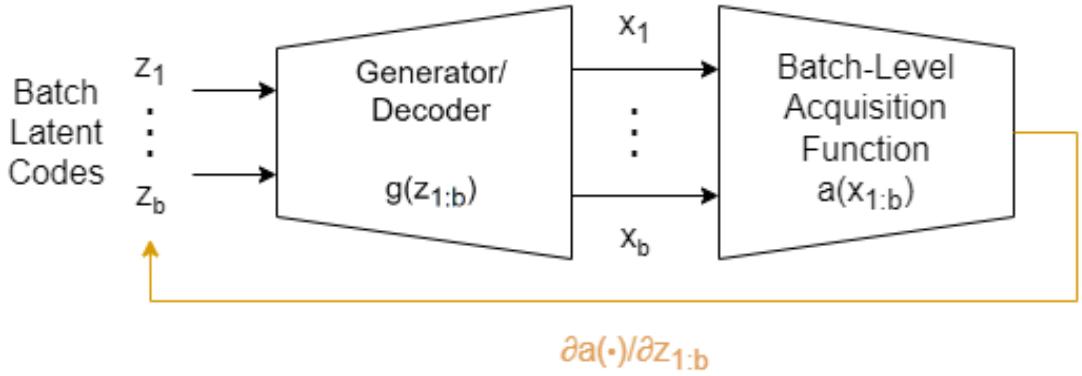


Figure 3.1: Illustration of LGO applied to QS

3.1.1 Contributions

This chapter makes two contributions: a formulation for high-dimensional batch query synthesis and a lower-variance estimator of its acquisition function.

Decoder-constrained synthesis. We formulate LGO-BatchBALD, which jointly searches for a batch in the latent space of a differentiable generative model while evaluating the BatchBALD mutual-information objective using a classifier in data space. The contribution is the combination of these elements. The original BatchBALD method selects greedily from a fixed pool and therefore cannot propose potentially more informative points that are absent from that pool. Previous

latent-space query-synthesis methods permit continuous search through a learned representation, including joint batch search, but do not directly optimize a mutual-information acquisition function. Conversely, previous gradient-based methods for information acquisition do not combine joint high-dimensional batch synthesis with a generative decoder constraint. Among the formulations summarized in Table 3.1, LGO-BatchBALD is the only one combining joint batch search, decoder-constrained synthesis, direct optimization of a mutual-information objective, and evaluation by a data-space classifier.

This combination is important in high-dimensional settings, where a finite pool may sparsely cover the input space and direct optimization over raw inputs may be computationally difficult or produce unnatural queries. Searching through the decoder's lower-dimensional latent space expands the candidate set beyond the observed pool, making it possible to search for potentially more informative points that the pool does not contain. Constraining the search to the decoder's learned range may also yield more realistic or natural-looking queries than unconstrained raw-input optimization. Jointly optimizing the batch allows the acquisition function to account for dependence and redundancy among proposed labels. These benefits are not guaranteed: the decoder range may still contain unrealistic or ambiguously labelable points, and the experiments in this chapter do not validate the end-to-end optimization procedure. Figure 3.1 illustrates the framework, and Figure 3.2 provides an overview of LGO-BatchBALD.

Paired estimator of BatchBALD. We also propose a new Monte Carlo estimator of the BatchBALD acquisition value. The existing analytic-entropy estimator computes conditional entropy analytically for each sampled model. Our estimator instead uses a sampled conditional log likelihood, evaluated on the same model-label draw used in the posterior-predictive term. Equivalently, it adds a mean-zero

control variate with coefficient one to the analytic-entropy estimator. The two estimators therefore have exactly the same finite-sample expectation and the same downward finite-sample bias relative to exact BatchBALD; the proposed estimator changes the variance, not the estimand.

The shared randomness allows fluctuations in the conditional and posterior-predictive terms to cancel. In the fixed-batch experiments, this substantially reduces variance for the same number of model draws. This is useful because query synthesis requires repeated comparisons of acquisition values across candidate batches: a lower-variance estimate can provide more precise comparisons at a fixed computational budget or comparable precision with fewer model evaluations. Because the control-variate coefficient is fixed at one rather than optimized, variance reduction is demonstrated empirically and is not guaranteed for every model and candidate batch.

Implementing LGO-BatchBALD also requires a valid gradient estimator. Differentiating a realized estimator while holding its sampled discrete labels fixed omits how their distribution changes with the candidate batch and therefore cannot generally be assumed unbiased. The formulation and acquisition-value estimator are the contributions established here; developing and validating an end-to-end optimizer remains future work.

3.1.2 Organization of Paper

We first introduce the necessary background in Section 3.2. In Section 3.3, we present LGO-BatchBALD, our proposed BatchBALD estimator, and the gradient-estimation issue. Section 3.4 validates acquisition-value estimation. Section 3.5 discusses limitations and future work, and Section 3.6 concludes.

3.2 BACKGROUND AND ASSUMPTIONS

3.2.1 Bayesian-Like Classification Models

We use x to denote an image and Y^x to represent the random variable modeling the unknown label of image x. Similarly, $x_{1:b}$ and $Y_{1:b}^{x_{1:b}}$ refer to a batch of b images and the corresponding vector of random variables for their labels.

To quantify epistemic uncertainty for estimating the BatchBALD score, we use a Bayesian-like classification model with uncertain parameters $\omega \sim p_{\omega|\mathcal{D}_{\text{labeled}}}(\cdot)$, where $\mathcal{D}_{\text{labeled}}$ contains the labeled training images and their labels. Conditioning on $\mathcal{D}_{\text{labeled}} = \{(x_1, y_1), (x_2, y_2), \ldots\}$ represents training the classifier on these data. For brevity, we drop this conditioning below. Let $\mathcal{D}_{\text{train}}$ denote the full training set, which may contain unlabeled images. Initially, $\mathcal{D}_{\text{labeled}} \subset \mathcal{D}_{\text{train}}$, but synthesized images are labeled and added to $\mathcal{D}_{\text{labeled}}$. The model outputs class probabilities $p(Y^x = c \mid \omega)$, one for each class c, that are differentiable with respect to x. Assuming label independence given ω, $p(y_{1:b}^{x_{1:b}} \mid \omega) = \prod_{i=1}^{b} p(y_i^{x_i} \mid \omega)$. The model can be any differentiable Bayesian classifier or ensemble neural network (ENN).

Ensemble Neural Networks (ENNs)

ENNs generalize Bayesian neural networks (BNNs) and allow for quantification of epistemic uncertainty. Like BNNs, ENNs comprise ensembles with an uncertainty index ω analogous to uncertain parameters in a Bayesian model. Suppose there are C classes. Our ENN has a base NN with feature extractor $\phi(x)$ and class-logit vector $s_{\text{base}}(x) \in \mathbb{R}^C$, and an epinet taking $\phi(x)$ and ω as input and outputting a class-logit vector $s_{\text{epi}}(\phi(x), \omega) \in \mathbb{R}^C$. The combined logit vector for one ENN realization is the elementwise sum

$$\ell(x, \omega) = s_{\text{base}}(x) + s_{\text{epi}}(\phi(x), \omega).$$

For each class $c \in \{1, \ldots, C\}$, the corresponding probability is

$$p(Y^x = c \mid \omega) = [\text{softmax}(\ell(x, \omega))]_c = \frac{\exp(\ell_c(x, \omega))}{\sum_{j=1}^{C} \exp(\ell_j(x, \omega))}.$$

Thus the softmax has one vector argument containing C class logits; its denominator normalizes across all classes. No temperature rescaling is used here; equivalently, the softmax temperature is one. We use the ENN with an epinet because it has lower computational costs than a BNN (see Osband et al., 2021, for details on ENNs).

3.2.2 BatchBALD Acquisition Function

The BatchBALD acquisition function (score)

$$a(x_{1:b}) = \mathbb{H}[Y_{1:b}^{x_{1:b}}] - \mathbb{E}\mathbb{H}[Y_{1:b}^{x_{1:b}} \mid \omega] \tag{3.1}$$

measures the mutual information between the predicted labels $Y_{1:b}^{x_{1:b}}$ and model parameters ω, where $\mathbb{H}$ represents entropy so that

$$\mathbb{H}[Y_{1:b}^{x_{1:b}}] = \mathbb{E}[-\log\left(p(y_{1:b}^{x_{1:b}})\right)]$$

and

$$\mathbb{H}[Y_{1:b}^{x_{1:b}} \mid \omega] = \mathbb{E}[-\log\left(p(y_{1:b}^{x_{1:b}} \mid \omega)\right)].$$

The first term in (3.1) captures total label uncertainty. The second term captures aleatoric uncertainty or expected inherent label entropy under the true ω. Thus, $a(\cdot)$ is high when each ω makes confident but differing predictions. Computing $a(\cdot)$ requires quantifying epistemic uncertainty and motivates our use of classification models with this capability.

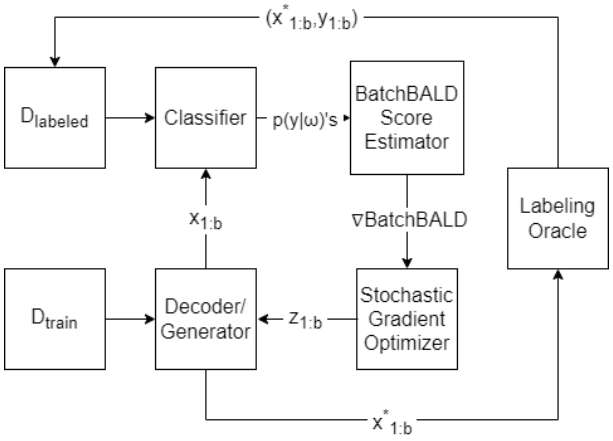


Figure 3.2: Overview of LGO-BatchBALD

3.2.3 Generative Models

We also assume a differentiable generative model $g(\cdot)$ trained on $\mathcal{D}_{\text{train}}$, which can generate a data point $x = g(z)$ or a batch $x_{1:b} = g(z_{1:b})$, where $z \sim p_Z(\cdot)$. Viable options include generative adversarial networks (GANs) and variational autoencoders (VAEs) (Kingma and Welling, 2013). We do not recommend existing diffusion models because our methodology requires repeated inference and backpropagation, which could become a computational bottleneck given the large number of time steps diffusion models tend to require in practice.

3.3 METHODOLOGY

3.3.1 LGO-BatchBALD: QS via LGO

In this section, we introduce LGO-BatchBALD, which performs QS through optimization. Figure 3.2 provides an overview of our approach. Our objective is to maximize the BatchBALD acquisition function with respect to the batch of images, as formulated in (3.2). Instead of directly optimizing the raw images $x_{1:b}$, we optimize within the latent space of a generative model to obtain latent variables

$z^*_{1:b}$. We then transform these variables into images $x^*_{1:b} = g(z^*_{1:b})$. This reparameterization constrains the generated batch to the range of the decoder. It may improve plausibility relative to unconstrained data-space optimization, although it does not guarantee realistic or unambiguously labelable images.

In contrast to greedy methods that add one image at a time, such as BatchBALD (Kirsch et al., 2019), we formulate simultaneous optimization over all images in the batch. Joint batch-level optimization may yield better batches, although we do not validate that claim here. We use stochastic gradient optimization methods to search for latent variables

$$z^*_{1:b} = \underset{z_{1:b} \in \mathbb{R}^{b \times d}}{\arg\max} \mathbb{H}[Y^{g(z_{1:b})}_{1:b}] - \mathbb{E}\mathbb{H}[Y^{g(z_{1:b})}_{1:b} \mid \omega]. \tag{3.2}$$

3.3.2 Estimating BatchBALD and Its Gradient

In this subsection, we omit the superscript $x_{1:b}$ on the random vector of class labels Y and use superscripts to index sampled label configurations. Fix a candidate batch $x_{1:b}$. Draw

$$\begin{aligned} \omega_1, \ldots, \omega_m &\overset{\text{iid}}{\sim} p_\omega(\cdot), \\ Y^t_{1:b} &\sim p(\cdot \mid \omega_t), \quad t = 1, \ldots, m, \end{aligned} \tag{3.3}$$

and estimate the posterior predictive probability of a label configuration by

$$\widehat{p}_m(y) := \frac{1}{m} \sum_{i=1}^{m} p(y \mid \omega_i). \tag{3.4}$$

Thus, the same sampled parameter values $\omega_1, \ldots, \omega_m$ are used to generate the sampled labels and to estimate their posterior predictive probabilities.

The *analytic-entropy estimator* of BatchBALD is

$$\widehat{a}^{\text{AE}}_m(x) = -\frac{1}{m} \sum_{t=1}^{m} \log \widehat{p}_m(Y^t_{1:b}) - \frac{1}{m} \sum_{t=1}^{m} \mathbb{H}[Y_{1:b} \mid \omega_t]. \tag{3.5}$$

Its first term is a nested Monte Carlo estimate of predictive entropy, while its second term computes conditional entropy analytically for each sampled model.

Our *proposed estimator*, defined below in Equation (3.6), replaces the analytic conditional-entropy term with a sampled log likelihood:

$$\widehat{a}_m^{\text{prop}}(x) = \frac{1}{m}\sum_{t=1}^{m}\left[\log p(Y_{1:b}^t \mid \omega_t) - \log \widehat{p}_m(Y_{1:b}^t)\right]. \tag{3.6}$$

Each summand is a sampled log-likelihood ratio for the same joint label configuration $Y_{1:b}^t$. The first probability, $p(Y_{1:b}^t \mid \omega_t)$, is assigned by the model that generated that configuration. The second, $\widehat{p}_m(Y_{1:b}^t)$, averages the probabilities assigned by all m sampled models. Pairing matters because, conditional on ω_t, the expected value of $\log p(Y_{1:b}^t \mid \omega_t)$ is exactly the negative conditional entropy under that model. The proposed estimator therefore replaces the analytic conditional-entropy term with a Monte Carlo estimate of the same quantity rather than changing the target.

The same sampled label configuration appears in the predictive and conditional terms. Their Monte Carlo fluctuations can consequently move together and partially cancel when the terms are subtracted. This shared randomness is the source of the potential variance reduction. It is internal to one estimator evaluation and does not imply that the analytic-entropy and proposed estimators used common random numbers in our experiment.

A control variate is a mean-zero random variable added to an estimator to reduce variance without changing its expectation (Glasserman, 2004). Here, the difference between Estimators (3.6) and (3.5) is the control variate

$$C_m = \frac{1}{m}\sum_{t=1}^{m}\left[\mathbb{H}[Y_{1:b} \mid \omega_t] + \log p(Y_{1:b}^t \mid \omega_t)\right]. \tag{3.7}$$

Conditional on the sampled models, $\mathbb{E}[C_m \mid \omega_{1:m}] = 0$. The more general control-variate family

$$\widehat{a}_m(\lambda) = \widehat{a}_m^{\text{AE}} + \lambda C_m \tag{3.8}$$

therefore has the same finite-sample expectation as $\widehat{a}_m^{\text{AE}}$ for every fixed λ.

When $\mathrm{Var}(C_m) > 0$, the variance-minimizing population coefficient is

$$\lambda^* = -\frac{\mathrm{Cov}(\widehat{a}_m^{\mathrm{AE}}, C_m)}{\mathrm{Var}(C_m)}. \tag{3.9}$$

Using the exact λ^* gives variance no greater than that of the analytic-entropy estimator, with strict improvement when the covariance is nonzero. In practice, obtaining λ^* would require estimating these potentially batch-dependent moments. If the coefficient is estimated from the same samples, the population variance guarantee does not automatically carry over without additional care.

Our proposed estimator fixes $\lambda = 1$. Substituting the definitions shows why this choice is structurally convenient:

$$\begin{aligned}
\widehat{a}_m^{\mathrm{AE}} + C_m &= -\frac{1}{m}\sum_{t=1}^{m} \log \widehat{p}_m(Y_{1:b}^t) - \frac{1}{m}\sum_{t=1}^{m} \mathbb{H}[Y_{1:b} \mid \omega_t] \\
&\quad + \frac{1}{m}\sum_{t=1}^{m} \left[\mathbb{H}[Y_{1:b} \mid \omega_t] + \log p(Y_{1:b}^t \mid \omega_t)\right] \\
&= \frac{1}{m}\sum_{t=1}^{m} \left[\log p(Y_{1:b}^t \mid \omega_t) - \log \widehat{p}_m(Y_{1:b}^t)\right].
\end{aligned}$$

Thus, setting $\lambda = 1$ exactly cancels the analytic conditional-entropy terms. What remains is a paired log-likelihood ratio: each sampled configuration $Y_{1:b}^t$ is evaluated both under the model ω_t that generated it and under the Monte Carlo posterior predictive distribution $\widehat{p}_m$. Fixing the coefficient at one avoids estimating the covariance and variance needed for λ^*, but it sacrifices the general variance-reduction guarantee. The proposed and analytic-entropy estimators nevertheless have exactly the same finite-sample expectation. Their common expectation is generally a lower bound on BatchBALD at finite m, and both estimators converge to BatchBALD under the support conditions given in Appendix 3.6.1.

Algorithm 2 exposes the estimators' parallel structure. They use the same sampled parameter values $\omega_1, \ldots, \omega_m$, sampled labels, and predictive log probabilities; they differ only in whether the conditional-entropy contribution is computed

Algorithm 1 LGO-BatchBALD

$z_{1:b} \sim p_{Z_{1:b}}(\cdot)$
for $t = 1, \ldots, n$ **do**
    $x_{1:b} \leftarrow g(z_{1:b})$
    $\widehat{a}(z_{1:b}) \leftarrow$ Proposed output of Algorithm 2
    $\widehat{\nabla}_z \leftarrow$ Valid_Gradient_Estimate$(\widehat{a}(\cdot), z_{1:b})$
    $z_{1:b} \leftarrow$ Gradient_Step$(z_{1:b}, \widehat{\nabla}_z)$
end for
return $x^*_{1:b} = g(z_{1:b})$

Algorithm 2 Analytic-Entropy and Proposed Estimators (3.5)–(3.6)

Require: Candidate batch $x_{1:b}$ and sample size m
for $t = 1, \ldots, m$ **do**
    $\omega_t \sim p_\omega(\cdot)$
    $y^t_{1:b} \sim p(\cdot \mid \omega_t)$
    $h_t \leftarrow \mathbb{H}[Y_{1:b} \mid \omega_t]$
    $r_t \leftarrow \log p(y^t_{1:b} \mid \omega_t)$
end for
for $t = 1, \ldots, m$ **do**
    $\ell_t \leftarrow \text{LogSumExp}_{i=1:m}\{\log p(y^t_{1:b} \mid \omega_i)\} - \log m$
end for
$\widehat{a}^{\text{AE}}_m \leftarrow \frac{1}{m}\sum_{t=1}^m (-\ell_t - h_t)$
$\widehat{a}^{\text{prop}}_m \leftarrow \frac{1}{m}\sum_{t=1}^m (r_t - \ell_t)$
return $\widehat{a}^{\text{AE}}_m, \widehat{a}^{\text{prop}}_m$

analytically as $-h_t$ or sampled as r_t. For numerical stability, we evaluate each predictive log probability $\ell_t = \log \widehat{p}_m(y^t_{1:b})$ in log space. For the sampled label vector $y^t_{1:b}$ and model draw ω_i, conditional label independence gives

$$a_{ti} := \log p(y^t_{1:b} \mid \omega_i) = \sum_{j=1}^{b} \log p(y^t_j \mid \omega_i),$$

so the b potentially small label probabilities are not multiplied directly. We then compute the logarithm of their average across the m model draws using the max-shifted log-sum-exp formula

$$\ell_t = c_t + \log\left(\sum_{i=1}^{m} \exp(a_{ti} - c_t)\right) - \log m, \qquad c_t = \max_{1 \le i \le m} a_{ti}.$$

Because $a_{ti} - c_t \le 0$, the exponentiated quantities are at most one. This avoids directly forming either the product of the batch-label probabilities or their potentially very small predictive average.

Gradient estimation. Let R collect the sampled parameter values $\omega_1, \ldots, \omega_m$ and sampled labels used by the estimator, and let $q_x(R)$ denote their joint distribution at candidate batch x. Define the expected finite-sample objective

$$\overline{a}_m(x) := \mathbb{E}_{R \sim q_x}[\widehat{a}^{\text{prop}}_m(x, R)].$$

Suppose q_x has support that does not change locally with x, both $q_x(R)$ and $\widehat{a}^{\text{prop}}_m(x, R)$ are differentiable in x, and their relevant derivatives are bounded by an integrable envelope so that differentiation and expectation can be interchanged. The likelihood-ratio identity for a parameter-dependent integrand then gives the following gradient formula (Asmussen and Glynn, 2007, Remark 3.7, pp. 223–224):

$$\begin{aligned} \nabla_x \overline{a}_m(x) = \mathbb{E}\,[&\nabla_x \widehat{a}^{\text{prop}}_m(x, R) \\ &+ \widehat{a}^{\text{prop}}_m(x, R) \nabla_x \log q_x(R)]\,. \end{aligned} \tag{3.10}$$

Thus, the random quantity inside the expectation in Equation (3.10) is an unbiased estimator of $\nabla_x \overline{a}_m(x)$, the gradient of the expected finite-m objective. At finite m, it is generally biased for the gradient of exact BatchBALD, $\nabla_x a(x)$, because $\overline{a}_m(x) \neq a(x)$ in general at finite m. Appendix 3.6.1 establishes convergence of the acquisition values as $m \to \infty$, but convergence of the values alone does not establish convergence of their gradients.

The sampled parameter values $\omega_1, \ldots, \omega_m$ do not depend on the candidate batch, but the sampled labels do, giving

$$\nabla_x \log q_x(R) = \sum_{t=1}^{m} \nabla_x \log p(Y_{1:b}^t \mid \omega_t). \tag{3.11}$$

The IPA derivative that treats sampled labels as fixed retains only the first term in Equation (3.10). Intuitively, changing x changes both the estimator evaluated at a given label configuration and the probability that each configuration is sampled. IPA captures the first effect but not the second. Its bias for $\nabla_x \overline{a}_m(x)$ is therefore the negative of the omitted likelihood-ratio expectation. That expectation may vanish in special cases, but no such cancellation is established here, so the IPA derivative cannot be presumed unbiased; see Asmussen and Glynn (2007, Sections VII.2–VII.3, pp. 214–224) and Eckman and Henderson (2020a). Finite differences can approximate the finite-m gradient, or a separately derived estimator can target it directly. Estimating $\nabla_x a(x)$ instead requires exact or high-accuracy acquisition evaluations, or additional conditions establishing that $\nabla_x \overline{a}_m(x) \to \nabla_x a(x)$.

3.3.3 Complexity

Evaluating all $p(Y_{1:b}^t \mid \omega_i)$ for $t, i = 1, \ldots, m$ requires m^2 model–configuration comparisons, each involving b batch elements. The direct arithmetic complexity of

our proposed estimator in Equation (3.6) is therefore

$$\mathcal{O}(bm^2), \tag{3.12}$$

apart from model forward-pass costs and hardware vectorization. Although this arithmetic cost is linear in b, likelihood evaluations for the b batch elements can be issued as batched accelerator operations. Wall-clock time may therefore grow more slowly than the operation count while the batch fits in device memory; we do not claim asymptotic independence from batch size.

For a C-class problem, a batch has C^b possible joint label configurations. Exact evaluation of the predictive-entropy term would sum over this space. Our proposed estimator instead samples joint configurations $Y_{1:b}$, avoiding explicit enumeration but not eliminating the possibility that the required Monte Carlo sample size grows with b. Greedy pool-based BatchBALD adds one candidate at a time and evaluates its incremental contribution, rather than searching over all b points jointly; nevertheless, its representation or sampling of labels for the selected partial batch can also grow with the batch. A nongreedy pool method that evaluates complete candidate batches faces both the joint-label configuration space and the combinatorial choice of a batch from the pool (Kirsch et al., 2019).

Unlike pool-based BatchBALD, score evaluation for one synthesized batch has no explicit dependence on the pool size. This score-evaluation complexity is not an end-to-end guarantee for valid optimization. Finite differences require multiple score evaluations per update, while a new gradient estimator may require additional samples for variance control. The costs of decoder evaluation and generative-model training are also excluded.

3.4 COMPUTATIONAL EXPERIMENTS

We validate the acquisition-value estimators on a fixed MNIST batch using a fully connected ensemble neural network (ENN). Experiments were conducted in PyTorch (Paszke et al., 2017) on an NVIDIA GeForce RTX 2070 Max-Q GPU, where applicable, and an Intel Core i7-9750H CPU at 2.60 GHz. Additional details are provided in Appendix 3.6.2.

3.4.1 Estimator Validation

We first consider an exactly enumerable example in which the random parameter has two possible values, $\omega \in \{1, 2\}$, there are two labels, and $m = 2$. Exact enumeration confirms that the two estimators have equal means throughout the explored parameter grid. The proposed estimator often has lower variance, sometimes substantially, although it does not dominate everywhere. Full results are given in Figure 3.3 in Appendix 3.6.2. The largest reductions occur when the label distributions conditional on the two possible values of ω are similar but not identical. Cells with ratios below one provide concrete counterexamples to universal variance reduction, illustrating the covariance dependence in Equation (3.17).

We compare repeated evaluations of the analytic-entropy estimator (Equation (3.5)) and our proposed estimator (Equation (3.6)) at the same fixed MNIST batch. Table 3.2 reports results for our proposed estimator using the construction in Equation (3.3), batch size $b = 4$, Monte Carlo size $m = 10$, and 1000 replications. Within each evaluation, every sampled label is evaluated under the model that generated it; this does not assert that the two estimators shared common random numbers. The two estimators have similar observed means, as predicted by Equation (3.16), while our proposed estimator has lower variance in both evaluated classifier states.

Training iterations	AE mean	Proposed mean	AE variance	Proposed variance	Variance ratio
1	0.0100	0.0095	0.0082	0.0021	3.9
20	0.0142	0.0115	0.1273	0.0024	53.0

Table 3.2: Fixed-batch MNIST score estimates with $b = 4$, $m = 10$, and 1000 replications. AE denotes the analytic-entropy estimator. The variance ratio is AE variance divided by proposed-estimator variance, so values above one favor the proposed estimator. Training iterations 1 and 20 correspond to test accuracies 0.1812 and 0.5985. The results provide descriptive evidence at these batches, not universal dominance.

The finite differences between the observed means are attributable to Monte Carlo error. The variance reductions support the proposed control-variate construction in these settings, but Equation (3.17) does not guarantee lower variance for every model and batch.

3.5 LIMITATIONS AND FUTURE WORK

Generative Model Our method relies on a generative model of sufficient quality to constrain synthesized batches to examples that can be labeled. This requires additional computation and enough, potentially unlabeled, data to train the model. The decoder range does not guarantee realistic or unambiguous queries, particularly when optimization moves latent variables outside regions well represented during training. Future work could constrain or regularize the latent search region and explicitly model ambiguous labeling feedback.

Memory The method as presented with simultaneous optimization over each data point in the batch could be memory intensive in some cases. Computationally expensive generative models, such as diffusion models with a large number of time steps, may be prohibitive in some cases due to the storage of intermediate values needed for backpropagation. In application, memory requirements could

be decoupled from batch size by greedily constructing the batch by synthesizing chunks of data points at a time, analogous to greedy batch selection (Kirsch et al., 2019).

Estimator As explained after Equation (3.16), the proposed estimator is biased downward at finite m because it uses the finite-sample posterior predictive mixture $\widehat{p}_m$ inside the logarithmic entropy term. This plug-in bias vanishes under the conditions in Appendix 3.6.1. Its control variate reduced variance in the validated fixed-batch experiments, but the fixed choice $\lambda = 1$ is not guaranteed to reduce variance for every model and batch; the result depends on the covariance in Equation (3.17).

Optimization The acquisition surface may be complex and nonconvex, and stochastic optimization introduces hyperparameters such as learning rates and termination conditions. More importantly, the IPA derivative that holds sampled labels fixed omits the likelihood-ratio term in Equation (3.10). It is unbiased for the finite-m objective gradient only if the omitted term has expectation zero, which we do not establish. A likelihood-ratio or finite-difference procedure could instead target that gradient, but the resulting optimizer would still optimize a finite-m lower-bound objective. Establishing convergence to the exact BatchBALD gradient requires additional assumptions and analysis. End-to-end comparisons should therefore report a clearly specified optimization target and evaluate methods at matched acquisition quality.

Other Applications While our experiments focused on image data, we expect our active learning methodology to be useful on other types of data, including, but not limited to, audio and video. Further, LGO may be useful in tasks of generating high dimensional data points lying on some manifold, quite apart from the present

setting of query synthesis for active learning. We plan to explore how LGO may be used in counterfactual analysis for models dealing with such data.

3.6 CONCLUSIONS

We formulated LGO-BatchBALD as a framework for joint batch synthesis in the latent space of a generative model. Our proposed estimator (Equation (3.6)) has the same finite-sample expectation as the analytic-entropy estimator, is asymptotically unbiased and consistent under the support conditions in Appendix 3.6.1, and had lower variance in the validated fixed-batch MNIST experiments.

Valid end-to-end optimization remains future work. A gradient estimator that accounts for the dependence of sampled labels on the candidate batch can target the expected finite-m objective, but we have not established that its gradient converges to the exact BatchBALD gradient. Any future comparison with pool-based methods should state which objective is optimized and evaluate final batches at matched acquisition quality.

Appendices to Chapter 3

3.6.1 Theory for the Proposed Estimator

Control-Variate Identity

For a fixed candidate batch, the analytic-entropy estimator in Equation (3.5) and our proposed estimator in Equation (3.6) satisfy the exact samplewise identity

$$\widehat{a}_m^{\text{prop}} = \widehat{a}_m^{\text{AE}} + C_m, \tag{3.13}$$

where C_m is defined in Equation (3.7). Conditional on $\omega_{1:m}$, each sampled label satisfies

$$\begin{aligned} &\mathbb{E}_{Y_{1:b}^t \sim p(\cdot \mid \omega_t)} \left[\mathbb{H}[Y_{1:b} \mid \omega_t] + \log p(Y_{1:b}^t \mid \omega_t) \right] \\ &\quad = \mathbb{H}[Y_{1:b} \mid \omega_t] + \sum_y p(y \mid \omega_t) \log p(y \mid \omega_t) = 0. \end{aligned} \tag{3.14}$$

Linearity of conditional expectation therefore gives

$$\mathbb{E}[C_m \mid \omega_{1:m}] = 0, \tag{3.15}$$

and iterated expectation gives $\mathbb{E}[C_m] = 0$. Consequently,

$$\mathbb{E}[\widehat{a}_m^{\text{prop}}] = \mathbb{E}[\widehat{a}_m^{\text{AE}}]. \tag{3.16}$$

Despite this equality, both estimators are biased downward for exact BatchBALD at finite m. Conditional on $\Omega = (\omega_1, \ldots, \omega_m)$, their expected negative-log term is $\mathbb{H}_{\widehat{p}_m}[Y]$, because the mixture of the m conditional label distributions is $\widehat{p}_m$. Their common expectation is therefore $\mathbb{E}[\mathbb{H}_{\widehat{p}_m}[Y]] - \mathbb{E}[\mathbb{H}[Y \mid \omega]]$, rather than the exact value $\mathbb{H}_p[Y] - \mathbb{E}[\mathbb{H}[Y \mid \omega]]$. Concavity of entropy and $\mathbb{E}[\widehat{p}_m] = p$ imply $\mathbb{E}[\mathbb{H}_{\widehat{p}_m}[Y]] \leq \mathbb{H}_p[Y]$. The bias thus comes from replacing p by $\widehat{p}_m$ inside the logarithmic entropy term, not from the mean-zero control variate.

For the general family in Equation (3.8), the variance identity is

$$\begin{aligned}\operatorname{Var}(\widehat{a}_m(\lambda)) = \operatorname{Var}(\widehat{a}_m^{\mathrm{AE}}) + \lambda^2 \operatorname{Var}(C_m) \\ + 2\lambda \operatorname{Cov}(\widehat{a}_m^{\mathrm{AE}}, C_m)\end{aligned} \tag{3.17}$$

and minimizing this quadratic gives Equation (3.9). The exact population coefficient λ^* therefore gives variance no greater than the choice $\lambda = 0$, which is the analytic-entropy estimator. Our proposed estimator instead uses $\lambda = 1$; it reduces variance only when the covariance term is sufficiently negative. Thus the experimental variance reduction for the proposed estimator is not a universal dominance result.

Finite-Sample Expectation and Consistency

Let J be uniform on $\{1, \ldots, m\}$ conditional on $\Omega = (\omega_1, \ldots, \omega_m)$, and draw $Y \sim p(\cdot \mid \omega_J)$. The expected proposed estimator has the information-theoretic representation

$$\mathbb{E}[\widehat{a}_m^{\mathrm{prop}}] = I(J; Y \mid \Omega). \tag{3.18}$$

Indeed,

$$\frac{p(J = j \mid Y, \Omega)}{p(J = j)} = \frac{p(Y \mid \omega_j)}{m^{-1} \sum_{i=1}^{m} p(Y \mid \omega_i)},$$

whose expected logarithm is Equation (3.6). Moreover,

$$I(J; Y \mid \Omega) = I(\omega_J; Y) - I(\Omega; Y) \leq I(\omega_J; Y) = a(x_{1:b}). \tag{3.19}$$

The proposed estimator is therefore a finite-sample lower-bound estimator in expectation. By Equation (3.16), the analytic-entropy estimator has the same finite-sample expectation.

For completeness, suppose the batch-label space is finite and there is an $\epsilon > 0$ such that $p(y \mid \omega) \geq \epsilon$ for every label configuration y and almost every ω. The strong law gives

$$\widehat{p}_m(y) = \frac{1}{m} \sum_{i=1}^{m} p(y \mid \omega_i) \longrightarrow p(y)$$

simultaneously for all y. The lower bound by ϵ makes the logarithm bounded and uniformly continuous. Define the oracle-predictive sample average

$$\widetilde{a}_m := \frac{1}{m} \sum_{t=1}^{m} \left[\log p(Y_{1:b}^t \mid \omega_t) - \log p(Y_{1:b}^t) \right].$$

The term omitted by replacing $\widehat{p}_m$ with p is exactly

$$\Delta_m := \widehat{a}_m^{\text{prop}} - \widetilde{a}_m = \frac{1}{m} \sum_{t=1}^{m} \log \frac{p(Y_{1:b}^t)}{\widehat{p}_m(Y_{1:b}^t)}.$$

Its magnitude satisfies

$$|\Delta_m| \leq \max_y |\log p(y) - \log \widehat{p}_m(y)| \longrightarrow 0 \qquad \text{almost surely.}$$

Moreover, $p(y), \widehat{p}_m(y) \in [\epsilon, 1]$, so the right-hand side is bounded by $|\log \epsilon|$. Dominated convergence therefore also gives $\mathbb{E}|\Delta_m| \to 0$; that is, the remainder vanishes in L^1 as well as almost surely.

The summands defining $\widetilde{a}_m$ are iid and bounded, with expectation $a(x_{1:b})$. The strong law and L^1 convergence of their sample average, together with the preceding bounds on Δ_m, therefore give

$$\widehat{a}_m^{\text{prop}}(x) \longrightarrow a(x) \tag{3.20}$$

almost surely and in L^1. In particular, $\mathbb{E}[\widehat{a}_m^{\text{prop}}(x)] \to a(x)$, so the finite-sample downward bias vanishes as $m \to \infty$. A realized estimate need not, however, be nonnegative. A summand of Equation (3.6) is negative whenever $p(Y_{1:b}^t \mid \omega_t) < \widehat{p}_m(Y_{1:b}^t)$, and sufficiently many negative summands can make their sample average negative. This does not contradict the mutual-information representation in Equation (3.18): conditional on Ω, averaging over the sampled labels gives the nonnegative quantity $I(J; Y \mid \Omega)$. Negative realized values are therefore Monte Carlo fluctuations around a nonnegative finite-sample target, distinct from the systematic downward plug-in bias of that target relative to exact BatchBALD.

3.6.2 Additional Experimental Details

Fixed-Batch MNIST Validation

The validated experiment used a fully connected ENN on MNIST. Its prior, learnable, and feature-extraction modules were fully connected, and its uncertainty index had size 1,000. At each of two classifier-training states, the analytic-entropy and proposed estimators were evaluated 1,000 times on the same fixed batch with $b = 4$ and $m = 10$. Within every proposed-estimator evaluation, each sampled label was evaluated under the same sampled parameter value ω_t that generated it, as specified in Equation (3.3). The archived results do not establish that common random numbers were shared between the analytic-entropy and proposed estimators.

Exact Two-State Calculation

Consider a random parameter with two possible values, $\omega \in \{1, 2\}$, and two labels, with

$$\Pr(\omega = i) = \pi_i, \qquad p(Y = j \mid \omega = i) = p_{ij}, \qquad i, j \in \{1, 2\}.$$

For $m = 2$, every combination of sampled model states and labels can be enumerated, allowing the expectation and variance of both estimators to be calculated without Monte Carlo error. The enumerated means agree throughout the grid, as required by Equation (3.16). Figure 3.3 shows that the variance ratio depends strongly on the model-state probability and conditional label distributions. Ratios above one favor the proposed estimator, but the cells below one confirm that variance reduction is not universal.

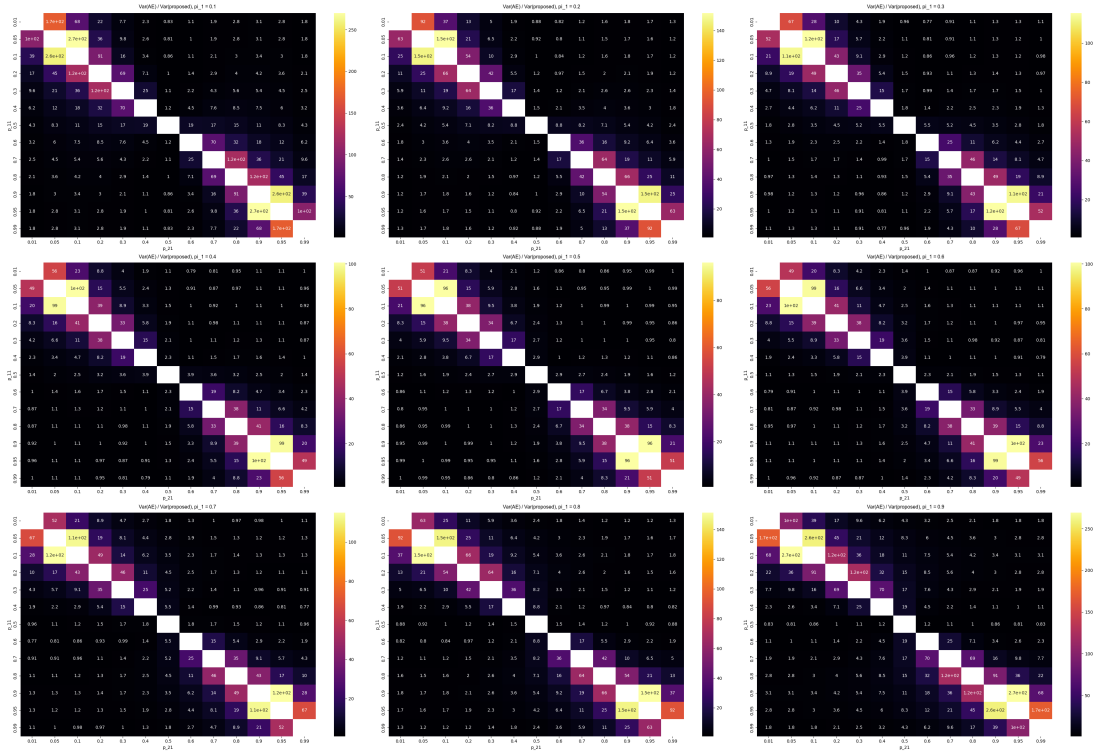


Figure 3.3: Exact ratios $\mathrm{Var}(\widehat{a}_m^{\mathrm{AE}})/\,\mathrm{Var}(\widehat{a}_m^{\mathrm{prop}})$ in the binary, two-state example with $m = 2$. Each panel fixes $\pi_1 = \Pr(\omega = 1)$; the vertical and horizontal axes give $p_{11} = \Pr(Y = 1 \mid \omega = 1)$ and $p_{21} = \Pr(Y = 1 \mid \omega = 2)$, respectively. Values above one indicate lower variance for the proposed estimator. White diagonal cells are degenerate cases in which the ratio is undefined. The calculation concerns acquisition-value estimation, not gradient validity.

CHAPTER 4

OPTIMIZATION ALGORITHMS FOR STUDENT-AWARE EXAM TIMETABLING

4.1 Introduction

This chapter studies optimization methods for student-aware exam timetabling. The problem is to assign each exam to a time slot while accounting for direct conflicts and other reported bad events, such as back-to-back exams and dense exam windows, together with operational restrictions such as front-loading large exams and limiting daily student participation. These costs are not simple linear functions of exam-slot assignment variables: they depend on the chronological order of each student's exams and the gaps between occupied slots. To represent this structure, we use precomputed personal timetable costs.

The chapter makes three contributions. First, it develops and evaluates a fast scheduling heuristic using public Cornell semester benchmark data (Ye et al., 2024; Jovine, 2025). The heuristic begins by scheduling exams that share many students and then makes local adjustments to reduce undesirable outcomes, such as closely spaced exams. For a fair comparison, the main benchmark focuses on the three semesters in which Cornell's published Hybrid schedule contains no direct conflicts. On these semesters, the heuristic finishes in about two minutes per semester and produces schedules whose independently evaluated scores are competitive with the corresponding Hybrid schedules under Cornell's published weighting of undesirable schedule features. Additional experiments examine the benefits of local improvement, the effect of scheduling large exams earlier, and violations of limits on students' daily exam participation. These results are based on one fixed configuration of a stochastic heuristic and therefore do not establish general

performance guarantees. They support the heuristic as an inexpensive method for generating and improving candidate schedules, rather than as a replacement for Cornell's integer-programming pipeline.

Second, the chapter gives an exact two-stage full-space formulation and a Benders decomposition that separates the global exam schedule from student-specific timetable evaluation. The decomposition supports warm starts and model reuse, gives immediate primal subproblem solutions at integer master schedules, and permits direct reuse of solver-returned linear-programming (LP) duals. The student cost depends largely on sorted occupied slots rather than on the labels of the exams occupying them. Sorted-combination and occupied-slot reformulations expose this label symmetry and reduce the student-side representation.

Third, a computational investigation identifies the master problem, rather than student-side evaluation, as the principal bottleneck in the implemented global Benders method. The reported runs optimize an exam-count-restricted objective containing the student groups whose members take at most five exams. Complete precomputation of the corresponding occupied-pattern costs and duals makes student-group lookup negligible, but every second-iteration master reaches its 300-second limit, master optimization accounts for 99.6–99.8% of measured loop time, and the lower bound remains zero. Even in a conservative comparison that uses the restricted Benders objective against the heuristic's full-population objective, the best Benders values are 82–115% worse and require 7.0–10.6 times as much computation. These results motivate stronger master formulations and cut strategies, as well as local use of Benders around heuristic schedules.

The chapter is organized as follows. Related work in Section 4.2 is followed by the empirical overview in Section 4.3. Section 4.4 presents the primal heuristic, followed by separate experiments on front-loading large exams and limiting daily

student participation in Sections 4.5 and 4.6. The difficulty of imposing the latter rule, together with the heuristic's lack of bounds or certificates, motivates exact optimization in Section 4.7. Section 4.8 presents the full-space mixed-integer formulation, Section 4.9 develops the Benders decomposition and its occupied-slot reformulation. Sections 4.10 and 4.11 report formulation scale and analyze Benders performance and its computational bottleneck. Section 4.12 discusses improvements to the master formulation and cut strategy, limited translation symmetry, and local use of Benders around heuristic starts.

4.2 Related Work

Realistic exam timetabling extends beyond direct-conflict feasibility: it must also address spacing, administrative restrictions, and schedule acceptability. Educational timetabling has a long history of constructive heuristics, integer programming, and metaheuristic improvement methods. Ceschia et al. (2023) survey standard educational timetabling formulations, benchmark instances, search methods, runtimes, and upper and lower bounds. Exam timetabling heuristics often build an initial assignment and then improve it with local moves, tabu search, simulated annealing, great deluge methods, variable-neighborhood search, or hyper-heuristics; Sin and Kham (2012) give one representative hyper-heuristic treatment based on great deluge move-acceptance rules. Decomposition and reformulation ideas have also been used in university timetabling: Burke et al. (2010) use restricted submodels and aggregated assignments to define neighborhoods for subsequent integer-programming improvement.

The survey by Qu et al. (2009) provides a useful taxonomy for positioning the primal heuristic in this chapter. Its constructive phase belongs to the graph-based sequential family: ordering exams by current conflict count is closely related to

Largest Degree and Color Degree, weighting edges by co-enrollment is related to Largest Weighted Degree, and giving large exams early priority is related to Largest Enrollment. These ideas are established components of exam-timetabling heuristics rather than claims of novelty here. The pairwise proximity score also has a direct precedent in the Toronto benchmark objective reviewed by Qu et al. (2009), which assigns penalties $16, 8, 4, 2, 1$ to co-enrolled exams separated by one through five slots. Our pairwise construction and proposal scores use the same proximity intuition, while move acceptance and final evaluation use Cornell-specific higher-order event metrics.

The improvement phase is a lightweight local search using slot swaps, single-exam moves, and one displacement, with strict acceptance under a weighted event score. Qu et al. (2009) document stronger mechanisms for escaping local optima, including tabu search, simulated annealing, Great Deluge, the Greedy Randomized Adaptive Search Procedure (GRASP), Kempe-chain exchanges, and variable-neighborhood search. The present method does not claim to dominate these approaches. Its possible use as a warm start or post-processing phase within Cornell's pipeline remains untested; multi-starts and larger conflict-preserving neighborhoods are natural extensions.

The Benders formulation differs more substantially from the decomposition methods in the survey. Classical timetabling decomposition generally partitions the exam set into difficult and easy subsets or conflict-free clusters and fixes decisions made in earlier stages. Here, the master retains the global exam-slot assignment and decomposition is by student or a group of students with identical exam sets and cost tables, represented by its multiplicity. This supports exact student value functions, dual-based lower bounds, and the occupied-slot symmetry analysis developed below. The computational outcome is mixed: the decomposition exposes useful structure

and valid bounds, but its labeled cuts did not provide an effective global algorithm on the full instances.

Two recent case studies are especially relevant: the Cornell integer-programming deployment of Ye et al. (2024) and the Bucknell open-source scheduling tool of Chaplin et al. (2026).

Ye et al. (2024) describe an integer-programming framework for Cornell's final exam schedule. They exclude undesirable slots and penalize direct conflicts, back-to-backs, two exams within three slots, triples, and three exams within four slots, with additional distinctions such as evening–morning adjacency. Their objective is therefore multi-term and window-based rather than a single pairwise conflict penalty.

Their Group-then-Sequence (GtS) method assigns exams to unlabeled blocks and sequences those blocks into slots. The tables below use Ye et al. (2024)'s method labels: GtSP is GtS followed by Post-Processing, and Zero-GtSP uses the Zero-Conflict block assignment before the same sequencing and post-processing stages. Layer-Cake instead introduces exams in enrollment-ordered layers, permits the newest layer to be rescheduled, and fixes older layers.

The Hybrid method uses Layer-Cake block assignments before Group-then-Sequence refinement. Layer-Cake is closest to the hard-exams-first motivation here, while Hybrid similarly separates initial combinatorial structure from subsequent placement improvement.

Chaplin et al. (2026) present a Bucknell case study and open-source tool for final-exam scheduling. Their work combines visualization, integer-programming models, and heuristics in a registrar-facing workflow. The Bucknell setting has hundreds of courses, thousands of students, and a fixed set of three-hour exam slots. Their evaluation criteria include overlapping exams, three exams within a

24-hour period, four exams within 48 hours, back-to-back exams, and night-to-morning patterns. A distinguishing feature of the Bucknell work is its emphasis on deployability: the tool is open source, uses an open-source solver, and supports schedule exploration in addition to optimization.

Like both studies, this chapter evaluates direct conflicts and spacing penalties and uses established hard-exams-first intuition. Its distinct focus is the representation of precomputed personal timetable costs, the resulting Benders decomposition, and the symmetry hidden by labeled exam-slot variables. Sorted-combination and occupied-slot views remove exam-label distinctions that do not affect a student's timetable cost.

4.3 Empirical Overview

The input records which exams each student takes. A schedule assigns every exam exactly once to one of 24 ordered slots, with three slots treated as one calendar day. The reported hard-conflict runs also require that no student have two exams in the same slot. Room assignment and room capacity are outside the scope of these experiments. Schedule quality is reported through events that are easy to interpret. A direct conflict occurs when a student has two exams in the same slot. A back-to-back event places two of a student's exams in consecutive slots, while the reported two-in-24-hour category places them exactly two slots apart. A triple places three exams in three consecutive slots, and a three-in-four event occupies three of four consecutive slots without forming a triple. For comparison with Cornell, events are counted in the order triple, three-in-four, back-to-back, and two-in-24-hour; slots used by an earlier event are not reported again as part of a later one. Section 4.3.4 gives the exact set-based definitions and tie-breaking rule.

Two operational rules are considered separately from these quality metrics. The

front-loading rule keeps exams with at least 300 students out of the final slot. The daily participation rule limits to 5,000 the number of distinct students with at least one exam on each three-slot calendar day. A student with several exams on one day is counted once. The first rule concerns the placement of individual large exams; the second couples all exams assigned to a day and is consequently more difficult for local repair.

4.3.1 Semester Instances

The computational difficulty depends not only on the number of exams and slots, but also on how many exams each student takes. A student taking k exams induces a personal timetable problem whose number of possible assignments grows quickly with k. It is also useful to count unique exam sets: students taking the same set of exams share the same subproblem structure and can be represented by one group with a student multiplicity. Semester labels use SP for spring and FA for fall, followed by the two-digit year.

Table 4.1 reports summary statistics computed from the five semester enrollment files in the public scheduling benchmark repository (Jovine, 2025). They contain approximately 16,000–18,000 students, 540–600 exams, and 10,000–12,000 unique exam sets. The number of slots is a modeling input; the experiments use 24 unless otherwise noted.

Table 4.1: Summary of exam enrollment instances across semesters.

Metric	SP24	FA23	SP23	FA22	SP22
Students	16,265	18,319	16,249	18,081	16,261
Unique exams	544	601	539	588	553
Unique exam sets	10,665	11,584	10,105	10,852	10,378
Time slots used in optimization	24	24	24	24	24

Table 4.2 shows that most students take between one and four exams. The

smaller group taking five or more exams is nevertheless important for the Benders computation because personal timetable enumeration grows rapidly with k.

Table 4.2: Distribution of students by number of exams taken. Percentages are within semester.

Semester	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
SP24	3,261 (20.05%)	4,011 (24.66%)	4,918 (30.24%)	3,214 (19.76%)	790 (4.86%)	67 (0.41%)	4 (0.02%)
FA23	3,368 (18.39%)	4,464 (24.37%)	5,396 (29.46%)	3,612 (19.72%)	1,353 (7.39%)	125 (0.68%)	1 (0.01%)
SP23	3,629 (22.33%)	4,091 (25.18%)	4,614 (28.40%)	2,946 (18.13%)	892 (5.49%)	76 (0.47%)	1 (0.01%)
FA22	3,376 (18.67%)	4,794 (26.51%)	5,476 (30.29%)	3,293 (18.21%)	1,060 (5.86%)	81 (0.45%)	1 (0.01%)
SP22	3,360 (20.66%)	4,063 (24.99%)	4,871 (29.96%)	3,035 (18.66%)	830 (5.10%)	97 (0.60%)	5 (0.03%)

Table 4.3 gives the corresponding counts after grouping students by unique exam set. This grouping reduces the number of student subproblems by roughly one third in the representative semesters. Shared, label-independent assignment and occupied-slot dictionaries can then be reused by all groups with the same k.

Table 4.3: Distribution of unique exam sets by number of exams. Percentages are within semester.

Semester	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
SP24	457 (4.29%)	2,598 (24.36%)	3,889 (36.47%)	2,883 (27.03%)	767 (7.19%)	67 (0.63%)	4 (0.04%)
FA23	481 (4.15%)	2,537 (21.90%)	3,982 (34.38%)	3,184 (27.49%)	1,277 (11.02%)	122 (1.05%)	1 (0.01%)
SP23	453 (4.48%)	2,629 (26.02%)	3,594 (35.57%)	2,572 (25.45%)	782 (7.74%)	74 (0.73%)	1 (0.01%)
FA22	475 (4.38%)	2,656 (24.47%)	3,943 (36.33%)	2,718 (25.05%)	979 (9.02%)	80 (0.74%)	1 (0.01%)
SP22	460 (4.43%)	2,582 (24.88%)	3,710 (35.75%)	2,705 (26.06%)	819 (7.89%)	97 (0.93%)	5 (0.05%)

These distributions explain two recurring algorithmic choices. The heuristic works directly with the sparse enrollment conflict matrix at the scale of roughly 600 exams. The reported Benders implementation groups repeated exam sets and uses the occupied-slot formulation to share precomputed pattern costs and duals across groups of the same size.

4.3.2 Headline Comparison

Table 4.4 gives the main empirical comparison before the algorithmic details. It averages the three semesters for which every listed Cornell method and the proposed

heuristic report zero direct conflicts. The score uses Cornell's default bad-event penalty ratios, scaled so that the back-to-back weight is one. Here D, R_3, B_1, B_2, and R_4 denote the direct-conflict, triple, back-to-back, two-in-three, and three-in-four counts, respectively; Section 4.3.4 gives their precise no-double-counting definitions. In table headings, B2B abbreviates back-to-back, 2-in-24 abbreviates two exams in 24 hours, and 3-in-4 abbreviates three exams in four consecutive slots. BS denotes Block Sequencing, PP denotes Post-Processing, BA denotes Block Assignment, and NR means not reported. For a schedule x, the weighted score is

$$F_{\text{Cornell}}(x) = 1000D(x) + 10R_3(x) + B_1(x) + 0.5B_2(x) + 5R_4(x).$$

The proposed heuristic uses the single parameter configuration reported in Table 4.5 for all three semesters and finishes in about two minutes per semester. The Cornell paper (Ye et al., 2024) reports only that each method finishes within several hours, so method-level observed runtimes are unavailable. It does, however, report limits of 25 minutes for Block Sequencing, 10 minutes for Post-Processing, and 25 minutes for each Schedule IP (integer-programming subproblem) in Layer-Cake (Ye et al., 2024). Precise event definitions and the full semester-by-semester comparison appear below.

The allowances are stage limits, not observed runtimes or end-to-end method bounds, so they do not support a precise runtime ratio.

Under Cornell's weighted aggregate objective, the independently evaluated heuristic schedules have the lowest three-semester average score among the listed methods, numerically 3.4% below the published Hybrid average. The unweighted component columns provide a finer-grained view and show that this aggregate advantage does not extend to every event type. In particular, the heuristic's average triple count (26.7) is higher than those of Layer-Cake (22.7) and Hybrid (21.3). This does not establish that the heuristic dominates the Cornell methods: the

Table 4.4: Headline three-semester averages on FA23, SP23, and SP22. Score is the weighted aggregate measure minimized under Cornell's published objective weights; lower is better, and the proposed heuristic has the lowest average score among the listed methods. Direct, Triple, B2B, 2-in-24, and 3-in-4 decompose that score into unweighted bad-event counts for a finer-grained comparison. Cornell stage allowances are solver limits, not observed runtimes.

Method	Runtime (s)	Reported stage allowance	Score	Direct	Triple	B2B	2-in-24	3-in-4
Proposed heuristic	118.6	Fixed iteration budget	4839.7	0	26.7	1776.3	2546.7	304.7
Cornell GtS	NR	Up to 25 min/BS; BA NR	6952.3	0	49.0	2588.0	2912.0	483.7
Cornell GtSP	NR	Up to 35 min for BS + PP; BA NR	5508.8	0	38.7	1997.3	2729.7	352.0
Cornell Zero-GtSP	NR	Up to 35 min for BS + PP; BA NR	5869.3	0	34.7	1800.3	2924.7	452.0
Cornell Layer-Cake	NR	Up to 25 min per Schedule IP	5713.2	0	22.7	1905.7	2805.0	435.7
Cornell Hybrid	NR	Layer-Cake + up to 35 min for BS + PP	5007.5	0	21.3	1642.3	2533.7	377.0

event evaluator was implemented independently, the methods trade off event types differently, and operational restrictions are examined separately in Sections 4.5 and 4.6. For each semester, the heuristic results come from a single replication of a stochastic method. They show what the reported runs produced, but do not guarantee completion, feasibility, or quality on another seed or instance.

4.3.3 Evaluation Notation

Let E be the set of exams, S the set of students, and $T = \{0, 1, \ldots, \tau\}$ the set of $h = |T| = \tau + 1$ discrete exam time slots. For each student $s \in S$, let $E_s \subseteq E$ denote the exams taken by student s, and let $k_s = |E_s|$. The primary scheduling variable is

$$x_{e,t} = \begin{cases} 1, & \text{if exam } e \text{ is assigned to slot } t, \\ 0, & \text{otherwise.} \end{cases}$$

4.3.4 Bad-Event Metrics and Operational Rules

The event categories are based on those reported by Ye et al. (2024); the explicit overlap and tie-breaking conventions below specify how they are evaluated in this chapter. All definitions are expressed in terms of the exam-slot variables x. Assume each exam is assigned exactly once, $\sum_{t\in T} x_{e,t} = 1$ for every $e \in E$. For student s, define the slot multiplicity and occupied-slot set

$$m_{s,t} = \sum_{e\in E_s} x_{e,t}, \qquad O_s(x) = \{t \in T : m_{s,t} \geq 1\}.$$

The direct-conflict count is

$$D(x) = \sum_{s\in S} \sum_{t\in T} \binom{m_{s,t}}{2}.$$

Thus, if a student has three exams in one slot, all three conflicting exam pairs are counted.

For each student, define four families of possible bad-event slot sets:

$$\mathcal{C}_s^{R_3}(x) = \big\{\{t, t+1, t+2\} : t, t+1, t+2 \in T,\ \{t, t+1, t+2\} \subseteq O_s(x)\big\},$$
$$\mathcal{C}_s^{R_4}(x) = \big\{A : A \subseteq O_s(x),\ |A| = 3,\ \max A - \min A = 3\big\},$$
$$\mathcal{C}_s^{B_1}(x) = \big\{\{t, t+1\} : t, t+1 \in T,\ \{t, t+1\} \subseteq O_s(x)\big\},$$
$$\mathcal{C}_s^{B_2}(x) = \big\{\{t, t+2\} : t, t+2 \in T,\ \{t, t+2\} \subseteq O_s(x)\big\}.$$

These are, respectively, three exams in three consecutive slots (a triple), three exams occupying three of four consecutive slots, a back-to-back pair, and a pair whose exams are exactly two slots apart. The last category is called two-in-24-hour or two-in-three in the tables; the subscript in B_d records the slot difference d. The aggregate back-to-back count includes every pair of adjacent slot indices, including an evening–morning boundary; the reported tables do not split that aggregate into within-day and overnight subtypes.

The search evaluator used in the reported heuristic runs permits the event sets to overlap. Its count of type $q \in \{R_3, R_4, B_1, B_2\}$ is $\sum_{s \in S} |\mathcal{C}_s^q(x)|$. Thus, a slot may contribute to several search events, but the definitions retain the exact-gap distinction above: an adjacent pair is in B_1, not also in B_2, and a width-two triple is in R_3, not also in R_4. For later use, denote these four raw overlapping counts by $\widetilde{R}_3(x)$, $\widetilde{R}_4(x)$, $\widetilde{B}_1(x)$, and $\widetilde{B}_2(x)$, respectively. The Cornell-style result tables instead use the following ordered no-reuse convention. Let

$$\mathcal{C}_s(x) = \mathcal{C}_s^{R_3}(x) \cup \mathcal{C}_s^{R_4}(x) \cup \mathcal{C}_s^{B_1}(x) \cup \mathcal{C}_s^{B_2}(x).$$

Order the event types by

$$R_3 \prec R_4 \prec B_1 \prec B_2,$$

and, within each type, order the increasing slot tuples lexicographically. These two rules define $\prec$ on $\mathcal{C}_s(x)$.

Define the counted events $\mathcal{P}_s(x)$ as follows. Set $\mathcal{R}_s^0 = O_s(x)$ and $\mathcal{P}_s^0 = \varnothing$. For $i \geq 1$, let

$$\mathcal{A}_s^i = \{A \in \mathcal{C}_s(x) : A \subseteq \mathcal{R}_s^{i-1}\}.$$

If $\mathcal{A}_s^i = \varnothing$, terminate and set $\mathcal{P}_s(x) = \mathcal{P}_s^{i-1}$. Otherwise, define

$$A_s^i = \min_{\prec} \mathcal{A}_s^i, \qquad \mathcal{P}_s^i = \mathcal{P}_s^{i-1} \cup \{A_s^i\}, \qquad \mathcal{R}_s^i = \mathcal{R}_s^{i-1} \setminus A_s^i.$$

Consequently, $A \cap A' = \varnothing$ for distinct $A, A' \in \mathcal{P}_s(x)$. The reported totals are

$$R_3(x) = \sum_{s \in S} |\mathcal{P}_s(x) \cap \mathcal{C}_s^{R_3}(x)|, \qquad R_4(x) = \sum_{s \in S} |\mathcal{P}_s(x) \cap \mathcal{C}_s^{R_4}(x)|,$$

$$B_1(x) = \sum_{s \in S} |\mathcal{P}_s(x) \cap \mathcal{C}_s^{B_1}(x)|, \qquad B_2(x) = \sum_{s \in S} |\mathcal{P}_s(x) \cap \mathcal{C}_s^{B_2}(x)|.$$

This rule uniquely determines the reported counts once x is fixed. The result depends on the stated priority and need not contain the largest possible number or total penalty of non-overlapping events. Ye et al. (2024) specify that triples

exclude their constituent pair events but do not fully specify tie-breaking for denser patterns; the order above is therefore our explicit reporting convention. Direct conflicts are counted separately and do not remove slots from the remaining set $\mathcal{R}_s^i$. In the reported experiments $D(x) = 0$, so slot multiplicities create no additional ambiguity.

We also distinguish two operational rules. Assume three exam slots per calendar day and define the day set and its slot partition by

$$\mathcal{D} = \{0, 1, \ldots, \lfloor \tau/3 \rfloor\}, \qquad T_d = \{3d, 3d+1, 3d+2\} \cap T \quad (d \in \mathcal{D}).$$

For each student and day, define the participation indicator and daily total ($\mathbf{1}\{\cdot\}$ equals one when its stated condition holds and zero otherwise):

$$u_{s,d}(x) = \mathbf{1}\left\{ \sum_{e \in E_s} \sum_{t \in T_d} x_{e,t} \geq 1 \right\}, \qquad N_d(x) = \sum_{s \in S} u_{s,d}(x).$$

A student with two or three exams on one day contributes one unit to N_d, and slots 2 and 3 belong to different days; the rule does not use arbitrary sliding three-slot windows. Our interpretation of the 5,000-student daily cap discussed by Ye et al. (2024) is

$$N_d(x) \leq 5000 \qquad \text{for every calendar day } d.$$

For diagnostic reporting, define

$$\begin{aligned} N_{\max}(x) &= \max_d N_d(x), \\ V_{5000}^{\max}(x) &= \max\{0, N_{\max}(x) - 5000\}, \\ V_{5000}^{\Sigma}(x) &= \sum_{d \in \mathcal{D}} \max\{0, N_d(x) - 5000\}. \end{aligned}$$

If the cap is imposed in a mixed-integer program (MIP), binary variables $u_{s,d}$ can be linked to x by

$$\begin{aligned} u_{s,d} &\geq x_{e,t} && \forall s \in S,\ e \in E_s,\ d \in \mathcal{D},\ t \in T_d, \\ u_{s,d} &\leq \sum_{e \in E_s} \sum_{t \in T_d} x_{e,t} && \forall s \in S,\ d \in \mathcal{D}, \end{aligned}$$

with $u_{s,d} \in \{0,1\}$ and $\sum_{s \in S} u_{s,d} \leq 5000$ for every $d \in \mathcal{D}$.

Finally, let $n_e = |\{s \in S : e \in E_s\}|$ be exam enrollment and define

$$E^{\text{large}} = \{e \in E : n_e \geq 300\}, \qquad T^{\text{early}} = \{t \in T : t \leq 22\}.$$

With 24 zero-indexed slots, our interpretation of Cornell's default size-300/front-slot-23 rule (Ye et al., 2024) is the linear restriction

$$\sum_{t \in T^{\text{early}}} x_{e,t} = 1 \quad \text{for every } e \in E^{\text{large}},$$

or equivalently $x_{e,23} = 0$ for every large exam. The number of front-loading violations is

$$V_{\text{late}}(x) = \sum_{e \in E^{\text{large}}} x_{e,23}.$$

The threshold and slot indexing above are the explicit conventions used throughout this chapter; they should be changed if the operational data specify a different enrollment or early-slot cutoff.

The full-space and Benders formulations below use the same $x_{e,t}$ variables. The family $\mathcal{P}_s(x)$ is used only to compute the reported metrics after a schedule is fixed. The formulations impose this counting rule only if they explicitly include its priority and set-exclusion conditions.

4.4 Primal Scheduling Heuristic

The symbol x denotes an exam-slot schedule throughout the chapter. In the heuristic, x is the current schedule, $\hat{x}$ is a proposal, and x^{best} is the best accepted schedule. In the MIP formulations, the same $x_{e,t}$ are decision variables.

The primal heuristic is the practical algorithmic contribution of the chapter. It constructs schedules using pairwise conflict scores and improves them under a weighted bad-event objective. It works with the same kinds of interpretable

bad-event metrics used in recent exam-scheduling deployments: direct conflicts, back-to-back exams, exact-gap-two pairs, and short runs of consecutive exams. Its pairwise scores can be evaluated through a sparse enrollment conflict matrix, without solving a MIP or enumerating every possible personal timetable.

The method uses the ordered structure of the calendar directly. It reasons about previous slots, next slots, adjacent-slot conflicts, and three-consecutive-slot patterns. The exact formulation developed later provides a common objective interpretation and a possible source of bounds, but the heuristic was not integrated with Benders in this work.

4.4.1 Representation and Search Objective

Let $W \in \mathbb{R}_{+}^{|E|\times|E|}$ be the weighted exam conflict matrix, where $W_{e,e'}$ is the number of students enrolled in both exams e and e', and $W_{e,e} = 0$. A schedule x assigns exams to slots using the binary variables $x_{e,t}$. The matrix W is computed once from the enrollment data and is typically sparse, because most pairs of exams have no students in common. The direct-conflict count is

$$D(x) = \frac{1}{2} \sum_{t\in T} \sum_{e\in E} \sum_{e'\in E} W_{e,e'} x_{e,t} x_{e',t},$$

and the back-to-back conflict count is

$$B_1^{\text{pair}}(x) = \sum_{t=0}^{\tau-1} \sum_{e\in E} \sum_{e'\in E} W_{e,e'} x_{e,t} x_{e',t+1}.$$

For a direct-conflict-free schedule, $B_1^{\text{pair}}(x) = \widetilde{B}_1(x)$; the separate notation emphasizes that the former is evaluated through the exam conflict matrix and is also used as an insertion score. The heuristic enforces direct-conflict feasibility and uses W to guide construction and move proposals toward student-weighted gap-one and gap-two improvements. Avoiding an adjacent pair of highly overlapping exams

benefits many students at once. The local-search acceptance decision, however, uses the overlapping-event score reported in Table 4.5, including separate penalties for back-to-backs, exact-gap-two pairs, triples, and width-three triple patterns. Thus the pairwise score guides proposals but is not substituted for the search objective. In the reported runs, strict move acceptance uses

$$F_{\text{search}}(x) = 10^6 D(x) + 2000\widetilde{R}_3(x) + 4\widetilde{R}_4(x) + 20\widetilde{B}_1(x) + 8\widetilde{B}_2(x). \tag{4.1}$$

The Cornell-style score in the result tables is evaluated separately using the non-overlapping convention in Section 4.3.4.

A student with exams in slots $t, t+1, t+2$ contributes two adjacent pairs, so the pairwise score already gives the search an incentive to break a triple, particularly by moving the middle exam. Pairwise counts cannot, however, distinguish one triple from two separate back-to-backs. The reported event score makes that distinction through its explicit triple penalty.

4.4.2 Configuration Used for the Reported Benchmark

Table 4.5 gives the configuration used for the headline unconstrained results. The same values were used for every semester. The run used a fixed number of proposals rather than a wall-clock stopping rule. The implementation processes one semester at a time in a single Python process and does not use a parallel worker pool or an optimization solver. Reported heuristic runtime is cumulative from data loading and conflict-matrix construction through schedule construction and local search. The previous-slot weight ρ controls how strongly the constructor avoids placing exams immediately after slots containing many students in common; its formal role in the priority score is defined in Section 4.4.3. In the table, "raw exact-gap overlap" refers to the overlapping event counts $\widetilde{R}_3, \widetilde{R}_4, \widetilde{B}_1, \widetilde{B}_2$ defined in Section

4.3.4; "weighted strict improvement" means that a proposal is accepted only when it strictly decreases F_{search}.

Table 4.5: Heuristic configuration used for the headline benchmark.

Component	Parameter	Value
Calendar	Slots; slots per day	24; 3
Construction	Previous-slot weight ρ	3
Construction	Forbid all previous-slot conflicts	No
Search	Metric mode; acceptance	Raw exact-gap overlap; weighted strict improvement
Search	Slot-swap proposals I_{swap}	2,000
Search	Exam-move proposals I_{move}	10,000
Search	Base seed; proposal temperature	0; 0.5
Search	One displacement allowed	Yes
Objective	Direct-conflict weight	1,000,000
Objective	B2B; two-in-three weights	20; 8
Objective	Triple; three-in-four weights	2,000; 4
Operations	Front-loading; daily cap	Off; monitored only after evaluation

This configuration was chosen to place substantial emphasis on eliminating triples and was not tuned separately by semester. It is only one point in a consequential parameter space: ρ changes the starting schedule; the event weights change which tradeoffs are accepted; the temperature changes which exams are proposed; the displacement rule changes the available moves; and the iteration counts determine search effort. Multi-starts, different weights, more proposals, or direct acceptance under the final Cornell evaluator could plausibly improve the reported Cornell score. No systematic tuning or sensitivity study has yet established the size or consistency of such an improvement. The reported results should therefore be interpreted as a benchmark for one fixed configuration, not as the best score attainable by this heuristic family.

4.4.3 Constructive Scheduling Phase

The constructive phase follows a hard-exams-first motivation: an exam that conflicts with many other exams may be more difficult to place after the available slots have been partly filled. It processes slots from first to last and dynamically orders the currently unscheduled exams. Exams that conflict with many remaining exams are prioritized, while exams with large student overlap with the preceding slot are deprioritized. These choices are heuristics rather than guarantees about the resulting schedule.

Let $\mathcal{U}_t$ be the set of unscheduled exams before processing slot t. For an exam $e \in \mathcal{U}_t$, let $r_{e,t}$ be the number of exams in $\mathcal{U}_t$ that conflict with e. For $t > 0$, let $\sigma_{e,t}$ be the total student-weighted conflict between e and the exams in slot $t-1$, divided by the largest such value among exams in $\mathcal{U}_t$; set $\sigma_{e,t} = 0$ when that largest value is zero. The priority score for slots after the first is

$$\kappa_{e,t} = -r_{e,t} + \rho\sigma_{e,t},$$

where $\rho \geq 0$ controls the previous-slot penalty. For the first slot, exams are instead ordered by decreasing weighted conflict degree $\sum_{e'} W_{e,e'}$. For later slots, they are considered in increasing order of $\kappa_{e,t}$. An exam is placed only if it creates no direct conflict with exams already in the current slot.

Lemma 4.1. *Algorithm 3 never creates a direct conflict within a slot.*

Proof. An exam is assigned to a slot only after checking that it has zero conflict weight with all exams already assigned to that slot. Therefore every accepted placement preserves direct feasibility within the current slot. The result follows by induction over the sequence of accepted placements. □

This lemma is a preservation statement, not a completeness guarantee. The constructive phase may leave exams unscheduled. This can occur either because

Algorithm 3 Constructive Scheduling Heuristic

Require: Conflict matrix W, slot set T, adjacency parameter ρ

Initialize $x_{e,t} \leftarrow 0$ for all e, t
Mark every exam as unscheduled
for each slot $t \in T$ **do**
    Let $\mathcal{U}_t$ be the set of unscheduled exams
    if $\mathcal{U}_t = \emptyset$ **then**
        break
    end if
    if $t = 0$ **then**
        Order $\mathcal{U}_t$ by decreasing weighted conflict degree
    else
        for each $e \in \mathcal{U}_t$ **do**
            Compute $r_{e,t}$, $\sigma_{e,t}$, and $\kappa_{e,t} \leftarrow -r_{e,t} + \rho\sigma_{e,t}$
        end for
        Order $\mathcal{U}_t$ by increasing $\kappa_{e,t}$
    end if
    for each exam e in this order **do**
        if e has no direct conflict with exams already assigned to slot t **then**
            Set $x_{e,t} \leftarrow 1$
            Mark e as scheduled
        end if
    end for
end for
return schedule x and any unscheduled exams

no assignment to the available slots avoids every direct conflict, or because early placement choices block a feasible completion that would have required a different earlier placement. In the hard-conflict version, unscheduled exams must be handled by restarts, randomized tie-breaking, additional repair moves, or a fallback optimization model. The reported benchmark instances were completed without introducing direct conflicts; this is an empirical observation, not a completeness guarantee for the constructor and insertion procedure.

4.4.4 Insertion of Unscheduled Exams

The constructive pass may leave exams unscheduled. The insertion phase evaluates every pair (e, t) of an unscheduled exam e and a slot t, discarding a placement if it creates a direct conflict. For each remaining pair, its insertion score is

$$\iota_{e,t} = \sum_{e' \in E} W_{e,e'} x_{e',t-1} + \sum_{e' \in E} W_{e,e'} x_{e',t+1},$$

where terms outside the slot range are omitted. The implementation computes these scores once for the partial schedule and sorts all feasible pairs by increasing $\iota_{e,t}$, breaking ties in favor of exams that conflict with more other exams. It then traverses this list. A pair is skipped if its exam has already been inserted or if earlier insertions have made its slot directly infeasible. The score is the marginal increase in $B_1^{\text{pair}}(x)$ when the list is formed; it is not recomputed after each insertion. Consequently, the procedure is an inexpensive completion rule, not a sequence of globally best current insertions. Algorithm 4 gives the complete procedure.

4.4.5 Local Search

After construction and insertion, the heuristic applies local search to reduce the current search objective (4.1). Its input is direct-conflict-free, and every accepted

Algorithm 4 Insertion of Unscheduled Exams

Require: Partial schedule x, unscheduled exam set $\mathcal{U}$, conflict matrix W

Initialize candidate list $\mathcal{Q}_{\text{ins}} \leftarrow \emptyset$
for each $e \in \mathcal{U}$ and $t \in T$ **do**
    if placing e in t creates no direct conflict **then**
        Compute $\iota_{e,t}$ and add $(e, t, \iota_{e,t})$ to $\mathcal{Q}_{\text{ins}}$
    end if
end for
Sort $\mathcal{Q}_{\text{ins}}$ by increasing $\iota_{e,t}$, then decreasing conflict count
for each $(e, t, \iota_{e,t}) \in \mathcal{Q}_{\text{ins}}$ in this order **do**
    if $e \in \mathcal{U}$ and placing e in t remains directly feasible **then**
        Insert e in t and remove e from $\mathcal{U}$
    end if
end for
return updated schedule x and any remaining unscheduled exams

move preserves direct feasibility. The search uses two neighborhoods.

The first is a time-slot swap neighborhood. Two distinct slots are sampled and their complete exam assignments are exchanged. This move preserves direct feasibility because all exams in each slot move together.

The second neighborhood selects one exam and proposes a new slot. If the target slot has no direct conflict with that exam, this is a single-exam move. The reported experiments also allow one displacement: if the moving exam conflicts with exactly one exam in the target slot, that exam may be moved into the vacated slot, provided it creates no conflict there. All other proposals are rejected. Longer displacement sequences are a possible extension but were not used in the reported results.

The reported local search, summarized in Algorithm 5, is stochastic because slot pairs, exams, and target slots are sampled. For $d \in \{1, 2\}$, define exam e's student-weighted slot-distance contribution by

$$b_{e,d}(x) = \sum_{e' \in E} W_{e,e'} \sum_{\substack{t,t' \in T \\ |t-t'|=d}} x_{e,t} x_{e',t'}, \qquad b_e(x) = b_{e,1}(x) + \frac{1}{4} b_{e,2}(x).$$

Here $d = 1$ is exactly a back-to-back pair and $d = 2$ is exactly a two-in-three pair. The score $b_e(x)$ is an exam-selection priority, not an estimate of the objective change from a particular proposed move. It favors exams currently involved in student-weighted back-to-back and exact-gap-two pairs. The factor $1/4$ gives exact-gap-two pairs less influence than adjacent pairs and was fixed for the reported runs rather than tuned empirically. The score therefore determines only which exam is considered; the complete candidate is evaluated separately. At temperature $\vartheta > 0$, exam e is selected with probability proportional to $(b_e(x) + 10^{-9})^{1/\vartheta}$; if all scores are zero, selection is uniform, and if $\vartheta = 0$, an exam of maximum score is selected. The target slot is uniform on T. The reported runs use $\vartheta = 0.5$ and accept only strict improvements in (4.1).

More precisely, the implementation compares the tuple consisting of the number of unscheduled exams, the direct-conflict count, and F_{search}, in that lexicographic order. Every candidate considered by the reported local search is complete and directly feasible, so the first two components remain zero and this rule reduces exactly to strict decrease in F_{search}.

The insertion score, proposal rule, and acceptance rule have different roles. The insertion score $\iota_{e,t}$ ranks feasible placements of exams left unscheduled by the constructor and measures only the marginal gap-one, or back-to-back, contribution. During local search, $b_e(x)$ changes only the probability that exam e is selected for a move; slot pairs and target slots are otherwise sampled uniformly. A proposal is accepted only if it remains directly feasible and strictly decreases F_{search}. Cornell's non-overlapping event score is used for final reporting, not for insertion, proposal generation, or acceptance.

Lemma 4.2. *Every accepted local-search move preserves completeness and direct-conflict feasibility.*

Algorithm 5 Structured Local Search for Conflict-Based Objectives

Require: Direct-conflict-free schedule x, objective F_{search}, iteration limits I_{swap} and I_{move}

$x^{\text{best}} \leftarrow x$
for $i = 1, \ldots, I_{\text{swap}}$ **do**
    Sample two distinct slots t and t'
    Let $\hat{x}$ be obtained by swapping slots t and t' in x^{best}
    if $F_{\text{search}}(\hat{x}) < F_{\text{search}}(x^{\text{best}})$ **then**
        $x^{\text{best}} \leftarrow \hat{x}$
    end if
end for
for $i = 1, \ldots, I_{\text{move}}$ **do**
    Sample an exam e, biased by $b_e(x^{\text{best}})$, and sample a target slot
    Attempt a single-exam move, followed by at most one displacement if needed
    if the proposal returns a direct-conflict-free candidate $\hat{x}$ **then**
        if $F_{\text{search}}(\hat{x}) < F_{\text{search}}(x^{\text{best}})$ **then**
            $x^{\text{best}} \leftarrow \hat{x}$
        end if
    end if
end for
return x^{best}

Proof. A slot swap exchanges two complete slot columns, so it preserves all same-slot exam pairs. A single-exam move is accepted only when the target slot contains no conflicting exam. In a one-displacement move, the moving exam has exactly one conflict in the target slot, and that conflicting exam is returned to the vacated slot only when it is feasible there. Each operation reassigns rather than removes or duplicates the affected exams. □

Remark 4.3. *If the local search accepts only strict improvements in $F_{\text{search}}(x)$, then the score decreases at every accepted move. This does not imply local optimality because the reported algorithm samples only finitely many proposals rather than exhaustively checking either neighborhood. It also does not imply monotonic improvement in a separately computed reporting metric unless that metric is exactly F_{search}.*

4.4.6 Construction and Local-Search Improvement

Table 4.6 separates the two empirical components of the proposed heuristic. The construction row is the complete schedule produced by the hard-exam-first insertion heuristic before any improvement moves. The construction is deterministic under fixed tie-breaking, while the local-search row applies 2,000 slot-swap attempts and 10,000 exam-move attempts, with one-exam displacement enabled. The same parameters and random seed are used for every semester. The complete configuration is given in Table 4.5. All reported schedules assign every exam and have zero direct conflicts. The preservation of these properties by accepted local-search moves follows from Lemma 4.2. Metrics are recomputed after each phase using the no-double-counting convention described above.

The constructive phase is therefore best viewed as a fast feasibility mechanism and a starting point, not as the final scheduling method. It produces a complete conflict-free schedule in roughly six to nine seconds, but most of the schedule-quality gain comes from local search. Across the five semesters, the average Cornell-weighted score falls from 16,750.5 after construction to 4,809.6 after local search. In particular, the triple count falls by more than an order of magnitude. The two-in-24-hour count does not decrease monotonically in every semester because the search optimizes a weighted scalarization and may trade one event type against another.

Pairwise conflict-matrix scores guide construction and the sampling of exam moves. Move acceptance in these runs uses (4.1), whose event accounting is not identical to Cornell's no-double-counting convention. The Cornell score in the table is computed separately. Its observed decrease is not a mathematical consequence of optimizing the search objective, and the sampled search provides no local- or global-optimality guarantee.

4.4.7 Cornell-Weighted Comparison

Table 4.7 gives the semester-level values behind the headline averages. It includes all five approaches reported by Ye et al. (2024) and restricts attention to semesters on which every listed schedule has zero direct conflicts. The proposed heuristic uses the single fixed configuration in Table 4.5; it was not tuned by semester.

The proposed heuristic has the lowest score on FA23 and SP23, while Hybrid has the lowest score on SP22. It does not dominate Hybrid componentwise: on average it has more triples and back-to-backs, nearly the same number of two-in-24-hour events, and fewer three-in-four events.

4.5 Heuristic with Front-Loading

Ye et al. (2024) also use a default front-loading rule that keeps exams with at least 300 students out of the final slot. Table 4.8 compares the original heuristic with a variant that enforces this rule before and during local search. Both variants use the same local-search iteration limits and objective weights. The original runs assign a weight of 3 to the constructor's penalty for student overlap with the preceding slot, whereas the separately implemented front-loading variant assigns a weight of 1. The comparison therefore measures two complete configurations and should not be interpreted as a one-factor causal estimate of front-loading alone. In the reported no-cap runs, the ordinary constructor is followed by a whole-slot repair: a slot containing a large exam after the cutoff is exchanged with an eligible earlier slot containing no large exam. Because complete slot columns are exchanged, this repair preserves completeness and the direct-conflict count. During local search, any move that would place a large exam after the cutoff is rejected. The table also reports $N_{\max}$ and $V_{5000}^{\max}$ from Section 4.3.4; thus days are fixed three-slot calendar

Table 4.6: Construction and local-search results for the original heuristic. Runtime is cumulative from data loading through the indicated phase.

Semester	Stage	Runtime (s)	Score	Direct	Triple	B2B	2-in-24	3-in-4
SP24	Construction	6.8	16210.5	0	677	3480	3131	879
SP24	Local search	108.7	4820.5	0	35	1668	2615	299
FA23	Construction	8.9	18173.5	0	734	3783	3751	1035
FA23	Local search	127.9	5431.0	0	29	2131	2430	359
SP23	Construction	5.9	14002.0	0	561	3245	2904	739
SP23	Local search	113.8	4348.0	0	22	1706	2324	252
FA22	Construction	7.2	18427.0	0	734	3843	3538	1095
FA22	Local search	119.9	4708.5	0	26	1765	2337	303
SP22	Construction	6.3	16939.5	0	751	3421	3087	893
SP22	Local search	114.1	4740.0	0	29	1492	2886	303
Average	Construction	7.0	16750.5	0	691.4	3554.4	3282.2	928.2
Average	Local search	116.9	4809.6	0	28.2	1752.4	2518.4	303.2

Table 4.7: Comparison under Cornell's default reported-bad-event penalty ratios on semesters where all listed methods report zero direct conflicts. Runtime is not reported (NR) by method for the Cornell experiments.

Semester	Method	Runtime (s)	Score	Direct	Triple	B2B	2-in-24	3-in-4
FA23	Proposed heuristic	127.9	5431.0	0	29	2131	2430	359
FA23	Cornell GtS	NR	8420.0	0	51	2938	3534	641
FA23	Cornell GtSP	NR	6979.5	0	48	2215	3669	490
FA23	Cornell Zero-GtSP	NR	6715.5	0	33	2038	3185	551
FA23	Cornell Layer-Cake	NR	6857.0	0	26	2323	3088	546
FA23	Cornell Hybrid	NR	5887.5	0	26	1950	2775	458
SP23	Proposed heuristic	113.8	4348.0	0	22	1706	2324	252
SP23	Cornell GtS	NR	6266.5	0	43	2332	2709	430
SP23	Cornell GtSP	NR	4934.5	0	40	2024	2211	281
SP23	Cornell Zero-GtSP	NR	4940.0	0	27	1651	2648	339
SP23	Cornell Layer-Cake	NR	5376.5	0	22	1784	2545	420
SP23	Cornell Hybrid	NR	4985.5	0	22	1566	2309	409
SP22	Proposed heuristic	114.1	4740.0	0	29	1492	2886	303
SP22	Cornell GtS	NR	6170.5	0	53	2494	2493	380
SP22	Cornell GtSP	NR	4612.5	0	28	1753	2309	285
SP22	Cornell Zero-GtSP	NR	5952.5	0	44	1712	2941	466
SP22	Cornell Layer-Cake	NR	4906.0	0	20	1610	2782	341
SP22	Cornell Hybrid	NR	4149.5	0	16	1411	2517	264

groups and a student with multiple exams on one day is counted once.

Because the search rejects every proposal that would place a large exam in the final slot, it preserves front-loading whenever the initial schedule satisfies the rule. This does not guarantee discovery of a complete front-loaded schedule. Complete, conflict-free, front-loaded starts were observed on all five semester instances, but neither this observation nor Lemma 4.1 guarantees feasibility or a specified objective value on another run.

Table 4.8: Comparison with a variant enforcing the size-300/front-slot-23 rule. "Late" is the number of size-300 exams in the final slot. "Max/day" is $N_{\max}$, and "peak excess" is $V_{5000}^{\max} = \max\{0, N_{\max} - 5000\}$. The 5,000-student cap is reported only as a diagnostic and is not imposed in these runs.

Semester	Variant	Score	Triple	B2B	2-in-24	3-in-4	Late	Max/day	Peak excess
SP24	Original	4820.5	35	1668	2615	299	3	6449	1449
SP24	Front-loaded	5177.5	41	2143	2329	292	0	6347	1347
FA23	Original	5431.0	29	2131	2430	359	3	8005	3005
FA23	Front-loaded	5481.0	40	2128	2686	322	0	7716	2716
SP23	Original	4348.0	22	1706	2324	252	3	6481	1481
SP23	Front-loaded	4209.5	15	1657	2385	242	0	5491	491
FA22	Original	4708.5	26	1765	2337	303	3	7346	2346
FA22	Front-loaded	5437.5	23	1857	3101	360	0	6771	1771
SP22	Original	4740.0	29	1492	2886	303	4	6815	1815
SP22	Front-loaded	4572.5	21	1718	2689	260	0	7048	2048
Average	Original	4809.6	28.2	1752.4	2518.4	303.2	3.2	7019.2	2019.2
Average	Front-loaded	4975.6	28.0	1900.6	2638.0	295.2	0.0	6674.6	1674.6

Front-loading removes every late large exam and reduces average peak daily participation, but it does not improve the bad-event score on average. It does, however, produce lower scores on SP23 and SP22. This is evidence of search-path sensitivity rather than a benefit of the constraint itself. Let $\mathcal{F}'$ be the schedules satisfying front-loading and $\mathcal{F}$ the schedules without that restriction. Since $\mathcal{F}' \subseteq \mathcal{F}$, for this minimization objective

$$\min_{x \in \mathcal{F}} F_{\text{Cornell}}(x) \leq \min_{x \in \mathcal{F}'} F_{\text{Cornell}}(x).$$

Thus, a lower heuristic incumbent—the best schedule found by that run—from a

front-loaded run cannot mean that adding the constraint improved the attainable optimum. Instead, the different construction coefficient and random stream, the initialization repair where it was needed, and the restricted move sequence led the finite local search to a different incumbent. This behavior supports running multiple randomized and structurally distinct starts in parallel, including front-loaded and unconstrained starts, and retaining the best schedule that satisfies the desired operational rules.

4.6 Attempts to Limit Daily Student Participation

The Cornell study describes 5,000 students per day as a potential operational cap and reports that its generated schedules remain below it, but the methods reported in Table 2 of Ye et al. (2024) do not explicitly include this cap in their stated formulations. Under the definition in Section 4.3.4, none of the original or front-loaded runs in Table 4.8 satisfies the cap.

We therefore tested a separate repair procedure. It starts from a complete, direct-conflict-free, front-loaded schedule and considers single-exam moves and cross-day exam swaps. Let

$$K(x) = \left(V_{5000}^{\Sigma}(x), V_{5000}^{\max}(x), \sum_{d \in \mathcal{D}} N_d(x) \right).$$

The procedure retains the lexicographically smallest $K(x)$ encountered. Its trajectory uses a simulated-annealing acceptance rule (Kirkpatrick et al., 1983): for a proposal $\hat{x}$, it forms

$$\Delta = \left[V_{5000}^{\Sigma}(\hat{x}) - V_{5000}^{\Sigma}(x)\right] + 0.05\left[V_{5000}^{\max}(\hat{x}) - V_{5000}^{\max}(x)\right] + 0.001 \sum_{d \in \mathcal{D}} \left[N_d(\hat{x}) - N_d(x)\right]$$

and accepts when $\Delta \leq 0$ or with probability $\exp(-\Delta/T_i)$ otherwise, where T_i is the temperature at attempted move i. The temperature decreases linearly from

3, subject to a floor of 0.03. Consequently, the search may temporarily increase cap violation, but the returned incumbent cannot be worse in K than the starting schedule. Every proposed move is screened to preserve completeness, direct-conflict feasibility, and front-loading. Table 4.9 reports a single replication of 300,000 attempted moves per semester.

Table 4.9: Daily-cap repair from complete, conflict-free, front-loaded schedules. "Cap met" indicates whether every day has at most 5,000 participating students; "Max/day" is the largest daily participation count; "total excess" sums the amount above 5,000 over all days; and "Accepted" is the number of accepted proposals among 300,000 attempted moves. Runtime covers the repair procedure.

Semester	Cap met	Max/day	Total excess	Runtime (s)	Accepted	Triple	B2B	2-in-24	3-in-4
SP24	Yes	5000	0	7.4	193	110	2485	2552	364
FA23	No	6430	2408	26.6	27651	794	4197	3772	606
SP23	Yes	5000	0	5.9	63	38	2012	2486	382
FA22	No	5153	390	24.8	27131	808	4246	3357	772
SP22	Yes	5000	0	6.2	162	128	2332	2634	317

The procedure reaches the cap for the three spring semesters but not for FA23 or FA22. Failure within the move limit does not establish that a cap-feasible schedule is impossible; it shows only that these moves and this starting schedule did not find one. The distinction matters because daily participation counts the union of students with an exam during a day. Moving one exam can change the loads of two days, and whether a student is added to or removed from a day depends on that student's other exams. This is harder to control than an exam-level capacity sum.

For the three cap-feasible spring schedules, a subsequent local search was restricted to moves preserving the cap, direct-conflict feasibility, completeness, and front-loading. Runs of about 51–54 seconds retained all four conditions and reduced the Cornell-weighted scores to 6450.0 on SP24, 4652.5 on SP23, and 5888.0 on SP22. Longer three-minute runs further reduced the first two scores to 6358.0 and 4321.0, respectively. These schedules remain weaker on some bad-event metrics than the unconstrained heuristic, illustrating the cost of maintaining several operational

restrictions with a small local-search move set. The approach would benefit from moves that rearrange several exams jointly or from an optimization model that imposes the daily limits directly.

The published Cornell-method counts are taken from Ye et al. (2024); the proposed-heuristic counts are computed from the semester benchmark enrollment files using the evaluator described in this chapter. The evaluator was implemented independently and was not cross-validated against Cornell's implementation. The heuristic values are from a single replication rather than a distribution over repeated runs.

4.7 Motivation for Exact Optimization

The experiments show that the heuristic can produce strong schedules quickly, but adding interacting operational requirements may require substantial constraint-specific engineering in its construction, repair, and move filters, and the tested daily-cap repair did not succeed on every semester. The heuristic also provides neither a lower bound nor a certificate that no better feasible schedule exists. Integer programming addresses these limitations by imposing operational rules explicitly and producing valid bounds. Because a direct model contains an enormous family of personal timetables, we first state it as a reference formulation and then decompose it into a global exam-assignment master and student-level evaluations.

4.8 Full-Space Mixed-Integer Formulation

The full-space model provides a common mathematical reference for the heuristic objective, the Benders investigation, and future local certification around heuristic solutions. It has a deterministic two-stage block structure. The first-stage binary

variables x choose the common exam schedule. Conditional on x, the second-stage variables $y_s = (y_{s,\pi})_{\pi \in \Pi_s}$ evaluate the personal timetable induced for each student, and these student blocks are mutually separable. Here "stage" describes the algebraic decomposition, not uncertainty or the chronological order in which scheduling decisions are made; all input data are known when the model is solved. We use *MIP* for this formulation and the Benders master because they combine binary schedule variables with continuous auxiliary variables. The student subproblems are linear programs (LPs). The broader abbreviation *IP* is retained only in generic discussion and in the names of cited methods.

For each student s, let Π_s denote the set of possible labeled personal timetables. A timetable $\pi \in \Pi_s$ maps every exam $e \in E_s$ to a slot $t_{\pi,e} \in T$. We also identify π with its graph $\{(e, t_{\pi,e}) : e \in E_s\}$ when writing expressions such as $(e, t) \in \pi$. Depending on the modeling choice, Π_s may contain only conflict-free personal timetables, or it may allow multiple exams for the same student in the same slot and penalize such conflicts through the disutility function.

For each $\pi \in \Pi_s$, define $U_{s,\pi}$ as the disutility experienced by student s under timetable π. The value $U_{s,\pi}$ is precomputed by evaluating the complete personal timetable; the exact formulation does not require this cost to be additive or separable across gaps. We assume $U_{s,\pi} \geq 0$; if negative column costs are allowed, the lower bound $\theta_s \geq 0$ in the Benders master must be removed. Let $D_s(\pi)$, $R_{3,s}(\pi)$, $R_{4,s}(\pi)$, $B_{1,s}(\pi)$, and $B_{2,s}(\pi)$ denote student s's contributions to the event counts in Section 4.3.4 when their exams follow π. In particular, the higher-order counts and exclusions are evaluated before the cost is stored. One relevant specification is

$$U_{s,\pi} = \omega_D D_s(\pi) + \omega_{R_3} R_{3,s}(\pi) + \omega_{R_4} R_{4,s}(\pi) + \omega_{B_1} B_{1,s}(\pi) + \omega_{B_2} B_{2,s}(\pi) + \omega_L L_s(\pi), \tag{4.2}$$

where $L_s(\pi)$ is an optional last-exam-time or other personal-timetable term. The

coefficients $\omega_D, \omega_{R_3}, \omega_{R_4}, \omega_{B_1}, \omega_{B_2}, \omega_L$ are chosen nonnegative weights and may reproduce the reported Cornell scalarization or another institutional objective. More generally, the formulation treats $U_{s,\pi}$ as a table of column costs and therefore also applies to nonseparable alternatives.

The full formulation contains the global exam assignment variables x and student-specific variables $y_{s,\pi}$, where $y_{s,\pi}$ is the convex coefficient assigned to personal timetable π for student s. At an integral schedule, one coefficient equals one and the others equal zero; at a fractional schedule, the coefficients describe a distribution over personal timetables. For notational convenience, write

$$\Pi_{s,e,t} = \{\pi \in \Pi_s : t_{\pi,e} = t\},$$

so that sums over $\Pi_{s,e,t}$ collect exactly the personal timetables in which exam e is placed in slot t. The formulation is

$$\min_{x,y} \quad \sum_{s\in S} \sum_{\pi\in\Pi_s} U_{s,\pi} y_{s,\pi} \tag{4.3}$$

$$\text{s.t.} \quad \sum_{t\in T} x_{e,t} = 1 \qquad \forall e \in E, \tag{4.4}$$

$$\sum_{\pi\in\Pi_s} y_{s,\pi} = 1 \qquad \forall s \in S, \tag{4.5}$$

$$\sum_{\pi\in\Pi_{s,e,t}} y_{s,\pi} = x_{e,t} \qquad \forall s \in S,\ e \in E_s,\ t \in T, \tag{4.6}$$

$$x_{e,t} \in \{0,1\} \qquad \forall e \in E,\ t \in T, \tag{4.7}$$

$$y_{s,\pi} \geq 0 \qquad \forall s \in S,\ \pi \in \Pi_s. \tag{4.8}$$

If student conflicts are forbidden as hard constraints, then the model also includes

$$\sum_{e\in E_s} x_{e,t} \leq 1 \qquad \forall s \in S,\ t \in T. \tag{4.9}$$

If student conflicts are allowed but penalized, then (4.9) is omitted and same-slot personal assignments are included in Π_s with appropriate costs $U_{s,\pi}$.

Constraints (4.5) and (4.6) force the student-level assignment distribution to be consistent with the global timetable. When x is integral, these constraints set the coefficient of the unique personal timetable induced by x to one and all other coefficients to zero. When x is fractional, the y-variables can be interpreted as a distribution over possible personal timetables whose marginals agree with the fractional master solution.

The difficulty of (4.3)–(4.8) is the number of y-variables. Even when each student takes only a small number of exams, the sets Π_s can be large across many students and many slots. This motivates a decomposition approach.

4.9 Benders Decomposition

Benders (1962) introduced a decomposition that separates the global scheduling decisions from the student-level cost calculations. The master problem chooses an exam timetable x and lower bounding variables θ_s for the contribution of each student. The student subproblem evaluates the cost of a proposed timetable and returns a dual solution used to generate an optimality cut. At a high level, Benders iterates between these two models: the master proposes a schedule, the subproblems evaluate that schedule from the students' perspectives, and the resulting cuts teach the master lower bounds on the cost of nearby or related schedules.

This decomposition is not the only possible way to attack the large y-variable formulation. A Dantzig-Wolfe or column-generation approach (Dantzig and Wolfe, 1960) would instead keep a restricted set of personal timetable columns and price new columns as needed. The Benders approach used here eliminates the student timetable variables from the master and treats the student problems as value-function evaluations. This is attractive when the master schedule is the main object of interest, but it also means that the strength and number of cuts determine the

size and quality of the master approximation.

The decomposition supports warm starts and grouped student evaluations. At an integer master schedule, each student is forced into the unique induced personal timetable. The occupied-slot reformulation below goes further: when personal cost depends only on occupied slots, an optimal LP dual can be precomputed once for each occupied-slot pattern and reused whenever that pattern occurs. The formulation also exposes that many exam-label assignments share the same sorted occupied slots and gaps.

The student problems have complete recourse on the stated master domain, meaning that each student subproblem is feasible for every feasible master point. If conflicts are allowed, Π_s contains every labeled personal assignment. If conflicts are forbidden, the master enforces the hard-conflict inequalities (4.9) and Π_s contains every conflict-free personal assignment. In the first case, the product of the exam-slot marginals gives a feasible distribution over labeled assignments. In the second, the exam assignment equalities and slot-capacity inequalities describe the bipartite matching polytope and hence the convex hull of conflict-free personal assignments (Nemhauser and Wolsey, 1988). Hence every feasible master point admits a student timetable distribution satisfying the linking constraints. The subproblems are also bounded because their column costs are finite. Consequently, this decomposition requires optimality cuts but no Benders feasibility cuts.

To keep the two recourse formulations distinct, $(\nu_s, \lambda_{s,e,t})$ denotes an assignment-level dual throughout the chapter. The occupied-slot reformulation later uses $(\mu_s, \zeta_{s,t})$. The different symbols reflect different linking constraints and different coefficient spaces, even though the two pairs play analogous dual roles. The assignment-level value function is $f_s(x)$, and the occupied-slot value function is $\phi_s(o_s)$. The master variable θ_s retains the same symbol because it is the lower

bound on whichever one of these alternative recourse functions the master uses; the two recourse formulations are not imposed simultaneously in the reported model.

The reported Benders implementation uses the Cornell no-double-counting objective defined in Section 4.3.4 as the stored personal-timetable cost $U_{s,\pi}$. Because these costs enter the recourse formulation as data, the mathematical decomposition can accommodate other personal-timetable objectives without changing its dimensions, but all reported Benders experiments use the Cornell objective. The occupied-slot implementation is conflict-free, as required by Section 4.9.6.

4.9.1 Master Problem

For any fixed exam assignment x, let $f_s(x)$ denote the optimal value of student s's assignment-level evaluation problem (4.16)–(4.19), defined explicitly in the next subsection. Let $\mathcal{K}_s^{(r)} \subseteq \{1, \ldots, r-1\}$ be the indices of the cuts available for student s before the iteration-r master is solved; in particular, $\mathcal{K}_s^{(1)} = \varnothing$. At iteration r, the master problem has the form

$$\min_{x,\theta} \quad \sum_{s \in S} \theta_s \tag{4.10}$$

$$\text{s.t.} \quad \sum_{t \in T} x_{e,t} = 1 \qquad \forall e \in E, \tag{4.11}$$

$$\sum_{e \in E_s} x_{e,t} \leq 1 \qquad \forall s \in S,\ t \in T \quad \text{if conflicts are forbidden,} \tag{4.12}$$

$$\theta_s \geq \alpha_s^{\ell} + \sum_{e \in E_s} \sum_{t \in T} \lambda_{s,e,t}^{\ell} x_{e,t} \qquad \forall s \in S,\ \ell \in \mathcal{K}_s^{(r)}, \tag{4.13}$$

$$x_{e,t} \in \{0,1\} \qquad \forall e \in E,\ t \in T, \tag{4.14}$$

$$\theta_s \geq 0 \qquad \forall s \in S. \tag{4.15}$$

The master can also be solved with $x_{e,t} \in [0,1]$ to obtain relaxed lower bounds. For a cut generated at iteration ℓ,

$$\alpha_s^\ell = f_s(x^{(\ell)}) - \sum_{e \in E_s} \sum_{t \in T} \lambda_{s,e,t}^\ell x_{e,t}^{(\ell)},$$

so (4.13) is the affine Benders cut $\theta_s \geq f_s(x^{(\ell)}) + \lambda_s^\ell \cdot (x - x^{(\ell)})$, where the dot denotes the indexed sum over $E_s \times T$ displayed above.

4.9.2 Student Subproblem

Given a master solution $x^{(r)}$, the subproblem for student s is

$$f_s(x^{(r)}) = \min_y \quad \sum_{\pi \in \Pi_s} U_{s,\pi} y_{s,\pi} \tag{4.16}$$

$$\text{s.t.} \quad \sum_{\pi \in \Pi_{s,e,t}} y_{s,\pi} \leq x_{e,t}^{(r)} \qquad \forall e \in E_s,\ t \in T, \tag{4.17}$$

$$\sum_{\pi \in \Pi_s} y_{s,\pi} = 1, \tag{4.18}$$

$$y_{s,\pi} \geq 0 \qquad \forall \pi \in \Pi_s. \tag{4.19}$$

The inequality form of (4.17) is convenient for fractional master solutions and gives nonpositive linking duals. When $x^{(r)}$ is a feasible master solution and $\sum_t x_{e,t}^{(r)} = 1$ for each exam, these inequalities are tight in aggregate for each exam because $\sum_{\pi \in \Pi_s} y_{s,\pi} = 1$ and every personal timetable assigns each exam to exactly one slot. The dual variable for (4.18) is free, while the dual variables for (4.17) are nonpositive. The dual is

$$\max_{\nu_s, \lambda_s} \quad \nu_s + \sum_{e \in E_s} \sum_{t \in T} \lambda_{s,e,t} x_{e,t}^{(r)} \tag{4.20}$$

$$\text{s.t.} \quad \nu_s + \sum_{(e,t) \in \pi} \lambda_{s,e,t} \leq U_{s,\pi} \qquad \forall \pi \in \Pi_s, \tag{4.21}$$

$$\lambda_{s,e,t} \leq 0 \qquad \forall e \in E_s,\ t \in T. \tag{4.22}$$

The origin and validity of the cut follow directly from LP duality.

Proposition 4.4. *For any dual-feasible pair* (ν_s, λ_s) *satisfying* (4.21)–(4.22)*, the inequality*

$$\theta_s \geq \nu_s + \sum_{e \in E_s} \sum_{t \in T} \lambda_{s,e,t} x_{e,t}$$

is valid for the assignment-level Benders master. If the pair is dual optimal at $x^{(r)}$*, the inequality is tight at* $x^{(r)}$*.*

Proof. The dual feasible region does not depend on x. For any master point at which the student subproblem is feasible, weak duality gives

$$f_s(x) \geq \nu_s + \sum_{e \in E_s} \sum_{t \in T} \lambda_{s,e,t} x_{e,t}.$$

Thus every point satisfying $\theta_s \geq f_s(x)$ also satisfies the displayed inequality. If the dual is optimal at $x^{(r)}$, strong duality makes its right-hand side equal to $f_s(x^{(r)})$, proving tightness. □

Given an optimal dual solution $(\nu_s^{(r)}, \lambda_s^{(r)})$, strong duality gives

$$f_s(x^{(r)}) = \nu_s^{(r)} + \sum_{e \in E_s} \sum_{t \in T} \lambda_{s,e,t}^{(r)} x_{e,t}^{(r)}.$$

Substituting this identity into the cut in Proposition 4.4 gives the equivalent anchored form

$$\theta_s \geq f_s(x^{(r)}) + \sum_{e \in E_s} \sum_{t \in T} \lambda_{s,e,t}^{(r)} \left(x_{e,t} - x_{e,t}^{(r)} \right). \tag{4.23}$$

This assignment-level student problem comes directly from the full-space model in Section 4.8: after the master fixes x, the linking equalities (4.6) separate by student and are written in the equivalent inequality form (4.17). It has one linking row and one dual coefficient for each $(e, t) \in E_s \times T$. Section 4.9.6 later defines a different, aggregated student recourse model for conflict-free schedules. That model links personal timetable patterns to the slot-occupancy marginals $o_{s,t} = \sum_{e \in E_s} x_{e,t}$

and has one linking row and dual coefficient per slot. The later section states its assumptions and relation to the assignment-level relaxation; simply summing the assignment-level linking rows above does not replace them with that aggregated recourse model.

4.9.3 Benders Algorithm

Algorithm 6 writes $\mathcal{K}_s$ for the mutable cut set; at the start of iteration r, this set is $\mathcal{K}_s^{(r)}$ in the master notation above. The algorithm uses LB and UB for the best valid lower and feasible upper bounds, respectively.

Algorithm 6 Benders Decomposition for Student-Aware Exam Timetabling

Require: Exam set E, student sets E_s, slot set T, assignment costs $U_{s,\pi}$
Initialize cut sets $\mathcal{K}_s \leftarrow \emptyset$ for all $s \in S$
Initialize $LB \leftarrow -\infty$, $UB \leftarrow +\infty$
for $r = 1, 2, \ldots$ **do**
    Solve the integer master problem (4.10)–(4.15)
    Let $x^{(r)}$ be its feasible schedule and update LB with the solver's valid master bound
    Set $UB^{(r)} \leftarrow 0$
    for each student s **do**
        Solve (4.16)–(4.19) and obtain an optimal LP dual
        Add $f_s(x^{(r)})$ to $UB^{(r)}$
        Generate the Benders cut (4.23)
        Add the cut to $\mathcal{K}_s$
    end for
    Update $UB \leftarrow \min\{UB, UB^{(r)}\}$
    if the optimality gap is below tolerance **then**
        break
    end if
end for
return best incumbent schedule, lower bound, and upper bound

4.9.4 Why the Cuts Can Be Weak for the Binary Master

Although the Benders cuts are valid, they can provide weak global information for the binary master problem. The issue is not dual feasibility; it is representation. The cuts are written in the space of absolute exam-slot assignment variables $x_{e,t}$. In this space, time slots are treated as labels. However, a large part of the student disutility depends on ordinal and metric relationships among slots, such as relative order and gaps.

For example, suppose a student's spacing cost depends only on the gap between two exams. Then assigning those exams to slots $(3, 7)$ and assigning them to slots $(8, 12)$ are equivalent with respect to the gap-based part of the disutility. Both have gap 4. A Benders cut generated at the first timetable, however, is expressed using coefficients on variables such as $x_{e,3}$ and $x_{e',7}$. It does not automatically generate the translated statement involving $x_{e,8}$ and $x_{e',12}$. Thus the cut learns a fact about a particular labeled timetable rather than a fact about the relative pattern of exam times.

This is a mismatch between the geometry of the master problem and the structure of the student disutility. The binary master moves among vertices of the assignment polytope, and two vertices that are structurally similar from the student's perspective may be far apart in the x-space representation. As a result, the master can jump from one integer timetable to another while previously generated cuts remain nearly flat or only weakly informative at the new point. The lower bound may then improve slowly even though the algorithm is adding valid cuts.

The same phenomenon appears as a lack of symmetry exploitation. Many timetables are equivalent or nearly equivalent under translations of a student's exam pattern, swaps of similar slots, or replacements of an exam by another exam with a similar conflict neighborhood. The cuts in (4.23) are not invariant to these

transformations. They do not directly encode statements such as "a gap of one is bad" or "three consecutive exams are bad" independently of where those patterns occur in the calendar. Instead, those statements must be relearned through many labeled variables and many integer master solutions.

The cached semester-scale runs show that, after occupied-pattern precomputation makes student-side evaluation inexpensive, the master becomes the computational bottleneck once many cuts are added, while the lower bound progresses slowly. This behavior is consistent with the representation argument above: the master moved among labeled integer schedules, and previously generated cuts need not be informative at schedules having similar gap patterns in different absolute slots. The experiments do not establish that this representation was the unique cause of slow progress. The conclusion is narrower: the valid cuts used by this implementation did not make full Benders attractive as a standalone algorithm on the tested semesters.

4.9.5 Combination-Based Student Subproblem

The assignment set Π_s distinguishes exam labels even when the cost depends only on sorted occupied slots. For $k_s = |E_s|$, let $\mathcal{T}_s$ contain sorted tuples

$$c = (t_1, \ldots, t_{k_s}), \qquad t_1 \leq \cdots \leq t_{k_s}.$$

The inequalities are strict when within-student conflicts are forbidden. Define $U_{s,c}$ by evaluating the complete sorted slot tuple c under the chosen personal-timetable objective. Because this value is stored as data, the reduction does not require an additive gap objective. It requires only that the personal cost depend on the occupied slots rather than on which exam label occupies each position, so every bijection between the exams and positions of c has cost $U_{s,c}$. Under the no-conflict

assumption, the two column counts are

$$|\Pi_s| = h(h-1)\cdots(h-k_s+1) = k_s!\binom{h}{k_s}, \tag{4.24}$$

$$|\mathcal{T}_s| = \binom{h}{k_s}. \tag{4.25}$$

Thus there are $k_s!$ assignment-level columns for each sorted slot tuple in the no-conflict case. Assignment-level linking is still needed to identify which exam occupies each slot, but costs can be computed and filtered before exam-to-position matchings are inspected. This reduction would not apply to a cost term that treats two of student s's exam identities differently despite placing them in the same sorted slot tuple.

4.9.6 Occupied-Slot Student Reformulation

The combination representation can be made into an exact integer student reformulation when direct conflicts are prohibited. Define

$$o_{s,t} = \sum_{e \in E_s} x_{e,t}. \tag{4.26}$$

For an integral conflict-free schedule, $o_{s,t}$ is binary and records whether student s has an exam in slot t. Let $k_s = |E_s|$ and define the conflict-free occupied-slot pattern set

$$\Omega_s = \{C \subseteq T : |C| = k_s\}.$$

Each $C \in \Omega_s$ corresponds uniquely to one strictly increasing tuple in $\mathcal{T}_s$, so $|\Omega_s| = |\mathcal{T}_s| = \binom{h}{k_s}$. Define $U_{s,C}$ as the cost $U_{s,c}$ of the corresponding sorted tuple. This definition is well posed under the label-invariant cost assumption above. For fractional master solutions, the hard-conflict constraints and (4.26) imply

$$0 \le o_{s,t} \le 1, \qquad \sum_{t \in T} o_{s,t} = k_s. \tag{4.27}$$

A fractional value does not represent a physical partial exam. It is a slot-occupancy marginal induced by the LP relaxation of the exam assignments.

Introduce $\eta_{s,C} \geq 0$, the convex coefficient assigned to occupied-slot pattern C. The extended student problem is

$$\phi_s(o_s) = \min_{\eta} \quad \sum_{C \in \Omega_s} U_{s,C} \eta_{s,C} \tag{4.28}$$

$$\text{s.t.} \quad \sum_{C \in \Omega_s : t \in C} \eta_{s,C} \leq o_{s,t} \qquad \forall t \in T, \tag{4.29}$$

$$\sum_{C \in \Omega_s} \eta_{s,C} = 1, \tag{4.30}$$

$$\eta_{s,C} \geq 0 \qquad \forall C \in \Omega_s. \tag{4.31}$$

Both sides of (4.29), summed over t, equal k_s, so every linking inequality is tight. Moreover, writing $\mathbf{1}_C \in \{0,1\}^h$ for the indicator vector of C and conv for the convex hull of a set of vectors,

$$\left\{ o \in [0,1]^h : \sum_t o_t = k_s \right\} = \text{conv}\{\mathbf{1}_C : C \in \Omega_s\}.$$

Indeed, a feasible point with two fractional coordinates is not extreme because a small amount can be shifted between those coordinates in either direction. Exactly one fractional coordinate is impossible because the coordinate sum is the integer k_s. Thus every extreme point is the indicator of a k_s-element subset of T, which is also the standard fixed-cardinality set polytope (Nemhauser and Wolsey, 1988). Thus every o_s satisfying (4.27) admits such a pattern decomposition. At an integer vector $o_s = \mathbf{1}_{C^*}$, the constraints force $\eta_{s,C^*} = 1$, and therefore $\phi_s(\mathbf{1}_{C^*}) = U_{s,C^*}$. The formulation is consequently exact at every conflict-free integer exam schedule. At a fractional occupied-slot vector o_s, $\phi_s(o_s)$ is the minimum average pattern cost over convex combinations of pattern vectors whose average occupancy is o_s; this is the convex closure of the pattern cost.

This aggregation is an exact integer reformulation, not an automatic strengthening of the assignment-level LP relaxation. At a fractional x, it retains only the aggregate marginals $o_{s,t}$ and discards which fraction of each labeled exam occupies each slot. Every feasible assignment-level mixture induces a feasible occupied-pattern mixture, but the converse need not preserve those exam-specific marginals. Consequently, $\phi_s(o_s(x))$ can be smaller than the assignment-level fractional value. The intended tradeoff is a lower-dimensional, label-invariant student representation, potentially at the cost of a weaker fractional bound.

Let μ_s be the free dual variable for (4.30), and let $\zeta_{s,t} \leq 0$ be the dual variable for the slot-linking inequality (4.29). The dual is

$$\max_{\mu_s,\zeta_s} \quad \mu_s + \sum_{t\in T} \zeta_{s,t} o_{s,t} \tag{4.32}$$

$$\text{s.t.} \quad \mu_s + \sum_{t\in C} \zeta_{s,t} \leq U_{s,C} \qquad \forall C \in \Omega_s, \tag{4.33}$$

$$\zeta_{s,t} \leq 0 \qquad \forall t \in T. \tag{4.34}$$

Hence every feasible solution of this dual gives a genuine Benders cut for the occupied-slot recourse formulation:

$$\theta_s \geq \mu_s + \sum_{t\in T} \zeta_{s,t} o_{s,t}. \tag{4.35}$$

After substituting (4.26), the coefficient $\zeta_{s,t}$ applies equally to every exam of student s assigned to slot t. The cut therefore applies to all $k_s!$ label permutations that produce the same occupied-slot pattern. It does not remove the original x variables or the coupling between students through shared exams, but it prevents the student dual from relearning the same spacing cost separately for each labeling.

The variables $o_{s,t}$ need not appear explicitly in the master. Because (4.26) defines them uniquely as linear functions of x, adding o and its linking equalities gives the same Benders cuts as substituting $\sum_{e\in E_s} x_{e,t}$ directly into (4.35). Thus

an "explicit occupied-slot master" is not, by itself, a stronger formulation. The substantive change is the use of the occupied-pattern recourse function and its slot-level dual, rather than whether the definitional variables o are written in the master.

Cut classification. Equation (4.23) is a conventional Benders optimality cut for the assignment-level value function f_s. Equation (4.35) is likewise a conventional Benders optimality cut, but for the alternative occupied-pattern value function ϕ_s. Because $\phi_s(\mathbf{1}_C) = U_{s,C}$, the latter is also valid for every integral, conflict-free schedule in the original problem. At a fractional x, however, $\phi_s(o_s(x))$ can be smaller than $f_s(x)$, so an occupied-slot cut should not be described as a cut obtained from the assignment-level LP relaxation.

Thus calling (4.35) only integer-valid understates its status inside the occupied-pattern extended formulation, where it is a genuine Benders cut valid over the fractional occupancy domain (4.27). Conversely, calling it an assignment-level Benders cut would overstate the connection. This classification depends on conflict freedom; if direct conflicts are allowed, $\sum_{e \in E_s} x_{e,t}$ is an exam count rather than the binary occupancy marginal used by Ω_s.

4.9.7 Implementation of the Reported Benders Runs

The semester-scale implementation uses the conflict-free occupied-slot formulation in Section 4.9.6 and the precomputed-dual cache described below.

When several students have the same exam set and the same timetable-cost table, the implementation aggregates them. Let $\mathcal{G}$ be the resulting groups, let E_g be group g's exam set, and let n_g be its number of students. The grouped master replaces $\sum_s \theta_s$ by $\sum_{g \in \mathcal{G}} n_g \theta_g$, solves one subproblem or performs one cache lookup

per group, and multiplies that value by n_g when computing an incumbent objective. The cuts are unchanged after replacing the student index by g. The semester-level counts and cached Benders runs below use this grouped version.

Solver-derived LP dual precomputation. For a fixed number k of exams and a common pattern-cost function, the constraint matrix and objective coefficients in (4.28)–(4.31) are the same for every student and every incumbent pattern. Only the right-hand side $o = \mathbf{1}_C$ changes. Define the common pattern set $\Omega_k = \{C \subseteq T : |C| = k\}$. The implementation therefore constructs one persistent occupied-pattern LP template for each k, changes its right-hand side for every $C \in \Omega_k$, solves the LP with Gurobi, and stores the returned optimal (μ_C, ζ_C) under the key (k, C). A later Benders iteration identifies the student's occupied pattern and retrieves its cost and dual coefficients without solving a new student problem.

More explicitly, an integer, conflict-free master solution $x^{(r)}$ determines

$$o_{s,t}^{(r)} = \sum_{e \in E_s} x_{e,t}^{(r)} \qquad \text{and} \qquad C_s^{(r)} = \{t \in T : o_{s,t}^{(r)} = 1\}.$$

The implementation retrieves the entry keyed by $(k_s, C_s^{(r)})$. Its cached coefficients define the occupied-slot cut, and substituting the occupancy definition translates that cut directly into the master variables:

$$\theta_s \geq \mu_{C_s^{(r)}} + \sum_{t \in T} \zeta_{C_s^{(r)},t} o_{s,t} = \mu_{C_s^{(r)}} + \sum_{t \in T} \zeta_{C_s^{(r)},t} \sum_{e \in E_s} x_{e,t}. \tag{4.36}$$

Thus Gurobi receives $\mathbf{1}_C$ as the LP right-hand side during precomputation and returns the cost and dual coefficients once. During the reported Benders runs, passing a master solution to the student side amounts to forming $C_s^{(r)}$, looking up those stored quantities, and adding (4.36).

The LP dual need not be unique. No special normalization is required for validity: any dual solution returned optimal by Gurobi satisfies (4.33), and strong

duality makes the resulting cut (4.35) tight at $\mathbf{1}_C$. Table 4.13 reports the complete precomputation through five exams, and Tables 4.14– 4.16 report its use in the semester-scale Benders loops.

4.9.8 Formulation Size

The tradeoff between the formulations is partly visible from their variable and constraint counts, summarized in Table 4.10. Let $M = |E|$ be the number of exams, $N = |S|$ the number of students, and $h = |T|$ the number of time slots. Let $k_s = |E_s|$, and let $J_s = |\Pi_s|$ be the number of personal timetable assignments for student s. Under the no-conflict assumption, Equations (4.24) and (4.25) give the assignment-level and occupied-slot column counts. If conflicts are allowed, then $J_s = h^{k_s}$. Let

$$J_{\text{tot}} = \sum_{s \in S} J_s.$$

Table 4.10: Approximate formulation sizes. Counts suppress constants and optional hard-conflict constraints. SP denotes student subproblem.

Formulation	Definition	Main variables	Main constraints
Full-space MIP	Eqs. (4.3)–(4.8)	$Mh + J_{\text{tot}}$	$M + N + \sum_s k_s h$
Benders master after cuts	Eqs. (4.10)–(4.15)	$Mh + N$	$M + \sum_s \lvert\mathcal{K}_s^{(r)}\rvert$
Assignment-level SP for s	Eqs. (4.16)–(4.19)	J_s	$k_s h + 1$
Occupied-slot SP for s	Eqs. (4.28)–(4.31)	$\binom{h}{k_s}$	$h + 1$

The full-space model is dominated by J_{tot}; Benders removes those variables from the master but retains large student subproblems and a growing cut set. The occupied-slot SP removes a factor of $k_s!$ from the no-conflict student columns and reduces the number of linking rows from $k_s h$ to h, with the fractional-relaxation tradeoff described above.

4.10 Implied Formulation Sizes for the Semester Instances

The instance statistics in Section 4.3.1 can be converted into approximate model sizes. Let $h = 24$ be the number of slots, M the number of unique exams, and $Q = |\mathcal{G}|$ the number of student groups defined above. The grouped Benders master has Mh assignment variables and Q student-cost variables if all groups are included. With hard student conflicts, the master has approximately $M + Qh$ base constraints before Benders cuts. Each Benders iteration adds at most one cut per optimized group, so a run optimizing all groups adds up to Q cuts per iteration.

The student subproblem burden depends strongly on the number k of exams in the group. With no within-student conflicts, Equations (4.24) and (4.25) give the assignment-level and occupied-slot column counts, respectively. Table 4.11 instantiates those counts for $h = 24$; their ratio is $k!$.

Table 4.11: Number of sorted slot combinations and assignment-level personal timetables by number of exams, for $h = 24$ slots and no within-student conflicts.

k	Sorted slot combinations $\binom{24}{k}$	Assignments $24!/(24-k)!$
1	24	24
2	276	552
3	2,024	12,144
4	10,626	255,024
5	42,504	5,100,480
6	134,596	96,909,120
7	346,104	1,744,364,160

The full counts are driven by the small number of students taking six or seven exams. For an exam-count threshold $k_{\max}$, define

$$S_{\leq k_{\max}} = \{s \in S : k_s \leq k_{\max}\}, \qquad \mathcal{G}_{\leq k_{\max}} = \{g \in \mathcal{G} : k_g \leq k_{\max}\},$$

where $k_g = |E_g|$. Let $o_g(x)$ be the occupied-slot vector induced by schedule x for

group g. The *exam-count–restricted objective* is

$$\Phi_{\leq k_{\max}}(x) = \sum_{g \in \mathcal{G}_{\leq k_{\max}}} n_g \phi_g(o_g(x)) \,,$$

whereas the full-population objective is

$$\Phi_{\text{all}}(x) = \sum_{g \in \mathcal{G}} n_g \phi_g(o_g(x)) \,.$$

Thus $\Phi_{\leq k_{\max}}$ includes student-cost contributions only from groups whose members take at most $k_{\max}$ exams. The reported experiments set $k_{\max} = 5$ and write the resulting sets as $S_{\leq 5}$ and $\mathcal{G}_{\leq 5}$. Students outside $S_{\leq 5}$ are not removed from the timetable: their exams remain in the assignment model, and their same-slot conflicts can still be prohibited by the hard-conflict constraints. Only their contributions to the restricted objective are omitted. Consequently, a lower bound for $\Phi_{\leq 5}$ does not certify a lower bound for the full-population objective Φ_{all}.

The grouped master for $\Phi_{\leq 5}$ has $Mh + |\mathcal{G}_{\leq 5}|$ variables because it retains one cost variable for each group included in the restricted objective. The Mh exam-assignment variables are unchanged. If direct conflicts are prohibited for every student, the base master also retains the approximately $M + Qh$ assignment and hard-conflict rows described above, including rows for groups outside $\mathcal{G}_{\leq 5}$. A complete iteration adds at most $|\mathcal{G}_{\leq 5}|$ cuts rather than Q.

Table 4.12 combines the semester-scale master and student-side views. Panel A reports the full conflict-free master before Benders cuts, including every observed student group. Panel B reports the restricted student-side workload used for groups with at most five exams. Its assignment-column and occupied-column counts are

$$J_{\leq 5}^{\text{assign}} = \sum_{g \in \mathcal{G}_{\leq 5}} \frac{24!}{(24 - k_g)!}, \qquad J_{\leq 5}^{\text{occ}} = \sum_{g \in \mathcal{G}_{\leq 5}} \binom{24}{k_g},$$

respectively. The corresponding row counts sum $24k_g + 1$ assignment-level rows or 25 occupied-slot rows per group. These are aggregate one-iteration workloads if

every group subproblem is instantiated; they are not the peak memory required when subproblems are processed sequentially.

Table 4.12: Semester-scale formulation sizes for $h = 24$. Panel A reports the full conflict-free master before Benders cuts. Panel B reports aggregate uncached student-side workload for groups with at most five exams. Coverage counts students represented in $\Phi_{\leq 5}$; all students remain subject to exam-assignment and hard-conflict constraints. "Assign." means assignment-level; numerical suffixes k, M, and B denote thousand, million, and billion and are unrelated to the model symbol $M = |E|$.

Panel A: Full master before Benders cuts

Semester	Master variables	Base constraints	All groups
SP24	23.7k	256.5k	10,665
FA23	26.0k	278.6k	11,584
SP23	23.0k	243.1k	10,105
FA22	25.0k	261.0k	10,852
SP22	23.7k	249.6k	10,378

Panel B: Student-side workload for $k_g \leq 5$

Semester	Included students	Coverage	Included groups	Assign. columns	Occupied columns	Assign./occupied rows
SP24	16,194	99.56%	10,594	4.696B	71.83M	795.1k / 264.9k
FA23	18,193	99.31%	11,461	7.375B	96.88M	890.4k / 286.5k
SP23	16,172	99.53%	10,030	4.690B	68.58M	746.6k / 250.8k
FA22	17,999	99.55%	10,771	5.736B	79.22M	812.0k / 269.3k
SP22	16,159	99.37%	10,276	4.914B	71.79M	770.3k / 256.9k

If every student uses the same occupied-slot cost function conditional on k_s, the cost data can be stored once for each pattern size rather than once per student or group. For $h = 24$ and $k \leq 5$, one shared occupied-pattern dictionary contains

$$\sum_{k=1}^{5} \binom{24}{k} = 55{,}454$$

cost entries. This sharing reduces precomputation and cache storage, but it does not by itself remove group-specific occupancy vectors, dual solutions, or master cuts. With sequential processing, the largest individual occupied-slot subproblem under the $k_{\max} = 5$ restriction is the $k = 5$ problem, with $\binom{24}{5} = 42{,}504$ columns and 25 rows. Model reuse can retain this common structure while changing the group-specific right-hand side. The corresponding largest assignment-level subproblem

has 5,100,480 columns and 121 rows. The implementation and master-variable mapping for this shared cache are described in Section 4.9.7.

The restriction retains between 99.31% and 99.56% of students in these semesters. Students taking six or seven exams have much larger pattern spaces and fewer ways to avoid the reported bad events in the Cornell objective, so they are reasonable candidates for separate treatment, an approximate objective term, or explicit exception review. The experiments in this chapter do not establish which treatment is best, and the restriction should not be interpreted as evidence that these students' schedules are unimportant.

Without the exam-count restriction, aggregate assignment-level enumeration across all groups would range from 13.61 to 23.04 billion timetable columns across the five semesters. These are scale indicators rather than objects constructed by the reported implementation. Together with Table 4.11, they show that the unrestricted counts are driven by the $k!$ exam-label permutations and the small number of groups with six or seven exams. The occupied-slot formulation removes those permutations from each student LP and permits one cost-and-dual cache to be shared across groups with the same pattern size.

4.11 Benders Performance and Bottleneck Analysis

The reported heuristic results in Section 4.4.7 provide the positive computational evidence of the chapter. The Benders study addresses three questions. First, does occupied-pattern precomputation make student-side evaluation inexpensive relative to solving the master problem? Second, once student evaluation is inexpensive, does the cut-augmented master make meaningful progress, as measured by incumbent improvement, lower-bound improvement, and cut growth relative to elapsed time? Third, does the resulting Benders procedure approach the solution quality and

runtime of the heuristic? The precomputation results, iteration profiles, timing breakdowns, and conservative objective comparison below address these questions in order.

4.11.1 Occupied-Pattern Precomputation

The occupied-slot experiments use 24 slots and the Cornell objective defined in Section 4.3.4. For each value of k separately, a pool of 12 spawned worker processes is created. Each worker constructs one persistent Gurobi LP template for that value of k, uses one solver thread, and repeatedly changes the right-hand side to process its assigned occupied patterns. The pool and its templates are recreated for the next value of k. Table 4.13 reports the wall-clock time required to solve and store a solver-derived optimal LP dual for every occupied pattern with one through five exams. The $k_{\max} = 5$ restriction includes 16,194 of 16,265 SP24 students (99.56%) and between 99.31% and 99.56% of students in the five semesters, as shown in Table 4.12.

Table 4.13: Complete solver-derived LP precomputation of Cornell-objective occupied-pattern costs and duals for 24 slots. For each k, precompute time is the elapsed wall-clock time required to solve and store the cost and dual for every occupied pattern in that row. Times include worker startup and persistent LP-template construction.

Exams k	Occupied patterns $\binom{24}{k}$	Labeled assignments $24!/(24-k)!$	Precompute time (s)
1	24	24	0.057
2	276	552	0.123
3	2,024	12,144	0.217
4	10,626	255,024	2.062
5	42,504	5,100,480	90.492
Total	55,454	5,368,224	93.575

The Cornell cache is 13.7 MB and loads in 0.188 seconds.

4.11.2 Five-Iteration Cornell-Objective Runs

Each semester run is independent and begins by constructing a new master model. That master is then retained for all five iterations, with new cuts added to the existing model rather than rebuilding it. The first master solve has no MIP start. Before each later solve, the preceding master incumbent is supplied as the start for the exam-assignment variables, and the current group-cost values are supplied as the start for the θ variables. No heuristic schedule or solution from another semester is used as a warm start. Each master solve uses Gurobi 13.0 with 12 configured threads, a target MIP gap of zero, and its own 300-second wall-clock limit; the limit is per master solve, not per five-iteration run. The complete occupied-pattern dual cache is loaded once at the beginning of each semester run. Consequently, the Benders loop performs cache lookups and adds the resulting cuts but does not rebuild or solve student LPs.

Four runs use the Cornell no-double-counting cost for occupied patterns and minimize the exam-count–restricted objective $\Phi_{\leq 5}$ over groups with $k_g \leq 5$ for five Benders iterations. The hard-conflict constraints still cover every observed student group. Thus groups whose members take six or more exams contribute to feasibility but not to $\Phi_{\leq 5}$. After every master solve, the implementation saves the complete exam-to-slot schedule and independently evaluates the full-population objective Φ_{all}. Table 4.14 gives the detailed SP24 trace.

Table 4.14: Five-iteration SP24 trace using the Cornell cost for occupied patterns. MP time is the master-problem solve time in seconds; Cuts is the cumulative number after adding that iteration's cuts; and LB is the valid master lower bound. "Restricted UB" is the best feasible value of $\Phi_{\leq 5}$, which includes only groups with $k_g \leq 5$; "full-population score" is Φ_{all}.

Iter.	MP time (s)	Cuts	LB	Restricted UB	Full-population score
1	3.770	10,594	0	14,789.0	15,180.0
2	300.022	17,106	0	9,679.5	9,937.5
3	174.784	18,465	0	9,567.0	9,844.0
4	144.414	18,465	0	9,567.0	9,844.0
5	300.032	19,088	0	9,530.0	9,785.5

Table 4.15 summarizes the best restricted and full-population objective values observed in each run. The iteration attaining the best full-population score need not attain the best restricted value because groups outside $\mathcal{G}_{\leq 5}$ contribute only to Φ_{all}.

Table 4.15: Best objective values observed in each five-iteration Cornell-objective run. "Best full iter." identifies the iteration attaining the best value of Φ_{all}. "Best restricted" is the lowest observed value of $\Phi_{\leq 5}$ and need not come from that iteration. "Initial full" and "best full" are values of Φ_{all}, with "initial full" evaluated after the first master solve. Time is end-to-end elapsed time from data loading and model construction through the fifth iteration, including cache loading but excluding the separate dual-cache precomputation.

Semester	Best full iter.	Initial full	Best restricted	Best full	Time (s)
SP22	5	10,404.0	8,642.5	9,068.0	1,211
SP23	3	16,433.0	8,127.0	8,384.0	798
FA23	5	18,569.0	11,693.5	12,220.0	1,184
SP24	5	15,180.0	9,530.0	9,785.5	929

Because all group costs are nonnegative, $\Phi_{\leq 5}(x) \leq \Phi_{\text{all}}(x)$ for every feasible schedule x. Thus a schedule's restricted objective is a lower bound on its own full-population objective. This schedule-level inequality is distinct from the valid Benders lower bound, which remained zero in these runs. Table 4.16 uses the inequality to give Benders a conservative advantage in its comparison with the fixed unconstrained heuristic: it compares Benders's best restricted value with the

heuristic's larger full-population score.

Table 4.16: Timing and conservative objective comparison. "Benders restricted" is the best observed value of $\Phi_{\leq 5}$, whereas "heuristic full" is the heuristic's Φ_{all} value. Since $\Phi_{\leq 5}(x) \leq \Phi_{\text{all}}(x)$, this comparison favors Benders. Benders time is the sum of the five iteration wall times and excludes initial data loading, model construction, and the reusable 93.6-second dual-cache construction. Heuristic time is cumulative from data loading through local search. "Lookup" is aggregate cache-lookup time over all five Benders iterations. "MP share" is the percentage of that Benders iteration-loop time spent solving the master problem.

	Benders, five iterations				Heuristic	
Semester	Time (s)	MP share	Lookup (s)	Best restricted	Time (s)	Full score
SP22	1,207	99.7%	0.024	8,642.5	114.1	4,740.0
SP23	794	99.6%	0.019	8,127.0	113.8	4,348.0
FA23	1,182	99.8%	0.031	11,693.5	127.9	5,431.0
SP24	925	99.8%	0.017	9,530.0	108.7	4,820.5

Table 4.17 reports the cross-semester iteration data underlying the progression claims below.

Table 4.17: Cross-semester progression of the five-iteration Cornell-objective Benders runs. "Initial MP" is the iteration-1 master solve before student cuts; "initial cuts" are added after that solve; "later MP" sums master-solve time over iterations 3–5. "Later improvement" compares the best full-population score in iterations 3–5 with the iteration-2 score. "LB range" is the range of valid Benders lower bounds across all five iterations.

Panel A: Master timing and cut growth

Semester	Initial MP (s)	Initial cuts	Iteration-2 MP (s)	Later MP (s)	Final cuts
SP22	2.903	10,276	300.029	901.141	20,699
SP23	3.690	10,030	300.069	487.700	22,368
FA23	3.007	11,461	300.078	876.271	21,147
SP24	3.770	10,594	300.022	619.229	19,088

Panel B: Full-population score progression and bound

Semester	Iteration-2 score	Best later score	Later improvement	LB range
SP22	9,818.5	9,068.0	7.6%	0–0
SP23	8,409.5	8,384.0	0.3%	0–0
FA23	12,333.5	12,220.0	0.9%	0–0
SP24	9,937.5	9,785.5	1.5%	0–0

As shown in Table 4.17, the initial masters, before student cuts were present,

solved in 2.9–3.8 seconds. After the first 10,030–11,461 cuts were added, every second-iteration master reached the 300-second limit. Master optimization subsequently accounted for 99.6%–99.8% of measured loop time (Table 4.16). By contrast, the complete reusable Cornell cache of 55,454 pattern duals required 93.6 seconds to construct, less than one time-limited master solve, and all cache lookups within a semester required at most 0.031 seconds in aggregate. Precomputation is not required by the algorithm: the same occupied-pattern LPs could instead be solved and cached on demand, although these runs do not measure that implementation.

Most incumbent improvement occurred in the first cut-augmented master. Table 4.17 shows that, after iteration 2, the runs spent a further 488–901 seconds in master optimization, while the best full-population score improved by 0.3% on SP23, 0.9% on FA23, 1.5% on SP24, and 7.6% on SP22. The lower bound remained zero as the models grew to 19,088–22,368 cuts. Thus the experiments show slow or stalled progress after the initial improvement, rather than uniform stalling on every instance.

Even under this comparison favorable to Benders, its best restricted objective values were 82%–115% higher (worse) than the heuristic's full-population scores. The heuristic's restricted scores can only be smaller than its reported full-population scores, so the heuristic also outperformed these Benders runs on $\Phi_{\leq 5}$, the objective that Benders directly minimized. The Benders loops took 7.0–10.6 times as long even before charging cache construction. This comparison does not show that no Benders implementation can be competitive; it shows that the implemented cut sequence was substantially slower and produced worse schedules than the reported heuristic runs.

The best full-population score was 12.8%–49.0% below the first-iteration score, with an unweighted mean reduction of 32.9% across the four semesters. The

Benders lower bound nevertheless remained zero in every iteration of every run. Thus additional iterations produced better candidate schedules but no nontrivial certificate.

Together, these results isolate the practical limitation of the tested Benders implementation. Occupied-pattern precomputation makes student evaluation negligible, but this reduction does not produce an effective overall algorithm because the cut-augmented master dominates runtime while producing no lower-bound progress. Moreover, even when Benders is evaluated using its smaller restricted objective and the heuristic is evaluated using the larger full-population objective, Benders produces substantially worse values in substantially more time. The computational priority is therefore not faster student subproblems, but a master formulation or cut strategy that converts inexpensive student evaluations into meaningful bound and incumbent progress.

4.12 Future Work

The most direct next step is to improve the cut-augmented master problem. Section 4.11.2 shows that occupied-pattern precomputation makes student evaluation negligible in the tested runs, but the master becomes slow after thousands of cuts are added. Promising directions include selecting, aggregating, or deleting cuts; strengthening symmetry handling in the master; using alternative master representations; and adapting solver tolerances as the cut model develops. Controlled experiments should measure lower-bound improvement per second, master growth, cut activity, and reuse across structurally similar schedules.

Future work could also exploit limited translation symmetry by identifying occupied-slot patterns that differ only by a common shift in slot indices. Such a formulation would need to preserve calendar boundaries and relevant opera-

tional restrictions, including front-loading, which prevent unrestricted translation invariance.

Another possible future use of Benders is local rather than global. Instead of running full Benders from scratch, one could first generate candidate schedules with the primal heuristic and then use Benders-style cuts to bound or improve restricted neighborhoods around those schedules. For example, most exams could be fixed at their heuristic assignments while Benders optimizes a selected subset of high-conflict exams, a time window, or the exams contributing most to back-to-back and two-in-three penalties.

This direction matches the computational lesson of the chapter. The heuristic produced schedules quickly, while the Cornell-objective Benders runs made no lower-bound progress in the labeled binary master. Local Benders would use the decomposition where it is more plausible: as a way to bound restricted improvements around good candidate schedules, rather than as a standalone solver for the entire semester.

4.13 Conclusion

The practical contribution of this chapter is the implementation and Cornell-data benchmarking of a fast scheduling heuristic whose main components have precedents in the related literature. Pairwise conflict scores guide construction and move proposals, while strict local-search acceptance uses the overlapping-event objective (4.1), including an explicit triple penalty. The reported runs produced complete conflict-free schedules in about two minutes and were competitive with Cornell Hybrid under Cornell's default weights. These single-replication results are empirical observations, not guarantees, and the configuration was not systematically tuned.

Front-loading was preserved successfully on all five semesters, whereas the separate daily-cap repair succeeded only on the three spring semesters within its move limit. The heuristic is therefore a promising candidate-generation or post-processing component, not a replacement for IP-based methods. Additional interacting requirements may demand constraint-specific construction, repair, and neighborhoods.

The exact formulation and Benders study provide the complementary result. Integer master schedules give immediate student primal solutions, and sorted-combination and occupied-slot views expose a student-side symmetry hidden by labeled exam-slot variables. Precomputing the costs and solver-returned duals for all 55,454 occupied patterns through five exams takes 93.6 seconds. Across four five-iteration semester runs, aggregate lookup requires at most 0.031 seconds, while master optimization accounts for 99.6%–99.8% of measured loop time.

These runs directly minimize the restricted objective containing student groups whose members take at most five exams. That objective is a lower bound on the corresponding full-population objective for any fixed schedule. Even in a conservative comparison that gives Benders this advantage, its best restricted-objective values are 82–115% worse than the heuristic's full-population scores, and the Benders loops require 7.0–10.6 times as much computation. The Benders lower bound also remains zero throughout. The occupied-pattern reformulation therefore makes student-side evaluation inexpensive but does not make the global master effective. Future work should focus on stronger master formulations and cut strategies and on using Benders locally to certify or improve neighborhoods around strong heuristic schedules.

CHAPTER 5

STUDENT PREFERENCES IN EXAM SCHEDULING: HETEROGENEITY IN THE TRADEOFF BETWEEN SPACING AND FINISHING EARLY

5.1 Introduction

The final exam schedule is one of the few academic decisions a university imposes on every student at once, and it is far from neutral: where exams land determines how much study time separates them, how compressed the exam period is, and how long a student must stay on campus before leaving. Schedules distribute time pressure, compression, and delayed departure across the student body. How those costs *should* be distributed is an assessment-policy question, but in practice it is settled implicitly, as a by-product of how the schedule is computed.

Final exam scheduling at large universities is typically formulated as a combinatorial optimization problem whose objective penalizes a weighted list of undesirable schedule features, including direct conflicts, back-to-back exams, two-in-three clustering, and late finish dates. Utility or penalty models built from exactly these kinds of schedule metrics are well studied in the operations-research timetabling literature, and at Cornell University an optimization of this form is used in practice to construct the final exam schedule. Two modeling choices are embedded in that practice and are the subject of this paper. First, the objective operates at the *population* level: a single weight vector is applied to all students, so individual differences in preference play no role. Second, the weights are *chosen by the exam-scheduling planning team* rather than estimated from student preference data. Together these mean the schedule is optimized against an assumed, common utility model rather than a measured, heterogeneous one. If students do not in fact share

that utility, the objective is quietly distributing exam burden according to one group's preferences while treating the result as neutral.

This work provides a quantification of the student preferences that such an objective requires. We collect ranking data from a survey of $N = 981$ Cornell students over designed exam-schedule scenarios.

5.1.1 Research questions

This study asks three research questions:

1. **RQ1:** Are there systematic individual differences in student preferences that a population-level exam-scheduling objective misses?
2. **RQ2:** Do College of Engineering students differ from non-COE students in their exam-schedule preferences, and if so, on which tradeoffs?
3. **RQ3:** What relative importance do students place on different undesirable schedule features, and how do these estimates compare with weights or survey evidence used in existing optimization systems?

To address these questions with one apparatus we fit a nested family of hierarchical Plackett–Luce choice models over schedule features used in practice. The *population* model gives every student one shared weight vector, the in-spirit analogue of the current planner-chosen objective. The *covariate* model lets those shared weights vary by COE status and class year. The *individual* model gives each student their own random-coefficient utility. Because the three models are strictly nested, the incremental value of demographic information and of individual heterogeneity can each be isolated and, crucially, measured *out-of-sample* by cross-validation rather than by in-sample fit, which a flexible per-student model would inflate.

Our goal is not to propose a new scheduling algorithm, but to shed light on the preferences optimization systems require: measuring the weights, testing whether a single weight vector is adequate, and showing where the common objective hides meaningful disagreement among students.

The findings, each addressing one of the research questions in Subsection 5.1.1, are:

1. **Individual effects are real and a population model misses them.** In both a held-out-question and a held-out-response cross-validation, the individual model predicts held-out rankings better than the population model. A single population weight vector, the current modeling choice, therefore discards systematic, generalizable student-level structure.
2. **COE and non-COE students differ on one specific tradeoff.** COE students prefer greater spacing at the cost of later exams compared to non-COE students. The difference is specific to that tradeoff: the groups do not differ materially on time-of-day preferences or on the ordering of bunching penalties.
3. **Student rankings provide empirical weights for scheduling.** The model estimates the relative disutility of bunching events: same-day back-to-back has the largest posterior mean, while evening–morning back-to-back and two-in-three each have posterior means roughly two-thirds as large. These estimates can be compared with planner-chosen or survey-informed weights used in existing optimization systems.

We also contribute a ranking dataset of 981 Cornell students over 20 designed schedule variants with high completion and attention-check rates.

5.2 Background and Related Work

Optimization systems assume objective weights. The exam-timetabling literature in operations research optimizes schedules against an objective built from schedule-feature penalties, including direct conflicts, back-to-backs, dense exam clusters, spread, and sometimes preferences over particular slots or dates, treating the penalty weights as exogenous inputs. Carter (1986) surveys practical applications and the standard objective structure. Cornell University's own system is a recent and especially relevant example: Ye et al. (2024) develop an optimization framework that assigns hundreds of exams to time slots, replacing the older lecture-time table with a multi-stage Group-then-Sequence model and a Layer-Cake heuristic. Their objective minimizes a weighted sum of "bad events" such as direct conflicts, back-to-backs, two exams in 24 hours, three exams in a row, and three exams in four successive slots; varying parameters and solution heuristics help the Registrar compare schedules that make different tradeoffs in schedule comfort. Those weights are operational choices, made with Registrar constraints and institutional priorities in view as well as student and faculty comfort, rather than estimates of student preferences alone. This is exactly the operational setting our paper takes as its starting point. The optimization model makes the tradeoffs explicit and schedulable, but the weights remain external to measured student preferences and are applied at the population level. Two gaps therefore motivate the present work: first, the objective is uniform across students, with no representation of individual or group differences; second, the weights encode an assumed allocation of exam burden rather than a measured distribution of how students trade off spacing against finishing early. Most methodological effort in the area targets the algorithmic side (optimization formulations and heuristics); our contribution is to measure the preference inputs that such objectives implicitly require. In the same

spirit, Bullnheimer (1998) formulate a small-scale exam-scheduling optimization model that maximizes aggregate student study time under the assumption that students study one exam at a time. The objective uses a distance function that assigns a negative value to same-slot conflicts, zero to back-to-backs, and a power-of-time-difference reward otherwise; the exponent is a free parameter rather than a quantity estimated from student preference data.

Preference surveys identify inconveniences more than utility weights. Chaplin et al. (2026) provide the closest recent comparison. They survey 226 students at Bucknell University, asking them to rate nine isolated exam-schedule inconveniences as fine, inconvenient, or extremely inconvenient. The survey shows broad agreement that bunching is costly because overlaps, three exams in 24 hours, four exams in 48 hours, same-day back-to-backs, and night-to-morning exams are rated as the most inconvenient, and the authors use these descriptive responses to choose inconvenience categories and build a survey-informed weighting scheme for an optimization-based scheduler. Their results also foreshadow the group difference we study: engineering students were less likely than students in Arts & Sciences or Management to rate an exam on the last day as extremely inconvenient, suggesting that willingness to remain longer in the exam period may vary by academic context. Our survey differs in the form of elicitation and in the object of inference. Rather than rating isolated inconveniences one at a time, students rank complete schedule alternatives, which forces the spacing-versus-finishing-early tradeoff to be expressed directly. The hierarchical Plackett–Luce model then estimates tradeoff weights statistically and, more importantly for the present paper, separates population averages, observed group shifts, and individual heterogeneity that can be tested out-of-sample. Muklason et al. (2017) surveys 50 undergraduate and taught-postgraduate students at the University of Nottingham

on exam-schedule preferences. Key findings relevant to our design: students strongly dislike exams within 24 hours of each other ($\sim$90% agreement); 74% prefer exams spread out evenly; 40% prefer a longer total period for better spacing while only 12% prefer the opposite; 54% avoid weekend exams; and when two exams fall in a three-day window, students prefer a larger gap when the second exam is harder. Their elicitation uses penalty scores for individual scenarios rather than full-schedule rankings, and their sample is too small ($n = 50$) to separate shared weights from individual heterogeneity. Funkhouser and Blenda (2024) surveys medical students at a US allopathic medical school and reports Likert-scale agreement with statements about clustering and last-exam timing; the methodology is descriptive.

Choice models turn rankings into heterogeneous weights. The Plackett–Luce model (Plackett, 1975; Luce, 1959) is the ranking-data analogue of the multinomial logit and is the natural likelihood when observations are full orderings over a small choice set. Random-coefficient (mixed) logit and its hierarchical Bayesian extensions (Train, 2009) model individual heterogeneity through person-specific utility weights; our individual model is the Plackett–Luce ranking analogue. Relative to that literature our contribution is empirical and institutional: we show, by cross-validation on a large sample, that per-student exam-schedule coefficients carry genuine out-of-sample signal, and we connect those coefficients to an operational university scheduling objective.

5.3 Methods: Survey Design and Participants

5.3.1 Schedule representation

An exam schedule is a set of occupied slots on a 10-day $\times$ 3-slot-per-day grid (9 am, 2 pm, 7 pm), for 30 slots total. A student's schedule with n exams occupies n of these slots. We index slots sequentially so that slot 1 is Day 1 at 9 am, slot 2 is Day 1 at 2 pm, ..., slot 30 is Day 10 at 7 pm.

5.3.2 Schedule features

For each schedule we compute six features:

1. **Evening–morning back-to-back** (x^{eve}): number of occupied consecutive slot pairs $(s, s+1)$ where s is a 7 pm slot ($s \bmod 3 = 0$). This is the number of 7 pm-followed-by-next-morning-9 am pairs.
2. **Other back-to-back** (x^{oth}): number of occupied consecutive slot pairs $(s, s+1)$ where s is *not* a 7 pm slot. This is the number of same-day back-to-backs (9 am$\rightarrow$2 pm or 2 pm$\rightarrow$7 pm).
3. **Two-in-three** (x^{2in3}): number of occupied slot pairs $(s, s+2)$, i.e., two exams separated by one intervening slot. Bunching counts may overlap: three consecutive exams would contribute one two-in-three pair and two back-to-back pairs. None of the survey schedules contains three consecutive exams, so this overlap does not occur in the analyzed questions.
4. **Last exam time** (x^{last}): index of the last occupied slot divided by 30. This expresses last-exam timing as a fraction of the 10-day exam period; moving one day later increases the feature by $3/30 = 0.1$.
5. **Number of 2 pm exams** (x^{2pm}): count of occupied 2 pm slots.

6. **Number of 7 pm exams** ($x^{7\text{pm}}$): count of occupied 7 pm slots.

Because adding the same constant to every option's utility does not change its ranking probability, time-of-day preferences are identified only relative to a baseline. We set the 9 am utility to zero and estimate the 2 pm and 7 pm utilities relative to it. Features are otherwise used in raw form (not standardized) so that "last exam time" has a consistent meaning across 2-exam and 4-exam questions.

5.3.3 Questions

Each respondent answers six ranking questions, numbered Q1–Q6 in the order presented. The full set of schedules shown in each question is reproduced in Appendix Figure 5.3.

A calibration and validation design. The six questions are arranged as two blocks. The first three questions are *anchor questions*: simple, stylized tasks meant to make one preference dimension especially visible. The last three questions are *application questions*: more realistic 4-exam schedules that combine these dimensions and serve as holdout validation tasks (Train, 2009). The goal is to estimate interpretable preferences from the anchor questions and then test whether those preferences jointly predict the application questions. We are not trying simply to learn whether students dislike bunching or prefer earlier finishes; we expect those directions to be fairly clear. The harder question is the *relative strength* of these preferences: how much later a finish a student will tolerate to avoid a back-to-back, or how a mild time-of-day preference trades off against spacing. The anchor questions are therefore designed as simple comparisons in which one tradeoff is easier to see:

- **Q1** (time-of-day): rank the three daily slots, 9 am vs. 2 pm vs. 7 pm. With

no gap structure to confound them, this estimates the *strength* of afternoon and evening preferences relative to a 9 am baseline.

- **Q2** (bunching): rank four 2-exam schedules contrasting same-day back-to-back, evening–morning back-to-back, two-in-three, and well-separated. By pitting the bunching types against each other this helps estimate their *relative* penalties, such as how much worse a same-day back-to-back is than a two-in-three, and so on.
- **Q3** (scheduling horizon): rank four 2-exam schedules that trade maximum spacing against an earlier finish date. This question focuses on the tradeoff between two attractive goals: more spacing versus finishing all exams sooner.

The last three application questions are realistic 4-exam schedules that combine all of these features at once:

- **Q4** (4 exams): degree of spacing holding finish date approximately fixed.
- **Q5** (4 exams): spacing with a different spread of options.
- **Q6** (4 exams): 4-exam spacing vs. finish date.

5.3.4 Deployment and participants

The survey was hosted on Qualtrics and distributed via Cornell's Registrar to enrolled undergraduates in Spring 2026 (responses were collected from March 13 through March 19, 2026). The survey included two embedded attention-check items. The study was approved by Cornell's IRB; no identifying information was retained.

Of 983 respondents who opened the survey, 981 completed it (completion rate 99.8%); the two dropouts occurred before any ranking question and are excluded from analysis. Median completion time was 6.5 minutes (mean 74 minutes, right-skewed by long idle sessions).

Respondent demographics

The survey begins by asking students their expected year of graduation, their college, and which engineering major (if any) they are enrolled in. Of the 981 respondents, 287 are engineering students in the College of Engineering. Cornell enrolls roughly 3,000–3,200 COE undergraduates out of about 16,000 undergraduates total, i.e. roughly 13,000 non-engineering undergraduates. As Table 5.1 shows, the estimated COE response rate is higher than the non-COE rate ($\approx$ 9–9.6% vs. $\approx$ 5.2%).

Appendix Table 5.6 reports the joint distribution of college and graduation year, with the broader sample composition summarized in Appendix Figure 5.4. The three largest colleges (Arts & Sciences, Engineering, and Agriculture & Life Sciences) account for 79.9% of responses; COE contributes 287 students (29.3%). COE and non-COE class-year distributions are not significantly different (χ^2 test of independence, $p > 0.3$), so class year is not confounded with college. Appendix Table 5.7 breaks the COE subsample down by major; the largest are Mechanical Engineering, Computer Science, and Chemical and Biomolecular Engineering. Majors are self-reported and multi-select, so a double-major is counted under each of their majors and the percentages, taken over the 287 COE respondents, sum to more than 100%.

Group	Undergraduate population	Respondents	Response rate
College of Engineering	$\approx$ 3,000–3,200	287	$\approx$ 9.0–9.6%
Non-engineering	$\approx$ 13,000	682	$\approx$ 5.2%
Total undergraduate	$\approx$ 16,000	969	$\approx$ 6.1%

Table 5.1: Survey response rates by College of Engineering status. Population figures are approximate Cornell undergraduate enrollment; respondent counts are the 969 undergraduates in the analysis sample. The COE response rate is roughly 1.7$\times$ the non-engineering rate.

Embedded attention checks were passed at high rates throughout (above 96.5%

on every question), with COE students answering at least as carefully as their peers (Appendix Figure 5.5); we retain all 981 responses.

5.4 Analytic Approach and Validation

5.4.1 Model overview

The models all describe the same object: a student's preferences over exam schedules. For each student i, we represent these preferences with a six-dimensional latent preference vector $\mathbf{r}_i$. The three models differ only in how much structure they allow in this vector:

$$\mathbf{r}_i = \underbrace{\boldsymbol{\mu}_0}_{\text{population}} + \underbrace{\mathbf{B}^\top \mathbf{x}_i}_{\text{+ covariate effects}} + \underbrace{\mathbf{L}\,\mathbf{z}_i}_{\text{+ individual deviation}}, \qquad \mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_6). \tag{5.1}$$

The population model keeps only $\boldsymbol{\mu}_0$, so every student shares one preference vector. The covariate model adds $\mathbf{B}^\top \mathbf{x}_i$, allowing COE status and class year to shift preferences. The individual model adds $\mathbf{L}\mathbf{z}_i$, allowing each student to deviate from the population and covariate means. These nested models let us ask whether observed group differences and student-specific preferences improve prediction beyond a single common scheduling objective.

Here $\boldsymbol{\mu}_0 \in \mathbb{R}^6$ is the grand-mean latent vector. The student information in $\mathbf{x}_i$ records whether student i is in the College of Engineering and their expected graduation year. Each of these variables is coded as 0 or 1: the COE indicator is 1 for engineering students and 0 otherwise, and the class-year indicators mark whether the student expects to graduate in 2027, 2028, or 2029. Students in the class of 2026 are the reference group, so they have all three class-year indicators set to 0. Thus $P = 4$: one COE indicator and three class-year indicators. The matrix $\mathbf{B} \in \mathbb{R}^{P \times 6}$ holds the estimated shift associated with each of these student

characteristics on each latent preference dimension, and $\mathbf{L} = \text{diag}(\boldsymbol{\sigma})\mathbf{L}_{\text{corr}}$ is the Cholesky factor of the 6×6 covariance matrix for the individual deviations. The entries of $\boldsymbol{\sigma}$ are the across-student standard deviations for the six latent preference dimensions; larger values mean more student-to-student variation on that preference. We use the non-centered form $\mathbf{L}\mathbf{z}_i$ for sampling efficiency (Papaspiliopoulos et al., 2007).

Zeroing the trailing blocks gives two strictly nested submodels:

$$\underbrace{\text{population}}_{\mathbf{B}=\mathbf{0},\,\mathbf{L}=\mathbf{0}} \subset \underbrace{\text{covariate}}_{\mathbf{L}=\mathbf{0}} \subset \text{individual (full)}.$$

The covariate-versus-population restriction $\mathbf{B} = \mathbf{0}$ is summarized by posterior credible intervals (CrIs) on $\mathbf{B}$. The individual-versus-covariate restriction $\mathbf{L} = \mathbf{0}$ is a boundary restriction because it sets variance components to zero ($\sigma_d \geq 0$). A likelihood-ratio test of a variance component on the boundary does not have the usual χ^2 null distribution (Self and Liang, 1987), so we assess the individual step by held-out predictive performance (Subsection 5.5.4).

5.4.2 Utility model

The latent vector $\mathbf{r}_i$ is not used directly in the schedule utility. First, it is mapped into six interpretable weights. The first three weights are bunching penalties and must be non-negative. We enforce this with the softplus function, $\text{softplus}(a) = \log(1 + \exp(a))$, which maps any real number to a positive value:

$$g(\mathbf{r}_i) = \big(\underbrace{\text{softplus}(r_{i1})}_{w_{\text{eve},i}}, \underbrace{\text{softplus}(r_{i2})}_{w_{\text{oth},i}}, \underbrace{\text{softplus}(r_{i3})}_{w_{\text{2in3},i}}, \underbrace{r_{i4}}_{u_{\text{2pm},i}}, \underbrace{r_{i5}}_{\lambda_i}, \underbrace{r_{i6}}_{\eta_i}\big). \tag{5.2}$$

The resulting weights enter a single utility template shared by all three models. The utility student i assigns to option j in question q is

$$U_{iqj} = \frac{1}{\tau_q}\Big(- w_{\text{eve},i}\, x_{qj}^{\text{eve}} - w_{\text{oth},i}\, x_{qj}^{\text{oth}} - w_{\text{2in3},i}\, x_{qj}^{\text{2in3}} + \lambda_i\, x_{qj}^{\text{last}} + u_{\text{2pm},i}\, x_{qj}^{\text{2pm}} + \eta_i\, x_{qj}^{\text{7pm}}\Big), \tag{5.3}$$

where $w_{\text{eve},i}, w_{\text{oth},i}, w_{\text{2in3},i} \geq 0$ are bunching penalties; $u_{\text{2pm},i}$ is an afternoon preference; η_i is an evening-exam preference; λ_i is a scheduling-horizon preference (positive λ_i means the student prefers a later last exam, i.e. more time on campus in exchange for more spacing); and $\tau_q > 0$ is a question temperature, defined in Subsection 5.4.3. In words, a schedule receives lower utility when it has more bunching. It receives higher or lower utility depending on that student's time-of-day preferences and their preference for a later, more spread-out exam period.

5.4.3 Question temperatures and ranking likelihood

The temperature τ_q controls how sharply rankings follow the utility equation. Smaller values mean the ranking is closely tied to the utility differences; larger values mean the same utility differences produce noisier rankings. We estimate temperatures by question block:

$$\tau_q = \begin{cases} \tau_{\text{time}}, & q = \text{Q1}, \\ \tau_{\text{bunch}}, & q = \text{Q2}, \\ \tau_{\text{horizon}}, & q = \text{Q3}, \\ \tau_{\text{complex}}, & q \in \{\text{Q4}, \text{Q5}, \text{Q6}\}. \end{cases}$$

Thus each anchor question has its own temperature, while the three application questions share one temperature. This does not assume that anchor questions are noiseless. It lets the model learn whether the simpler anchor questions produce

sharper rankings than the more complex 4-exam application questions; the fitted values are reported in Subsection 5.5.3.

Given the utilities, a student's ranking is modeled with a Plackett–Luce likelihood. For a ranking $\boldsymbol{r}_{iq} = (r_1, r_2, r_3, r_4)$ of four options, where r_k denotes the option placed in rank k, the probability is

$$P(\boldsymbol{r}_{iq} \mid U_{iq\cdot}) = \prod_{k=1}^{3} \frac{\exp(U_{iqr_k})}{\sum_{\ell=k}^{4} \exp(U_{iqr_\ell})}. \tag{5.4}$$

Because U_{iqj} in Equation 5.3 already divides the unscaled schedule score by τ_q, each step of the ranking likelihood is a softmax over temperature-scaled scores. Equivalently, if V_{iqj} denotes the expression inside the parentheses in Equation 5.3, then the first-choice probability is

$$P(r_1 = j) = \frac{\exp(V_{iqj}/\tau_q)}{\sum_m \exp(V_{iqm}/\tau_q)}.$$

Thus smaller τ_q magnifies utility differences and larger τ_q flattens the choice probabilities toward more random rankings. For Q1 the slot utilities are $(0, u_{2\text{pm},i}, \eta_i)/\tau_{\text{time}}$ with 9 am fixed at zero, which identifies η_i free of gap structure. The total log-likelihood for student i sums the five schedule questions and, when observed, the time question.

5.4.4 Priors and estimation

We use weakly informative priors as regularizers. The first three population means are centered away from zero, reflecting the design expectation that bunching is disliked while still allowing the corresponding penalties to be close to zero. Other location effects are centered at zero. Scale parameters use positive weakly informative priors, and the four block temperatures share a hierarchical prior so they can differ while still being partially pooled. Full prior specifications are given in Appendix 5.10.1. Posterior inference uses the No-U-Turn Sampler (NUTS; Hoffman

and Gelman, 2014) in NumPyro (Phan et al., 2019), with 2 chains of 300 warmup and 500 sampling iterations and target acceptance 0.9. Sampling diagnostics are reported in Appendix 5.10.2.

5.4.5 How the anchor questions map to model parameters

Subsection 5.3.3 described the anchor questions as simple comparisons that make one tradeoff easier to see. In the model, the same idea can be written as a decomposition of the schedule weights. Let s be an overall bunching scale and let $\boldsymbol{\pi}$ be the relative bunching profile, so that $w_k = s\,\pi_k$. Then write the scheduling-horizon weight as $\lambda = s\,\rho$, where $\rho = \lambda/s$ is the spacing preference expressed in units of bunching sensitivity. When we use a specific bunching penalty as the unit, we write the same derived quantity as $\rho_{i,k} = \lambda_i/w_{i,k}$ for student i or $\rho_{g,k} = \lambda_g/w_{g,k}$ for group g. This ρ notation is not a new model parameter; it is a scale-free way to describe how much later a student or group is willing to finish for more spacing, relative to their own bunching aversion. With this notation, the survey blocks emphasize different pieces of the preference vector:

- **Q1** estimates the time-of-day utilities $(\eta, u_{2\text{pm}})$.
- **Q2** emphasizes the relative bunching profile $\boldsymbol{\pi}$ and, together with its temperature, the bunching scale s.
- **Q3** emphasizes ρ, the scheduling-horizon preference normalized by bunching sensitivity. This is the scale-invariant ratio used in Subsection 5.5.5.
- **Q4**–**Q6** combine these components in more realistic schedules, providing the main validation test.

This decomposition also explains why the COE effect in Subsection 5.5.5 is reported as $\rho = \lambda/s$: it asks how much later a group is willing to finish for more spacing,

measured relative to that group's own bunching sensitivity.

5.4.6 Predictive-validation metrics

We evaluate held-out rankings with: **log-likelihood per observation** (LL/obs, where LL abbreviates log-likelihood; for a four-option ranking the uniform-ranking null gives $\ln\frac{1}{4} + \ln\frac{1}{3} + \ln\frac{1}{2} \approx -3.178$); **top-1 accuracy** (highest-utility option matches the student's first choice; chance 25% for the four-option schedule questions); **bottom-1 accuracy** (lowest-utility option matches the last choice; chance 25% for the four-option schedule questions); and **pairwise accuracy** (chance 50%), the fraction of option-pairs whose model ordering agrees with the student's, equivalently Kendall's τ up to the affine map $\tau = 2 \times (\text{pairwise accuracy}) - 1$, so the reported 70% pairwise corresponds to $\tau \approx 0.4$. The time-of-day question Q1 has three options rather than four, so its chance top-1 rate is 33% and its uniform log-likelihood is -1.79; Appendix Table 5.12 reports that case separately. When comparing the three nested models on held-out data, we use each model's latent-vector predictor: $\mathbf{r}_i^{\text{ind}}$, $\mathbf{r}_i^{\text{cov}} = \boldsymbol{\mu}_0 + \mathbf{B}^\top \mathbf{x}_i$, or $\mathbf{r}_i^{\text{pop}} = \boldsymbol{\mu}_0$. We map that vector to utility weights with Equation 5.2 and score the resulting rankings with Equation 5.4.

What the metrics measure, and why we report several. These metrics are not interchangeable; they probe different parts of the same underlying object, a utility function that should order *whole schedules* for a student, and a scheduling objective cares about the whole order, not just the single favorite. Top-1 accuracy checks only the highest-utility option and bottom-1 only the lowest, so each verifies one extreme position; they are easy to read but coarse, and a model can win on the modal first choice while misordering everything below it (this is exactly why the modal baseline is competitive on top-1). Pairwise accuracy is stricter: it scores

all $\binom{4}{2} = 6$ ordered relations within a ranking, so it rewards getting the *shape* of the whole order right, not just its endpoints. Log-likelihood is the most demanding and the most relevant to the objective: it is the log probability that the fitted model assigns to the student's *entire* observed ranking, so it is sensitive to the full order *and* to calibration: how confidently the utilities are spaced, not merely their sign. Calibration is what makes per-student utilities aggregate meaningfully when a timetabling objective sums costs across students and schedules, so a model that improves log-likelihood is improving the quantity a scheduler would actually optimize against, whereas one that only improves top-1 has merely guessed more favorites. We therefore read log-likelihood and pairwise accuracy as the primary evidence and top-1/bottom-1 as interpretable but partial summaries.

One latent preference implies many joint orderings. It is worth being explicit about why fitting a per-student utility is more than fitting a set of independent choices: a single latent coefficient induces a *coherent web* of preferences over many schedules at once. Consider two students who agree that bunching is bad but differ in λ (the scheduling-horizon weight). For a spacing-type student ($\lambda_i > 0$), the utility in Equation 5.3 implies, simultaneously and without any further parameters: a later-finishing but well-spread schedule is preferred to an earlier-finishing bunched one; among equally spaced schedules the later finish is tolerated rather than penalized; and a schedule that buys spacing by extending the period is ranked above one that finishes early at the cost of a back-to-back, up to the normalized spacing preference $\rho_{i,k} = \lambda_i / w_{i,k}$ for the relevant bunching type k. A finish-early student ($\lambda_i < 0$) inverts every one of these comparisons. So estimating one number per student is not bookkeeping over isolated answers; it pins down that student's ordering of the entire space of schedules, including the many configurations never shown in the survey. This is precisely what an exam-timetabling objective must

rank, and precisely what the held-out tests of Subsection 5.5.4 check.

5.5 Results

We first observe that students broadly agree on some schedule features, especially that bunching is bad, but disagree sharply on the tradeoff between spacing exams and finishing early. We then test whether those individual differences generalize from simple anchor questions to realistic 4-exam schedules, and find that College of Engineering students are disproportionately represented among students who prefer more spacing even at the cost of a later finish.

5.5.1 Descriptive pattern: Students agree on bunching but are split over spacing versus finishing early

Before fitting, several consensus patterns are visible in the raw rankings:

- In Q2, the same-day 9 am+2 pm option is ranked last by 77% of students, the strongest consensus in the survey.
- The 7 pm→next-day 9 am option is preferred over same-day back-to-back by 81% of students in Q2.
- In Q4 and Q5, the best-spaced option is ranked first by 61% and 76% of students respectively.
- Q3 is the most contested question: 50% of responses fall in its four most common rankings, vs. 88% for Q2.

The contestedness of Q3, the question that most cleanly trades spacing against finishing early, is the first hint that the scheduling-horizon preference is where students most disagree.

5.5.2 RQ1: The main individual heterogeneity is the spacing/finish tradeoff

Fitting the individual model, the σ parameters indicate how much students vary on each latent preference dimension. The raw posterior values suggest that heterogeneity is largest on the scheduling-horizon dimension: $\sigma_\lambda = 5.43$, compared with $\sigma_{w_{\text{oth}}} = 1.30$, $\sigma_{w_{\text{eve}}} = 1.21$, $\sigma_{w_{\text{2in3}}} = 1.03$, $\sigma_\eta = 0.66$, and $\sigma_{u_{\text{2pm}}} = 0.53$. Because coefficient magnitudes depend on feature scaling, we treat this comparison as descriptive rather than decisive. One simple scale check is to multiply each posterior standard deviation by the range of its feature over the actual Q2–Q6 schedule variants. Those observed ranges are 1 for each bunching indicator, 0.63 for last-exam time, 2 for the number of 2 pm exams, and 3 for the number of 7 pm exams. On this observed-feature scale, the implied heterogeneity in utility contribution is still largest for the scheduling-horizon term ($5.43 \times 0.63 \approx 3.44$), compared with 1.30 for same-day back-to-back, 1.21 for evening–morning back-to-back, 1.03 for two-in-three, 1.06 for 2 pm exams, and 1.98 for 7 pm exams. The stronger evidence is therefore both scale-aware and behavioral: Q3, the anchor that isolates spacing against finishing early, is the most contested raw question; the leave-one-question-out sweep shows the largest individual-model advantage on the scheduling-horizon anchor (Appendix Table 5.12); and students split sharply into spacing-preferring and finish-early types (Figure 5.2). Taken together, these patterns indicate that the main disagreement is not over whether bunching is bad or which time of day is preferred, but over how much later students are willing to finish in exchange for more spacing. The per-student bunching penalties are all concentrated at positive values, and the bunching penalties retain positive population means under the per-student parameterization (softplus(μ_0) gives $\bar{w}_{\text{oth}} = 1.85$, $\bar{w}_{\text{eve}} = 1.23$, $\bar{w}_{\text{2in3}} = 1.22$), so allowing heterogeneity does not overturn the consensus ordering of penalties.

5.5.3 Validation: The anchor questions elicit sharper choices than the application block

The fitted block temperatures are consistent with the pattern the calibration and validation design anticipates (Appendix Table 5.9 and Appendix Figure 5.7). The application block is markedly noisier than the anchor blocks: $\tau_{\text{complex}} = 0.66$ (95% CrI $[0.51, 0.82]$) versus $\tau_{\text{bunch}} = 0.13$ $[0.10, 0.17]$, $\tau_{\text{horizon}} = 0.24$ $[0.18, 0.30]$, and $\tau_{\text{time}} = 0.37$ $[0.28, 0.47]$ for the anchors. The credible intervals separate cleanly: the complex block's lower bound (0.51) exceeds every anchor block's upper bound. This is a credible difference in response noise, not a point-estimate artifact. Among the anchors, the bunching question is the sharpest, consistent with the strong consensus against clustering seen in the raw rankings. The intuition is simple: the 4-exam application questions ask students to balance several considerations at once, so their rankings can look less deterministic even when their underlying preferences are stable. In other words, harder tradeoffs introduce more decision noise. We emphasize the interpretation from Subsection 5.4.3: the larger τ_{complex} reflects genuinely noisier *choices* on the harder 4-exam schedules; it is estimated, not assumed, and it is distinct from the uncertainty in the preference parameters themselves, which the Bayesian posterior already propagates into the held-out predictions of Subsection 5.5.4. The hierarchical prior pools the four temperatures toward a common value (μ_τ corresponds to $\tau \approx 0.34$) while letting each block depart from it.

5.5.4 RQ1: Individual preferences generalize to realistic schedules

In-sample, the individual model fits far better than the population model on every metric (Appendix Table 5.10: total NLL 8498 vs. 13283; top-1 75.0% vs. 58.5%; pairwise 82.4% vs. 67.9%). But with $6N$ per-student parameters this comparison rewards overfitting, and indeed BIC, which penalizes parameter count, prefers the population model. The decisive evidence is therefore out-of-sample.

We run two cross-validations, each refitting the model with data held out and scoring the held-out rankings under the individual, covariate, and population predictors defined in Subsection 5.4.1.

Both designs use the same calibration and validation split introduced in Subsection 5.3.3. The model is trained on the anchor questions (Q1–Q3), which emphasize time-of-day preference, bunching, and the spacing-versus-finish tradeoff. It is then evaluated on the application questions (Q4–Q6), the more realistic 4-exam schedules that combine those features. This is the central validation test: do preferences estimated from simple anchor questions jointly predict rankings of more realistic schedules?

The **held-out-question** design asks whether the model can handle a new schedule configuration. It holds out one application question for all students and predicts it from a model refit without that question. Here the question is unseen, but each student's other responses are available. The **held-out-response** design asks a more individual-level question: after seeing a student's anchor responses, can the model predict that same student's realistic 4-exam rankings? It partitions students into five folds and, for each fold, masks those students' three application responses from the training likelihood, then predicts those masked responses. This design better isolates individual heterogeneity because the individual model has access to

the student's simple anchor responses, for estimating their personal parameters, but not to the realistic responses being scored. It is deliberately *not* a leave-student-out design: a fully unseen student would have no individual parameters, so such a test would collapse the individual model back toward the population/covariate predictor and would not test whether learned individual heterogeneity is useful.

Table 5.2 reports both designs. In each, and on every metric, the individual predictor is best: it improves held-out log-likelihood, top-1 accuracy, and bottom-1 accuracy over both the covariate and population predictors. The two designs agree closely (individual $\approx 62.5\%$ top-1 in both, about two points above the baselines). The per-question breakdown behind these means, together with the anchor-question holdouts, is given in Appendix Table 5.12.

Formal test. To test the gain rather than describe it, we compare held-out log-scores student by student. Let ℓ_i^{ind} be the average held-out log-score for student i under the individual model, and let ℓ_i^{other} be the corresponding score under either the covariate or population model. The null hypothesis is that the individual model does not improve the average held-out score: $H_0 : \mathbb{E}[\ell_i^{\text{ind}} - \ell_i^{\text{other}}] \leq 0$. This is a Vuong-style predictive comparison, which avoids the boundary problem that invalidates a naive likelihood-ratio test of $\sigma_\lambda = 0$ (Subsection 5.4.1). Because a student contributes several held-out 4-exam observations, treating the 2943 observations as independent would overstate significance; we therefore *cluster by student*, averaging each student's per-observation log-score difference into one value and running the paired t- and Wilcoxon signed-rank tests on the resulting 981 independent student-level differences. Appendix Table 5.11 reports all four contrasts. Across both designs and against both baselines the individual predictor wins comfortably (every one-sided $p < 10^{-10}$ under both tests). The effect *size*, however, is modest: a mean student-level advantage of $+0.13$ to $+0.25$ nats, about

Design	Predictor	LL/obs	Top-1	Bottom-1	Pairwise
Held-out question	Individual	**−2.847***	**62.2**%	**43.4**%*	**70.1**%*
		[−2.90, −2.79]	[60.5, 64.0]	[41.6, 45.1]	[69.2, 70.9]
	Covariate	−3.092	60.3%	36.7%	67.6%
		[−3.14, −3.04]	[58.6, 62.0]	[35.1, 38.3]	[66.7, 68.5]
	Population	−2.975	60.3%	38.6%	67.9%
		[−3.01, −2.94]	[58.6, 62.0]	[37.0, 40.1]	[67.0, 68.8]
	Random	−3.178	25.0%	25.0%	50.0%
Held-out response	Individual	**−2.693***	**62.6**%	44.9%	70.3%
		[−2.74, −2.64]	[60.9, 64.3]	[43.2, 46.7]	[69.4, 71.1]
	Covariate	−2.853	60.3%	36.7%	67.7%
		[−2.90, −2.81]	[58.6, 62.0]	[35.1, 38.3]	[66.8, 68.6]
	Population	−2.832	60.3%	38.6%	67.9%
		[−2.87, −2.80]	[58.5, 62.0]	[37.0, 40.0]	[66.9, 68.8]
	Modal	—	60.3%	**51.6**%*	**70.7**%
			[58.6, 61.9]	[49.8, 53.4]	[69.7, 71.6]
	Random	−3.178	25.0%	25.0%	50.0%

Table 5.2: Calibration-to-validation performance: out-of-sample performance of the three nested predictors on the held-out 4-exam application questions (Q4–Q6), with preferences estimated from the anchor questions (Q1–Q3), over all $n = 2943$ held-out observations. LL/obs is mean held-out log-likelihood per observation; accuracies are percentages (held-out-question values are means over the three held-out questions; pairwise accuracy is the fraction of the six option-pairs per ranking ordered correctly, chance 50%). Grey brackets are 95% cluster-bootstrap confidence intervals (resampling students, 2000 replicates); the random baseline is analytic. *Modal* is a zero-modeling baseline that predicts the single most-frequent ranking of each question (mode taken on the training-fold students, scored on the held-out fold; it has no LL, and no held-out-question version since an unseen question has no training mode). Within each design and metric, **bold** marks the best predictor (the modal baseline competes on accuracies but not LL; random is a reference floor) and a star (*) marks a 95% CI lying entirely above all competitors'. The individual predictor is best on top-1 and LL in both designs, so the preferences estimated from the anchor questions carry out-of-sample signal for realistic schedules; the covariate predictor does no better than the population mean. The modal baseline is strong: it matches the utility models on top-1 and credibly exceeds them on bottom-1 (and leads on pairwise in the held-out-response design). This locates the models' advantage in the log-score and top choice rather than in every metric.

two accuracy points. This is clear evidence that the heterogeneity generalizes, not evidence that the per-student gain is large.

Which questions can be held out, and why mostly the 4-exam block. The main test holds out the application block because those questions provide the intended calibration-to-validation test. It is still useful to ask what happens when each *other* question is held out in turn. Appendix Table 5.12 reports a leave-one-question-out sweep over the five schedule questions (Q2–Q6) plus the time question (Q1). The pattern matches the parameter mapping in Subsection 5.4.5. The time question (Q1) is the extreme case: the evening and afternoon preference parameters (η and $u_{2\text{pm}}$) are learned almost entirely from Q1, so with Q1 removed the population predictor is essentially at chance on its first choice (22.6%, below the 33% baseline). Holding it out is therefore not a clean generalization test; it mainly shows that time-of-day preferences require the time-of-day question. The bunching anchor (Q2) is the opposite: the bunching penalties also appear in every 4-exam schedule, so they are well estimated even without Q2, the population predictor already predicts it well (86% top-1), and the per-student deviations add only noise (individual is slightly *worse*). The scheduling-horizon anchor (Q3) is where heterogeneity matters most: holding it out, the individual predictor still recovers each student's scheduling-horizon preference (λ) well enough to nearly double the population predictor's top-1 (47.9% vs. 25.5%). Across the application questions (Q4–Q6) the individual predictor wins consistently, which is the primary result. The overall message is that individual heterogeneity helps most exactly where students genuinely disagree (the scheduling horizon and the complex application schedules), and little on dimensions that are either near-universal (bunching) or estimated by a single dedicated question (time-of-day).

The covariates, by contrast, do not generalize. A notable corollary of Table 5.2 is that the covariate predictor is no better than, and on bottom-1 slightly worse than, the bare population predictor out-of-sample, even though COE status is clearly detectable in-sample (Subsection 5.5.5). The generalizable signal is the individual deviation $\mathbf{z}_i$, not the demographic means. The heterogeneity advantage is also largest on bottom-1 accuracy (an $\approx$ 8-point individual-over-covariate gain in both designs), so the per-student weights are especially informative about which schedule a student will *reject*.

5.5.5 RQ2: College of Engineering students are more often spacing-preferring

The second main finding concerns where the one large group difference lies. We test it on the individual model through the posterior of the covariate coefficient $B_{\text{is_COE},\lambda}$, the average COE–non-COE difference in scheduling-horizon preference with all per-student uncertainty integrated out. (This is the correct population test: a two-sample test on the noisy posterior-mean $\hat{\lambda}_i$ would treat each estimate as error-free.) The effect is +1.35 (95% CrI $[+0.59, +2.18]$, $\Pr(>0) = 1.000$; Appendix Table 5.13 and Figure 5.1): COE students prefer a later last exam, meaning more time on campus, in exchange for more spread-out exams. Figure 5.1 shows the shift directly, plotting every student on the two spacing-related axes at once: COE students lie further toward a later finish (raw $\hat{\lambda}_i$ group means +1.88 vs. +0.42) and, as the right-hand marginal shows, also penalize bunching somewhat more (Appendix Table 5.14 and Appendix Figure 5.6). Both groups span the full range and overlap heavily, so within-group heterogeneity still dominates the between-group shift.

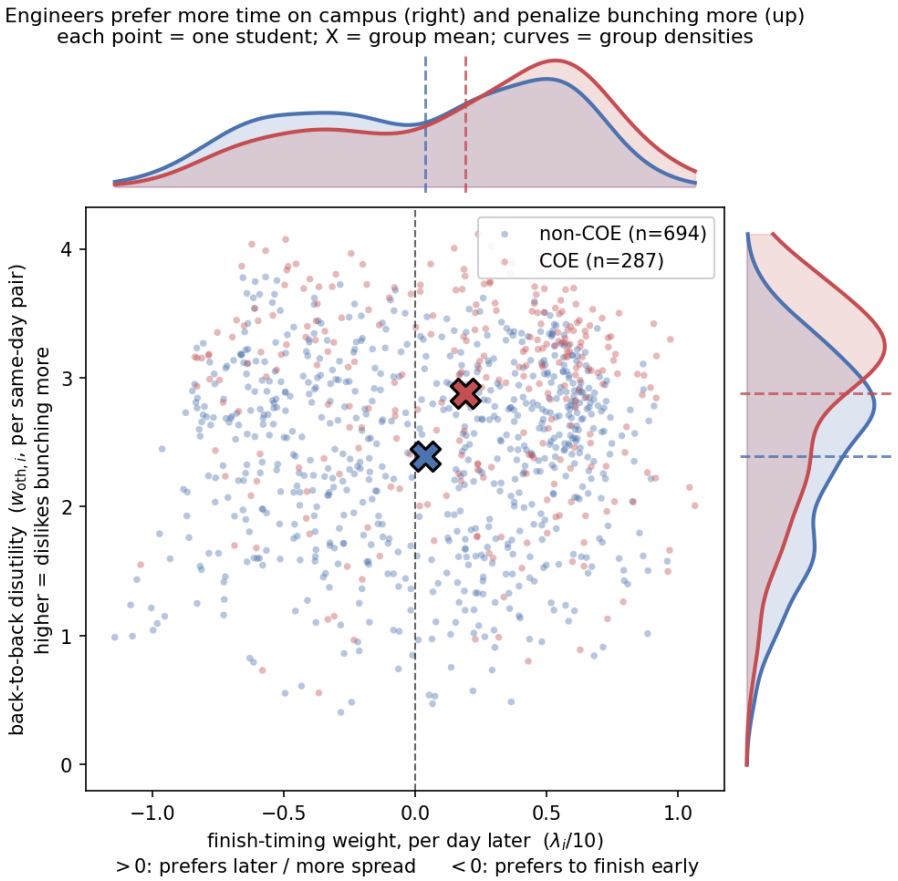


Figure 5.1: The College of Engineering difference on both spacing axes at once. Each point represents one student, placed by posterior-mean finish-timing weight ($\lambda_i/10$, utility per day of later finish; > 0 prefers a later, more spread-out schedule) against same-day back-to-back disutility ($w_{\text{oth},i}$, higher = more bunching-averse); the large "X" marks each group's mean and the marginal curves are the group densities on each axis. COE students are shifted *right* (they weight a later finish / more spacing more, group means 0.19 vs. 0.04 per day) *and up* (they penalize bunching more, 3.13 vs. 2.60). In the covariate model, the COE effect is +1.35 on λ and +0.50 to +0.58 on the three bunching penalties. Both groups overlap heavily, so the difference is a shift in the mix of student types, not a separation. Per-student points carry posterior uncertainty and should be read for the group pattern, not exact positions.

Willingness to finish later for more spacing. Because COE students also dislike bunching more (Appendix Table 5.14 and Appendix Figure 5.6), a natural concern is that their stronger scheduling-horizon preference merely reflects larger coefficients overall rather than a genuinely different tradeoff. To rule this out we ask a question that has a common scale removed by construction: *for the same level of bunching aversion, how much later is each group willing to finish in exchange for more spacing?* Because Plackett–Luce utilities are interpreted only relative to the question-level temperature scale, raw coefficient magnitudes are less meaningful than coefficient *ratios*; normalizing the later-finish/more-spacing weight by a group's own bunching penalty gives exactly this willingness, expressed in units of that group's own bunching aversion. We form each group's normalized spacing preference

$$\rho_{g,k} = \frac{\lambda_g}{w_{g,k}}, \qquad g \in \{\text{COE}, \text{nonCOE}\}, \quad k \in \{\text{oth}, \text{eve}, \text{2in3}\}, \tag{5.5}$$

with numerator and denominator estimated *within* the same group, and compare groups on the log scale via a difference-in-differences,

$$\begin{aligned} \Delta_{\log,k} &= \log \rho_{\text{COE},k} - \log \rho_{\text{nonCOE},k} \\ &= \big(\log \lambda_{\text{COE}} - \log w_{\text{COE},k}\big) - \big(\log \lambda_{\text{nonCOE}} - \log w_{\text{nonCOE},k}\big), \end{aligned} \tag{5.6}$$

computed at the group level within each posterior draw (the group-mean λ is positive in every draw, so the logs are well defined even though individual λ_i need not be). Table 5.3 reports $\rho_{g,k}$, $\Delta_{\log,k}$ with its 95% CrI and $\Pr(\Delta_{\log,k} > 0)$, and the multiplicative ratio $\exp(\Delta_{\log,k})$ for all three bunching denominators; Appendix Figure 5.8 shows the per-student normalized weights and the posterior of $\Delta_{\log,\text{oth}}$. The result is the same for every denominator: $\Delta_{\log,k}$ is positive with $\Pr(> 0) = 1.000$, and COE students' normalized spacing preference is roughly 3.9 to 4.2 times the non-COE value. In plain terms: *after accounting for how much each group dislikes compressed exams, COE students are still substantially more willing than non-COE*

students to accept a later last exam in order to obtain a more spread-out schedule; equivalently, non-COE students place relatively more weight on finishing earlier once bunching aversion is held fixed. The difference is therefore not an artifact of COE simply having larger coefficients overall. Normalizing by each group's own bunching aversion removes any common scale, yet the gap remains. Because the survey schedules couple last-exam timing with spacing (a later finish creates more room between exams), this should be read as a stronger preference on the combined scheduling-horizon axis, not as a fully separable estimate of a pure finish-date effect.

Bunching denominator k	$\rho_{\text{nonCOE},k}$	$\rho_{\text{COE},k}$	$\Delta_{\log,k}$	95% CrI	$\Pr(>0)$	$e^{\Delta_{\log,k}}$
same-day (w_{oth})	0.16	0.66	+1.44	$[+0.90, +2.23]$	1.000	4.2×
eve–morning (w_{eve})	0.27	1.02	+1.36	$[+0.82, +2.13]$	1.000	3.9×
two-in-three (w_{2in3})	0.28	1.06	+1.39	$[+0.84, +2.17]$	1.000	4.0×

Table 5.3: Normalized spacing preference (willingness to finish later per unit of bunching aversion) by COE status, one row per bunching denominator. $\rho_{g,k} = \lambda_g / w_{g,k}$ is group g's later-finish/more-spacing weight normalized by bunching type k, so it carries no overall utility scale; $\Delta_{\log,k} = \log \rho_{\text{COE},k} - \log \rho_{\text{nonCOE},k}$ is the log difference-in-differences (group-level, per posterior draw), and $e^{\Delta_{\log,k}}$ the multiplicative ratio of the two groups' values. The COE normalized spacing preference is larger for every bunching denominator ($\Pr > 0 = 1.000$), so the effect is not an artifact of utility scale.

Practical translation. The normalized spacing preference ρ asks how much value a group places on moving toward a later, more spread-out exam period, measured in units of its own bunching aversion. Using same-day back-to-back as the unit, the COE value is 0.66 same-day-back-to-back equivalents, compared with 0.16 for non-COE students. Thus, for the same amount of same-day back-to-back aversion, the later-finish/more-spacing axis carries about four times as much weight for COE students. On a rough per-day scale over the 10-day exam period, this is about 0.07 same-day-back-to-back equivalents per later day for COE students and about 0.02 for non-COE students. Because later finish and greater spacing are coupled in the survey schedules, this should be read as evidence that COE students place more value on spacing, not as a pure estimate of how many extra days they would accept in isolation.

The group effect is a shift in the mix of individual types. The clearest way to see this is to classify each student by the sign of their scheduling-horizon preference: a *spacing type* ($\lambda_i > 0$) is willing to finish later for more spacing, and a *finish-early type* ($\lambda_i < 0$) prefers an earlier last exam even at the cost of worse spacing. (We use the posterior probability $\Pr(\lambda_i > 0)$ as a soft version of this label. This avoids leaning on a noisy per-student ratio; when $\lambda_i \leq 0$, the ratio no longer means willingness to finish later, and its log is not defined.) Two things stand out (Figure 5.2). First, students are mostly *confident* types: the distribution of $\Pr(\lambda_i > 0)$ is sharply U-shaped, piling up near 0 and 1 rather than in the middle. Thus the heterogeneity in λ is a split into two fairly distinct groups, not a smear of mild differences. Second, COE students are more often spacing types: 56% are confidently spacing-preferring ($\Pr > 0.9$) versus 43% of non-COE students, and only 25% are confidently finish-early ($\Pr < 0.1$) versus 34% (group-mean $\Pr(\lambda_i > 0)$ 0.66

vs. 0.55). The group difference is therefore a shift in the *mix* of preference types: COE has proportionally more spacing types, rather than a uniform movement of every student, and both groups contain sizable fractions of both types.

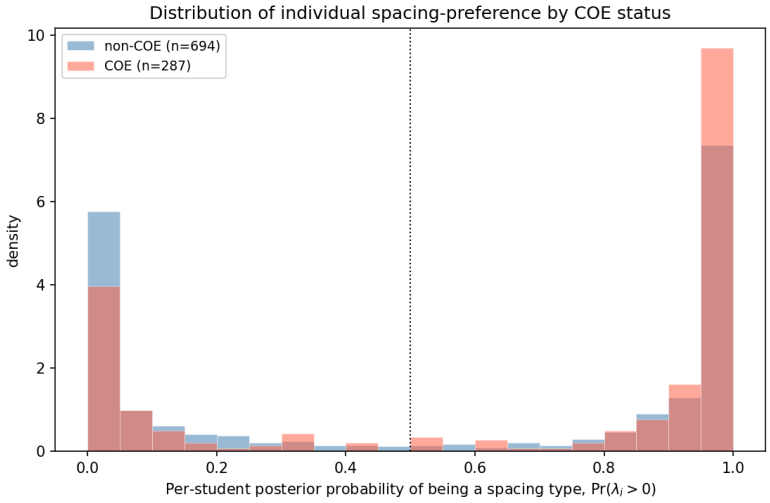


Figure 5.2: Per-student posterior probability of being a spacing type, $\Pr(\lambda_i > 0)$, by COE status (density-normalized). The U-shape indicates that most students are confidently one type or the other; COE students place more mass near 1 (spacing-preferring) and less near 0 (finish-early) than non-COE students.

COE bunching effects. COE students also penalize bunching *more* than non-COE students, on every type: the college effects are $B_{\text{is_COE},w_{\text{oth}}} = +0.55$ (95% CrI $[+0.27, +0.86]$), $B_{\text{is_COE},w_{\text{eve}}} = +0.58$ ($[+0.31, +0.91]$), and $B_{\text{is_COE},w_{\text{2in3}}} = +0.50$ ($[+0.25, +0.80]$), all with $\Pr(> 0) = 1.000$ (raw per-student same-day penalties 2.88 vs. 2.39, etc.). So the college difference is not that COE tolerate bunching while caring about finish date. They are *more* bunching-averse on all axes *and* weight a later, more spread-out finish more. All three effects are credibly positive (Appendix Table 5.14; Appendix Figure 5.6). Both tendencies point the same way: COE students care more about exam spacing overall. The relative *shape* of bunching aversion is broadly similar across groups: normalizing each group's penalties to its own same-day back-to-back gives $1 : 0.65 : 0.62$ for COE and $1 : 0.60 : 0.59$

for non-COE, so both rank the same-day back-to-back as worst and the other two bunching penalties at roughly 60%. The profiles are not identical: COE weight the evening→morning penalty slightly higher relative to same-day (normalized 0.65 vs. 0.60, posterior difference +0.05, 95% CrI $[+0.02, +0.08]$), while the two-in-three ratio differs only marginally (+0.03, $[+0.00, +0.06]$). The single largest college effect, however, is on the scheduling horizon: $B_{\text{is_COE},\lambda} = +1.35$, compared with +0.50 to +0.58 for the three bunching penalties.

The college difference is, by contrast, *not* present on time-of-day preferences. The evening effect $B_{\text{is_COE},\eta} = +0.10$ (95% CrI $[+0.00, +0.21]$) and the afternoon effect $B_{\text{is_COE},u_{2\text{pm}}} = +0.12$ ($[+0.03, +0.22]$) are an order of magnitude smaller than the λ effect and negligible relative to within-group spread, and the corresponding $\hat{\eta}_i$ and $\hat{u}_{2\text{pm},i}$ distributions nearly coincide across colleges. Summarizing the two time-of-day parameters as the model-implied probability that each slot is ranked first (Appendix Figure 5.9), afternoon is the most-preferred slot for both groups ($\approx$ 43–45%), and the COE and non-COE profiles differ by at most about two percentage points in any slot. Class-year effects are likewise minor, with the exception of a negative year-2028 effect on λ that mirrors a descriptive class-year pattern (Appendix Table 5.14).

Taken together, the evidence points to one conclusion. COE students care more about exam spacing on every axis we measure: they penalize all three bunching types more strongly, they weight a later/more-spread finish more (the largest single effect), and after normalizing the scheduling-horizon weight by each bunching penalty the exchange rate is still larger for COE ($\rho_{\text{COE},k} > \rho_{\text{nonCOE},k}$ for every k), so the difference is not merely one of overall coefficient scale. The groups largely agree on the relative shape of bunching aversion (small differences aside) and on time-of-day preferences. In short, COE and non-COE students differ mainly in how

much they value spacing and an earlier-versus-later finish, far more than in the relative ranking of bunching types or in what times of day are preferred.

Supporting checks. Additional comparisons support the same interpretation without changing the main story. COE and non-COE students have nearly the same *relative* bunching profile (same-day back-to-back has the largest posterior mean for both groups, with evening–morning and two-in-three posterior means at roughly 60% of that value), and they are nearly indistinguishable on time-of-day preferences. COE students are also somewhat more predictable out-of-sample (-2.73 versus -2.89 held-out log-likelihood per observation), but a model with COE-specific choice temperatures indicates that this is mostly preference concentration rather than lower response noise. These checks are consistent with the central conclusion: the meaningful group difference is not a broad difference in how students use the survey scale, but a difference in the mix of students on the spacing-versus-finishing-early axis.

5.5.6 Scale and identification checks

The COE effect is stated on the scheduling-horizon coefficient λ, so it is worth confirming it is not an artifact of the model's scale invariances. A Plackett–Luce utility is unchanged when all weights are multiplied by a constant and the temperatures divided by the same constant, which creates three entangled scales: (i) an overall scale of all weights against the temperatures; (ii) the relative scale of λ against the bunching penalties; and (iii) the choice noise τ, which can absorb scale block by block. The first two scale invariances are already neutralized by how we report the COE effect: the exchange-rate difference-in-differences in Equation 5.6 is a ratio of ratios, invariant to the overall scale and with the arbitrary λ-vs-bunching

baseline cancelling across groups.

The third, a confound between λ and choice noise, is the most relevant, because λ is identified largely from the horizon anchor (Q3) and a weight trades off directly against its question's temperature. A COE difference in λ could in principle be a COE difference in τ_{horizon} in disguise. We test this directly by refitting with a COE-specific multiplicative shift on *every* block temperature simultaneously, $\tau_{b,i} = \tau_b \exp(\delta_b \,\text{is_COE}_i)$ for $b \in \{\text{time}, \text{bunch}, \text{horizon}, \text{complex}\}$, jointly with the full covariate effects. This gives the noise on every block, especially the horizon block, every opportunity to absorb the COE signal. It does not: the COE scheduling-horizon effect is essentially unchanged, $B_{\text{is_COE},\lambda} = +1.26$ (95% CrI $[+0.54, +1.97]$, $\Pr(>0) = 1.000$, versus $+1.35$ in the main model), and the horizon-block noise shift is itself negligible ($\delta_{\text{horizon}} = -0.09$, CrI $[-0.35, +0.16]$). The per-block noise shifts show no uniform "COE answer more sharply" pattern: COE are if anything noisier on the time question ($\delta_{\text{time}} = +0.23$, CrI $[+0.01, +0.47]$) and only weakly sharper elsewhere. Thus the COE difference lives in the preference λ, not in choice noise. The exchange-rate difference-in-differences itself is likewise robust under this model: recomputed from the all-block-temperature posterior, the log ratio-of-ratios for λ/w_{oth} is $+1.47$ (95% CrI $[+0.90, +2.33]$, $\Pr(>0) = 1.000$; a $4.4\times$ multiplicative difference), essentially the $4.2\times$ of the main model. The scheduling-horizon tradeoff finding is therefore robust to all three scale invariances: the overall scale, the λ-vs-bunching baseline, and choice noise.

5.5.7 RQ3: Population-level weights for timetabling

Although the individual model is the object of study, its population rung yields the practical deliverable for schedulers: the shared bunching penalties. Reading softplus($\boldsymbol{\mu}_0$) and normalizing the posterior mean penalties to the same-day back-

to-back penalty gives relative weights

$$(w_{\text{oth}} : w_{\text{eve}} : w_{\text{2in3}}) \approx (1 : 0.67 : 0.66).$$

Same-day back-to-back has the largest posterior mean penalty, while evening–morning back-to-back and two-in-three each have posterior means roughly two-thirds as large. The exact ratios are imprecisely estimated, however (Table 5.4). Although the respondent sample is large, the schedule comparisons provide limited independent information for distinguishing the three bunching penalties from one another. The normalized posterior means can be used as objective weights in standard exam-timetabling formulations (Muklason et al., 2017; Bullnheimer, 1998); because the sample is from one institution the magnitudes may not transfer, and even the qualitative ordering remains uncertain.

Table 5.5 places these learned weights beside the corresponding quantities in the two closest operational scheduling papers. Cornell's two-in-24 penalty describes the same two-slot-gap event that we call two-in-three; Bucknell does not ask that item directly. Our model does not estimate a three-exams-in-24-hours weight because none of the survey schedules contains that event; we include the row only to compare the Cornell operational weight with Bucknell's survey rating. Current optimization systems already require relative inconvenience weights; our survey estimates one set of those weights from ranked complete schedules, while the Cornell and Bucknell systems use planner-chosen or survey-informed values.

As a robustness check on the Bucknell comparison, coding their response options as Fine = 0, Inconvenient = 1, and Extremely inconvenient = 2 gives severity scores that are similar for same-day back-to-back (1.60), night-to-morning (1.54), three exams in 24 hours (1.72), and four exams in 48 hours (1.63). Thus Bucknell's descriptive survey supports the broad conclusion that bunching events are among the largest schedule inconveniences. It does not, however, provide a direct two-

Penalty	Normalized posterior mean	Median ratio	95% CrI
Same-day back-to-back (w_{oth})	1.00	1.00	reference
Evening–morning back-to-back (w_{eve})	0.67	0.64	$[0.18, 2.12]$
Two-in-three (w_{2in3})	0.66	0.63	$[0.17, 2.42]$

Table 5.4: Relative population-level bunching penalties from the individual model, normalized to same-day back-to-back. The second column divides each raw posterior mean by the same-day posterior mean. The final two columns summarize the ratio computed separately in each posterior draw; medians are reported because the ratio posteriors are right-skewed. Raw coefficient summaries appear in Appendix Table 5.8.

Schedule inconvenience	Our learned weight	Cornell optimization	Bucknell survey
Same-day back-to-back	1.00	1.00	64.6% extremely inconvenient
Evening–morning back-to-back	0.67	1.00	58.9% extremely inconvenient
Two-in-three	0.66	0.50	N/A (not asked directly)
Three exams in 24 hours	N/A (not a fitted feature)	10.00	75.2% extremely inconvenient

Table 5.5: Comparison of learned bunching weights with operational scheduling weights and survey evidence in related work. Our entries are posterior mean penalties normalized to same-day back-to-back = 1; Cornell weights are also shown relative to its generic back-to-back penalty (Ye et al., 2024). Cornell's weights are operational defaults chosen with Registrar constraints and institutional priorities in mind, not student-only preference estimates. Cornell does not separate same-day from evening–morning back-to-backs in the default ratios; its 0.50 two-in-24 row uses the same two-slot-gap definition as our two-in-three feature. Bucknell reports descriptive ratings of isolated inconveniences rather than utility weights; its 24-hour survey item is three exams in 24 hours, not two exams in 24 hours (Chaplin et al., 2026). The three-exams-in-24-hours row is a direct Cornell–Bucknell comparison, but it is listed as N/A for our model because our fitted feature set does not include a three-exams-in-24-hours term. "N/A" means the feature is not separately estimated, reported, or asked in that source.

in-three comparison; our ranking design adds a utility-scale ordering among the Cornell bunching features.

5.6 Discussion and Implications

Read against current practice, a fixed, planner-chosen, population-level objective, our results expose two shortfalls.

5.6.1 Implications for exam-scheduling policy

A population objective discards real student-level structure. Individual preference heterogeneity is not a modeling artifact: it improves prediction of held-out rankings in two independent cross-validation designs and by formal test (student-clustered, all $p < 10^{-10}$; the gain is about two accuracy points). A single common weight vector, however carefully chosen, cannot capture this, so it optimizes against preferences that systematically differ from those of many students. The heterogeneity is concentrated in one dimension, the scheduling-horizon tradeoff between spacing and finishing early, on which students genuinely disagree, even as they agree that bunching is bad and that afternoons are mildly preferred. This conclusion does not rest only on raw coefficient magnitudes: it is visible in the contested raw responses to Q3, in the range-adjusted coefficient comparison, in the leave-one-question-out results, and in the split between spacing-preferring and finish-early types. Students vary most in how strongly they trade off a more spread-out exam period against finishing and leaving earlier, not in the relative ranking of bunching types or in time-of-day taste.

We should be candid, however, about the *practical* size of this gain. The heterogeneity is statistically clear and improves the held-out log-score reliably,

but in accuracy terms it buys only about two points of top-1 over the population predictor, and a zero-modeling modal baseline that simply predicts each question's most common ranking is competitive on the accuracy metrics (it matches the utility models on top-1 and beats them on bottom-1 in the held-out-response design; Table 5.2). The value the individual model adds is therefore best understood as a better-calibrated *ordering and log-score*, and as evidence of the heterogeneity structure it reveals, rather than a large jump in how often the single favorite schedule is identified. We read this as evidence that per-student preferences are real and generalize, not yet as a case that individualized scheduling interventions are warranted on predictive grounds alone; realizing that would likely require richer per-student features (e.g. actual exam load) than the six schedule features this survey measures.

Toward eliciting, not assuming, the weights. Current practice fixes a single population weight vector and optimizes against it. Our results suggest a modest alternative: because the heterogeneity that matters is concentrated on two interpretable axes, how strongly a student dislikes bunching and how willing they are to stay later to reduce it, a short, low-cognitive-load survey (a handful of paired-schedule choices like our anchor questions) could place each student approximately on these axes. Even coarse per-student estimates would let the registrar search for university-wide schedules that better balance these needs, or at least make the tradeoffs a fixed schedule imposes *explicit*: a timetable that reduces bunching by extending the period better matches students with stronger spacing preferences, while moving away from the preferences of students who would rather finish and go home, and vice versa. Surfacing the *distribution* of per-student costs under a candidate schedule, which the individual model produces directly, turns these implicit, currently invisible tradeoffs into a quantity the planner can see and weigh.

5.6.2 Implications for engineering education

Group differences are real but specific, and a uniform objective hides them. College of Engineering students are more likely to be spacing-preferring: they place more weight on a later, more spread-out finish ($B_{\text{is_COE},\lambda} = +1.35$, $\Pr(> 0) = 1.000$), and this difference remains after normalizing by each group's own bunching aversion. The point is not that every engineering student differs from every non-engineering student; the two groups overlap heavily. Rather, COE has a different mix of student types, with a larger share of students on the spacing-preferring side of the tradeoff. We emphasize that this is a difference in measured ranking behavior, not in any explanation for it: the survey does not record per-student exam load or course structure, so COE status should be read as a proxy for unmeasured curricular and contextual differences that travel with being an engineering student. The practical implication is not to replace one population objective with several group-specific objectives. Because within-group variation dominates the group mean and the COE/year covariates do not improve out-of-sample prediction by themselves, the fairer object of attention is the individual-level distribution.

A related descriptive pattern. As an aside, COE students responded at roughly 1.7× the non-COE rate (Table 5.1), passed the attention checks slightly more often (99.9% vs. 98.5%; Appendix Figure 5.5), and were marginally more predictable out-of-sample (-2.73 vs. -2.89 held-out log-likelihood). These measures do not establish greater engagement or explain why the patterns coincide. We flag them only as a pattern a purpose-built study measuring effort, stress, and exam load directly could investigate.

5.6.3 Limitations and future work

The survey is cross-sectional at a single institution; five questions surface only 20 unique schedule configurations, so triples, quads, quints, and hard conflicts are not identifiable; the per-student exam load is not recorded; responses are stated rather than revealed preferences; and the out-of-sample tests, while significant, rest on observations that are not fully independent across a student's questions. Relatedly, COE status is a coarse grouping variable: it stands in for a bundle of unmeasured curricular and contextual differences (course load, problem-set intensity, cohort norms) that we cannot separate here, so the engineering effect is established as a real difference but its cause is not. Finally, our group analysis contrasts only College of Engineering students with everyone else; *if* the difference is driven by heavier, more problem-set-intensive course loads, one plausible but untested account, then other quantitative majors with comparable structure (e.g., mathematics and the hard sciences), currently pooled into our non-COE group, might show similar patterns, and a finer-grained study across majors that also records course load could test both the grouping and the mechanism.

5.7 Conclusion

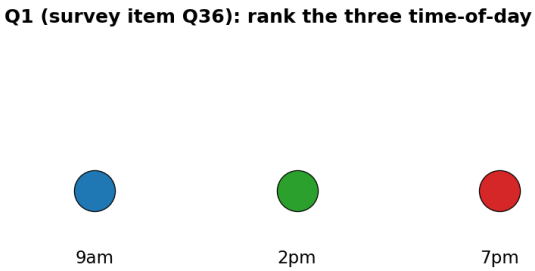
An exam schedule distributes study time, time pressure, and recovery opportunities across the student body, yet Cornell's scheduling optimization, like much of the timetabling literature, decides that distribution with a single planner-chosen weight vector applied to all students. Using a survey of $N = 981$ Cornell students and a nested family of hierarchical Plackett–Luce models, we gave two pieces of evidence that this uniform objective misrepresents whom the schedule is for: individual-level heterogeneity genuinely generalizes out-of-sample (so a population objective

discards real structure), and that heterogeneity is concentrated on a single tradeoff, staying longer for more spacing versus finishing and leaving, on which students sharply disagree and for which College of Engineering students are more likely to prefer greater spacing despite a later finish. Because the disagreement is real, no single schedule is neutral: it necessarily eases compression for one type of student at the cost of keeping the other on campus longer. The contribution of a preference model here is not primarily a better set of weights. The population rung does supply normalized posterior mean feature weights ($1 : 0.67 : 0.66$) estimated from student data, although the exact ratios remain substantially uncertain. More importantly, measuring preferences turns an implicit, invisible allocation of exam burden into one a planner can see, weigh, and defend. Better exam schedules, on this view, will come less from re-tuning weights than from treating scheduling as the assessment-policy decision it already is.

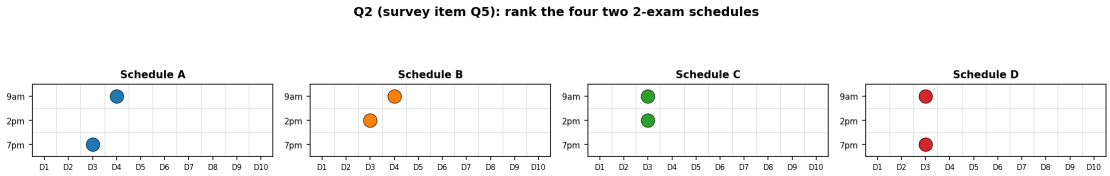
Appendices to Chapter 5

5.8 Survey ranking questions

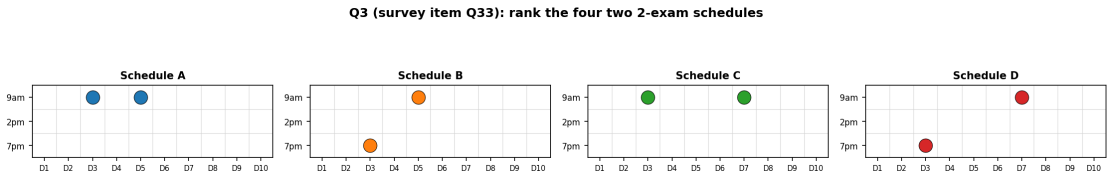
This appendix reproduces the six ranking questions in the order presented to respondents. Q1 ranks the three time-of-day slots; Q2–Q6 ask the student to rank four schedule variants (A–D), shown on the 10-day × 3-slot grid (rows 9 am / 2 pm / 7 pm, columns days D1–D10). Survey-item codes are given in each caption for traceability to the raw data.



(a) Q1 (survey item Q36): rank the three time-of-day slots.

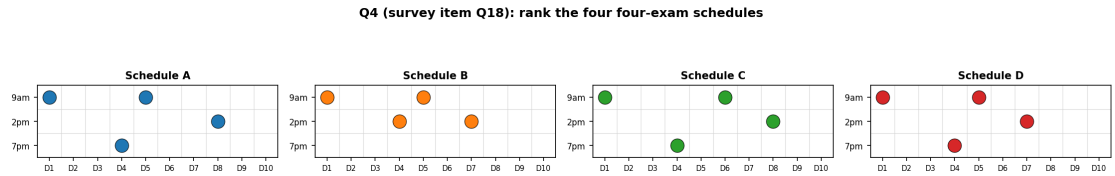


(b) Q2 (survey item Q5): rank four 2-exam schedules.

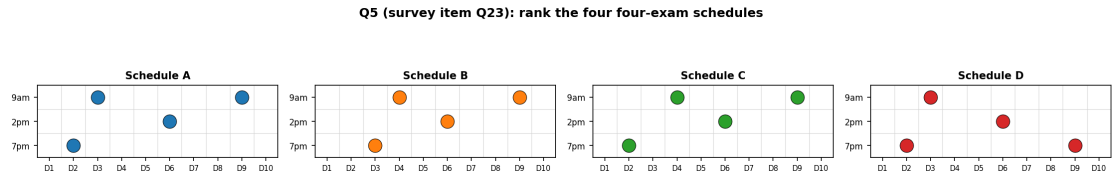


(c) Q3 (survey item Q33): rank four 2-exam schedules.

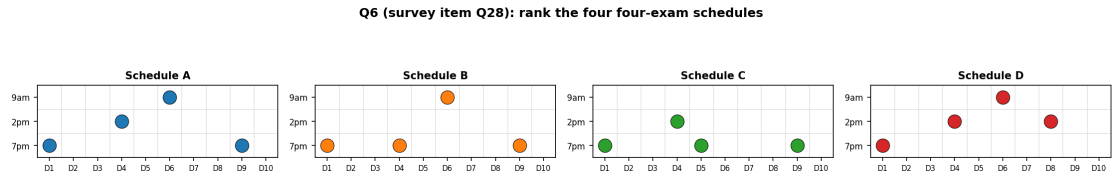
Figure 5.3: The six survey ranking questions in presentation order. Q1 ranks the three time-of-day slots; Q2–Q3 rank four schedule variants (A–D) on the 10-day × 3-slot grid (rows 9 am / 2 pm / 7 pm, columns D1–D10). Q4–Q6 continue on the following page. Survey-item codes are given per panel for traceability to the raw data.



(d) Q4 (survey item Q18): rank four 4-exam schedules.



(e) Q5 (survey item Q23): rank four 4-exam schedules.



(f) Q6 (survey item Q28): rank four 4-exam schedules.

Figure 5.3: Six survey ranking questions (continued). Q4–Q6 rank four 4-exam schedule variants on the same 10-day × 3-slot grid.

5.9 Survey sample and response quality

This appendix collects the sample-composition and response-quality material referenced from Subsection 5.3.4.

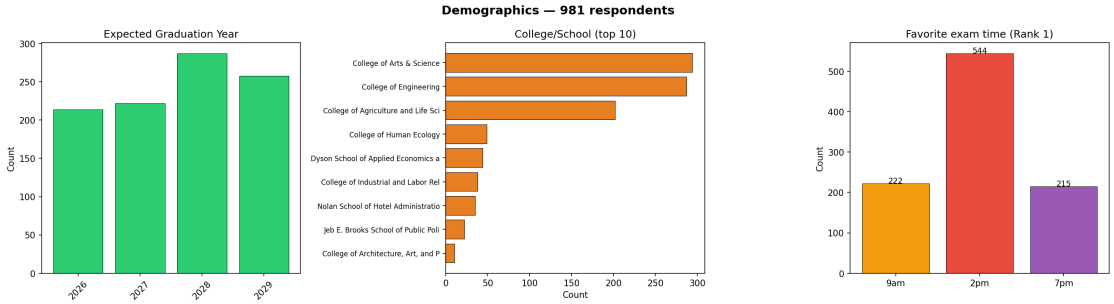


Figure 5.4: Sample composition: expected graduation year (the single 2030 respondent is grouped with 2029, as in Table 5.6), and college/school (top 10).

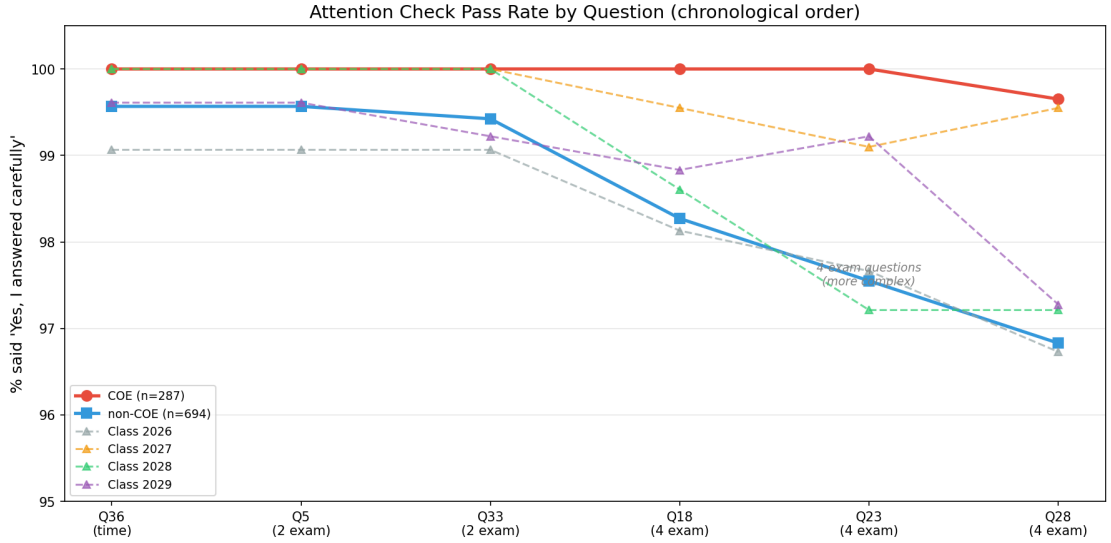


Figure 5.5: Self-reported careful-answering (attention-check) pass rate after each ranking question, in the order presented, split by COE status and by class year. Pass rates stay above 96.5% throughout and decline only mildly on the later, more complex 4-exam questions. COE students answer at least as carefully as non-COE students at every question (99.9% mean vs. 98.5%), so the COE results in Subsection 5.5.5 are not an artifact of differential response quality.

College / School	2026	2027	2028	2029	Total
Arts & Sciences	48	65	86	95	294
Engineering (COE)	63	64	87	73	287
Agriculture & Life Sciences	56	45	55	46	202
Human Ecology	12	18	8	11	49
Dyson School	11	6	15	12	44
Industrial & Labor Relations	11	9	11	7	38
Hotel	5	8	17	5	35
Brooks School of Public Policy	6	3	6	7	22
Architecture, Art & Planning	2	4	2	2	10
Total	214	222	287	258	981

Table 5.6: Joint distribution of college and expected graduation year for the $N = 981$ analysis sample. Columns are graduation years (one respondent reporting 2030 is grouped with 2029). The three largest colleges account for 79.9% of responses; class-year distributions do not differ significantly between COE and non-COE students (χ^2 test, $p > 0.3$).

Engineering major	n	% of COE
Mechanical Engineering	65	22.6
Computer Science	56	19.5
Chemical & Biomolecular Engineering	34	11.8
Operations Research & Information Engr.	26	9.1
Biomedical Engineering	26	9.1
Electrical & Computer Engineering	26	9.1
Civil Engineering	12	4.2
Applied & Engineering Physics	10	3.5
Materials Science & Engineering	9	3.1
Smaller majors (< 8 each)	17	5.9

Table 5.7: Engineering-major membership within the College of Engineering subsample ($n = 287$). Majors are self-reported and multi-select; a respondent enrolled in two majors is counted under each, and 25 COE respondents left the major question blank, so counts need not sum to 287 and percentages are taken over the 287 COE respondents. "Smaller majors" collects all engineering majors with fewer than eight respondents (Systems, Biological, Environmental, Aerospace, and Earth & Atmospheric Engineering).

5.10 Model details and supplementary analyses

This appendix collects model details, validation tables, and additional analyses referenced from the main text.

5.10.1 Prior specification

The full prior specification is

$$\begin{aligned}
\boldsymbol{\mu}_0 &\sim \mathcal{N}\big((0.5, 0.5, 0.5, 0, 0, 0)^\top,\ \mathrm{diag}(1.5^2, 1.5^2, 1.5^2, 2^2, 1^2, 1^2)\big), \\
B_{p,d} &\sim \mathcal{N}(0, 1), \\
\sigma_d &\sim \mathrm{HalfNormal}(1), \\
\mathbf{L}_{\mathrm{corr}} &\sim \mathrm{LKJCholesky}(2), \\
\mu_\tau &\sim \mathcal{N}(\log 0.5,\ 0.5), \\
\sigma_\tau &\sim \mathrm{HalfNormal}(0.4),
\end{aligned} \tag{5.7}$$

with the four block temperatures drawn from the shared hierarchical prior $\log \tau_b \sim \mathcal{N}(\mu_\tau, \sigma_\tau)$ for $b \in \{\text{time}, \text{bunch}, \text{horizon}, \text{complex}\}$. A HalfNormal prior is a normal distribution restricted to positive values. The diagonal covariance for $\boldsymbol{\mu}_0$ lists variances explicitly, so the squared entries are elementwise variances. These priors regularize weakly identified individual-level parameters without fixing the substantive ordering of the effects.

5.10.2 Sampling diagnostics

The full-model fit produced zero divergent transitions. We also examined $\hat{R}$, the potential scale reduction factor, which compares variation within chains with variation between chains. Values close to 1 indicate similar sampling behavior across chains (Gelman and Rubin, 1992). The largest value across the model's

parameters was 1.09.

Penalty	Posterior mean	95% CrI
Same-day back-to-back (w_{oth})	1.851	$[0.632, 3.320]$
Evening–morning back-to-back (w_{eve})	1.235	$[0.334, 2.450]$
Two-in-three ($w_{2\text{in}3}$)	1.216	$[0.341, 2.483]$

Table 5.8: Raw population-level bunching penalties from softplus($\boldsymbol{\mu}_0$) in the individual model. Their common utility scale is less interpretable than the normalized comparisons in Table 5.4. Positivity is imposed by the softplus link, so a posterior probability of being positive is not reported as evidence.

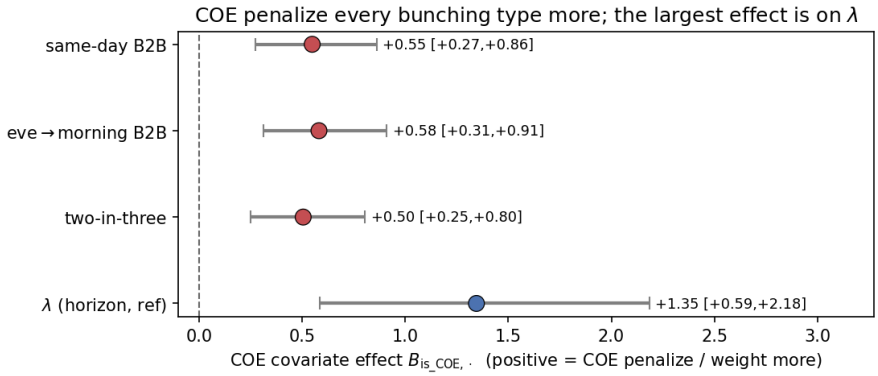


Figure 5.6: COE covariate effects $B_{\text{is_COE},\cdot}$ on each bunching penalty, with the scheduling-horizon effect λ for reference (posterior mean and 95% CrI). COE students penalize every bunching type more than non-COE students (all intervals above zero, $\Pr(>0) = 1.000$), and the largest single college effect is on the scheduling horizon.

Block	Questions	τ (mean)	95% CrI
τ_{time}	Q1 (time-of-day)	0.37	$[0.28, 0.47]$
τ_{bunch}	Q2 (bunching)	0.13	$[0.10, 0.17]$
τ_{horizon}	Q3 (scheduling horizon)	0.24	$[0.18, 0.30]$
τ_{complex}	Q4–Q6 (application)	0.66	$[0.51, 0.82]$

Table 5.9: Posterior block temperatures under the hierarchical (partial-pooling) prior. Smaller τ means sharper, more deterministic choices. The three anchor questions are all sharper than the application block, whose choice noise is roughly three to five times larger; the bunching anchor is the sharpest. Temperatures are estimated, not assumed, and are pooled toward a common mean ($\mu_\tau \to \tau \approx 0.34$, $\sigma_\tau = 0.61$).

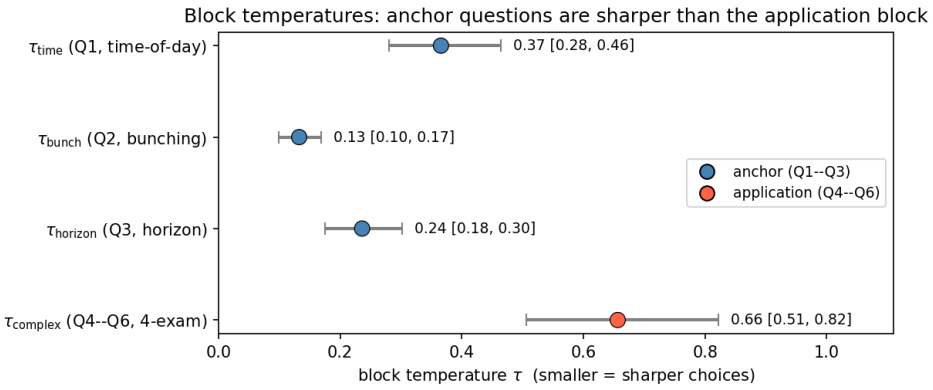


Figure 5.7: Posterior block temperatures (mean and 95% credible interval) for the four blocks, in presentation order. Smaller τ means sharper, more deterministic choices. The three anchor blocks (Q1–Q3) are all sharper than the application block (Q4–Q6), and the intervals do not overlap: the complex block's lower bound (0.51) exceeds every anchor block's upper bound. The bunching anchor (Q2) is the sharpest.

Predictor	LL/obs	Top-1	Bottom-1	Pairwise
Individual	$\mathbf{-1.73}^\star$	$\mathbf{75.0\%}^\star$	$\mathbf{66.1\%}^\star$	$\mathbf{82.4\%}^\star$
	[−1.76, −1.70]	[73.7, 76.2]	[64.8, 67.3]	[81.8, 83.0]
Covariate	−2.85	59.6%	44.4%	69.0%
	[−2.90, −2.79]	[58.3, 60.9]	[43.0, 45.7]	[68.2, 69.7]
Population	−2.74	58.5%	44.9%	67.9%
	[−2.78, −2.70]	[57.1, 59.8]	[43.7, 46.1]	[67.2, 68.6]
Modal	—	58.5%	52.7%	70.8%
		[57.2, 59.8]	[51.2, 54.2]	[69.9, 71.6]
Random	−3.178	25.0%	25.0%	50.0%

Table 5.10: In-sample fit of the three nested predictors over all five schedule questions (the data the model was fit on), $N = 981$. LL/obs is mean log-likelihood per observation; accuracies are percentages (pairwise = fraction of the six option-pairs ordered correctly, chance 50%). Grey brackets are 95% cluster-bootstrap confidence intervals (resampling students, 2000 replicates). *Modal* is a zero-modeling baseline that always predicts the single most-frequent ranking of each question (no LL, since a deterministic ranking gives $-\infty$ on any miss); the random baseline is analytic. The individual model dominates in-sample on every metric, but this rewards its $6N$ extra parameters. This is why the out-of-sample comparison (Table 5.2) is the decisive one. Note the modal baseline already matches the population model on top-1 and beats it on bottom-1 and pairwise. Per column, **bold** marks the best predictor (the modal baseline competes on accuracies but not LL; random is a reference floor), and a star (*) marks a 95% CI lying entirely above all competitors'.

Design	Comparison	Mean $\Delta\ell$ (SE)	95% CI	t	p_t	p_{Wilcoxon}
Held-out question	ind. > cov.	+0.245 (0.020)	$[+0.207, +0.284]$	12.4	3×10^{-33}	4×10^{-33}
Held-out question	ind. > pop.	+0.128 (0.020)	$[+0.090, +0.167]$	6.5	5×10^{-11}	3×10^{-13}
Held-out response	ind. > cov.	+0.160 (0.019)	$[+0.124, +0.196]$	8.6	1×10^{-17}	2×10^{-19}
Held-out response	ind. > pop.	+0.139 (0.019)	$[+0.103, +0.175]$	7.5	8×10^{-14}	4×10^{-17}

Table 5.11: Paired one-sided out-of-sample tests that the individual predictor's held-out log-score exceeds the covariate and population predictors'. *The test units are the $n = 981$ students*: because a student contributes several held-out 4-exam observations, we first average each student's per-observation log-score difference into a single value and test those 981 independent student-level differences, which avoids treating the clustered 2943 observations as independent. $\Delta\ell$ is the mean student-level difference in held-out log-likelihood; the 95% CI is the two-sided normal interval $\Delta\ell \pm 1.96\,\text{SE}$; p_t and p_{Wilcoxon} are the one-sided paired t-test and Wilcoxon signed-rank p-values.

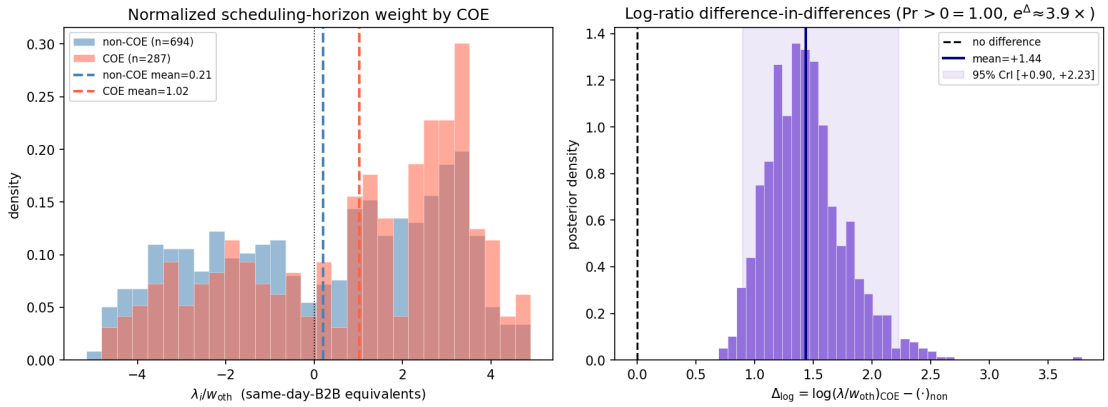


Figure 5.8: Normalized spacing preference, defined as willingness to finish later per unit of bunching aversion: $\rho_{i,\text{oth}} = \lambda_i / w_{i,\text{oth}}$, by COE status. The unit is same-day-back-to-back equivalents *over the full exam period*. Because x^{last} is the last occupied slot divided by 30, this unit corresponds approximately to moving the last exam across the full exam period. (a) Per-student normalized weights: both groups span the range but the COE distribution is shifted toward larger values (means 0.72 vs. 0.12); values near 0 correspond to indifference on the scheduling-horizon axis, large positive values to a strong preference for a later, more spread-out schedule. (b) Posterior of the log difference-in-differences $\Delta_{\text{log,oth}}$ (Table 5.3); the entire mass is above zero (mean +1.44, 95% CrI $[+0.90, +2.23]$), i.e. the COE normalized spacing preference is $\approx 4.2\times$ the non-COE value.

Held-out question	Metric	Individual	Covariate	Population
Anchor holdouts (test identification)				
Q1 (time) ids $\eta, u_{2\text{pm}}$	LL/obs	$\mathbf{-1.52}^{\star}$ [-1.57, -1.48]	-1.73 [-1.76, -1.70]	-1.78 [-1.79, -1.76]
	Top-1	51 [48, 54]	**55** [52, 59]	23 [20, 25]
	Bottom-1	**63*** [60, 66]	41 [38, 44]	41 [38, 44]
	Pairwise	**70*** [68, 71]	65 [63, 66]	48 [46, 50]
Q2 (bunch) ids $\boldsymbol{\pi}$	LL/obs	-3.51 [-3.61, -3.42]	-3.43 [-3.48, -3.37]	$\mathbf{-2.98}^{\star}$ [-3.00, -2.95]
	Top-1	70 [67, 72]	**86** [84, 88]	**86** [84, 88]
	Bottom-1	**38*** [34, 41]	15 [13, 18]	15 [13, 18]
	Pairwise	70 [69, 71]	61 [60, 62]	**72** [71, 73]
Q3 (horizon) ids λ	LL/obs	$\mathbf{-2.84}^{\star}$ [-2.92, -2.75]	-3.33 [-3.39, -3.27]	-3.18 [-3.21, -3.16]
	Top-1	**48*** [45, 51]	25 [23, 28]	25 [23, 28]
	Bottom-1	**49*** [46, 52]	31 [29, 34]	31 [28, 34]
	Pairwise	**69*** [68, 71]	55 [53, 57]	54 [52, 55]
Application holdouts (Q4–Q6; the main generalization test)				
Q4	LL/obs	$\mathbf{-2.52}^{\star}$ [-2.59, -2.46]	-2.75 [-2.84, -2.68]	-2.74 [-2.80, -2.68]
	Top-1	**61** [58, 64]	**61** [58, 64]	**61** [58, 64]
	Bottom-1	**63*** [60, 66]	51 [48, 54]	52 [49, 55]
	Pairwise	**76** [74, 77]	75 [73, 76]	75 [74, 76]
Q5	LL/obs	$\mathbf{-2.39}^{\star}$ [-2.45, -2.33]	-2.60 [-2.65, -2.55]	-2.61 [-2.66, -2.57]
	Top-1	**76** [73, 79]	**76** [73, 78]	**76** [73, 78]
	Bottom-1	44 [41, 48]	42 [39, 45]	**46** [43, 49]
	Pairwise	**73*** [72, 75]	69 [68, 70]	69 [67, 70]
Q6	LL/obs	-3.63 [-3.77, -3.50]	-3.92 [-4.02, -3.81]	$\mathbf{-3.57}$ [-3.65, -3.49]
	Top-1	**50** [47, 53]	44 [41, 47]	44 [41, 48]
	Bottom-1	**23*** [20, 25]	17 [15, 20]	17 [15, 20]
	Pairwise	**61** [59, 63]	59 [57, 60]	60 [58, 61]

Table 5.12: Leave-one-question-out validation. Each model is refit on five questions and evaluated on the sixth. LL/obs is mean held-out log-likelihood per observation; accuracies are percentages, with pairwise accuracy denoting the fraction of option pairs ordered correctly. Grey brackets are 95% cluster-bootstrap CIs over students. Bold marks the best predictor in each row; a star (*) marks a predictor whose CI lies entirely above both competing CIs. Q1–Q3 test identification, and Q4–Q6 test generalization.

Effect	Direction	Post. mean	95% CrI	Post. SD	Pr(> 0)
$B_{\text{is_COE},\lambda}$	> 0	$+1.35$	$[+0.59, +2.18]$	0.40	1.000

Table 5.13: COE effect on the scheduling-horizon preference λ in the individual model. The entire posterior mass lies above zero.

	Evening η		Horizon λ		Afternoon $u_{2\text{pm}}$	
Covariate	$\hat{B}$	95% CrI	$\hat{B}$	95% CrI	$\hat{B}$	95% CrI
is_COE	$+0.10$	$[+0.00, +0.21]$	$+1.35$	$[+0.59, +2.18]$	$+0.12$	$[+0.03, +0.22]$
year_2027	$+0.05$	$[-0.09, +0.18]$	$+0.09$	$[-0.80, +1.01]$	$+0.03$	$[-0.10, +0.15]$
year_2028	-0.09	$[-0.24, +0.03]$	-1.17	$[-2.13, -0.28]$	-0.07	$[-0.20, +0.05]$
year_2029	$+0.07$	$[-0.06, +0.21]$	-0.53	$[-1.46, +0.45]$	$+0.04$	$[-0.09, +0.15]$

Table 5.14: Posterior covariate effects $B_{p,d}$ on the three two-sided individual parameters, with 95% credible intervals. Reference category is non-COE $\times$ class of 2026. Only is_COE $\to \lambda$ (and, weakly, year_2028 $\to \lambda$) has a credible interval excluding zero by a substantial margin.

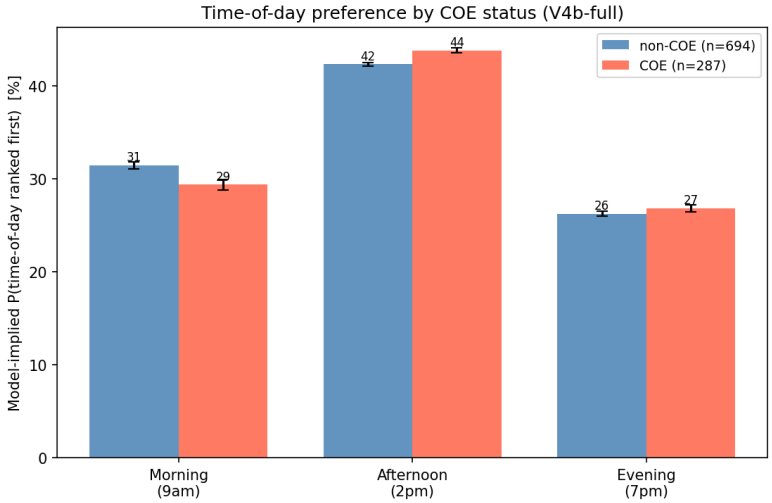


Figure 5.9: Model-implied probability that each time-of-day slot is ranked first, by COE status, from the per-student time utilities $(0, u_{2\text{pm},i}, \eta_i)$. Afternoon is most preferred for both groups; the profiles differ by at most about two percentage points. Error bars are standard errors of the mean across students.

CHAPTER 6

LLM-DERIVED PREFERENCE JUDGMENTS ARE NOT SELF-CONSISTENT

6.1 Introduction

A growing body of work asks a large language model (LLM) to turn a person's natural-language description of their preferences into structured preference data on which a decision procedure acts: explicit utility or score functions in LISTEN-U (Jovine et al., 2025) and LILO (Kobalczyk et al., 2025); item-level evidence for latent utilities in PEBOL (Austin et al., 2024); batch or tournament choices in LISTEN-T (Jovine et al., 2025); probabilistic priors over feature-utility weights from automated LLM interviews (Eichelbeck et al., 2026); and constraints for optimization models (Lawless et al., 2023; Sanguinetti et al., 2025). In these systems, LLM-derived or LLM-mediated preference signals become inputs to estimation, ranking, query selection, or optimization rather than merely generated prose, as illustrated by the representative workflow in Figure 6.1.

Many preference-learning systems represent or reason through a latent utility function that is assumed to be stable, although the precise observation model differs across interfaces. LISTEN-U and LILO represent or fit utility values directly. PEBOL updates a belief over latent item utilities, and LISTEN-T assumes a utility implicitly through transitivity and completeness. This follows the classical preference-learning view that judgments are noisy observations of a stable utility (Debreu, 1954; Kreps, 1988). Yet downstream performance does not establish that preference information obtained through different prompts is mutually consistent. We ask the more basic measurement question: can the population means of these numerical LLM judgments be reproduced by one stable quasi-linear,

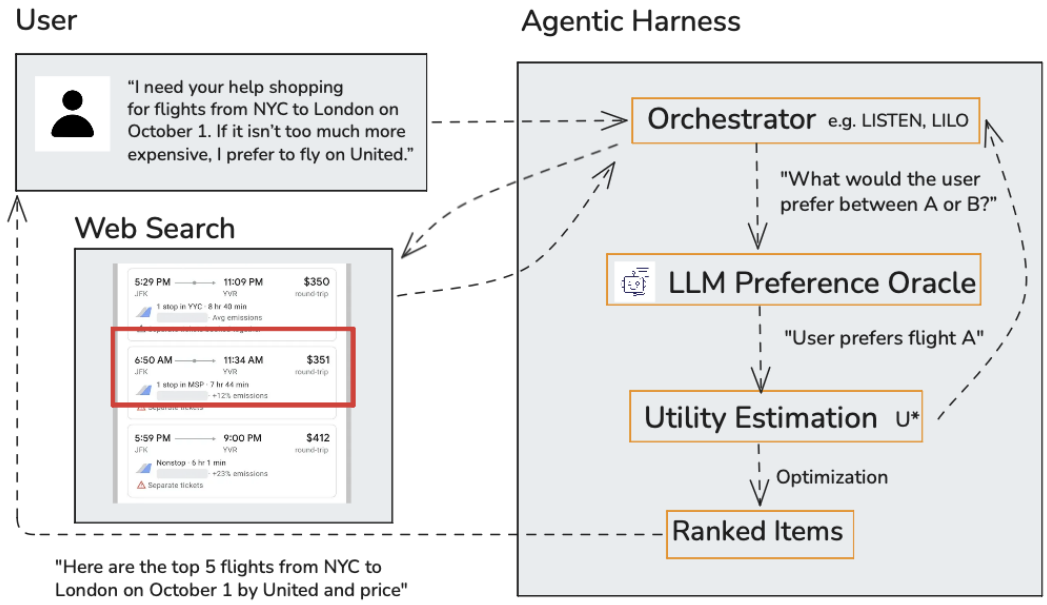


Figure 6.1: **Representative LLM-assisted preference learning.** Natural-language preferences condition LLM feedback used for utility estimation, optimization, and ranking; LISTEN and LILO instantiate related variants (Jovine et al., 2025; Kobalczyk et al., 2025).

dollar-denominated utility?

An analogous scalar-representation step appears in LLM evaluation and LLM feedback pipelines, where LLM-provided pointwise scores and pairwise judgments over candidate responses are used to construct scalar rewards (Zheng et al., 2023; Kim et al., 2023; Lambert et al., 2024). These settings use observation models different from ours and are not empirically audited here. They pose a broader representation question: can heterogeneous judgments over text-described alternatives be coherently compressed into one latent scalar? Our monetary setting is an externally anchored special case in which listed price fixes the units and supplies a prespecified cross-format relationship.

We study offers, a familiar format in everyday language and web text that pairs an item with a listed price. Price supplies a *numeraire* that places heterogeneous items on a common dollar scale. We assume we have a human *preference utterance*

to pass as context to influence the responses the LLM gives when queried. An *item query* asks the maximum price the user would pay for one item. An *offer-pair query* asks for the signed price change that would make the user indifferent between two offers. The sign supplies an ordinal preference direction, while the magnitude is a cardinal monetary judgment. Recovering such monetary trade-offs from binary choices instead requires price variation and a fitted choice model. For example, Reusens et al. (2026) infer hotel-attribute willingness to pay by fitting a multinomial-logit model to price-varying binary LLM choices. Direct numerical queries could avoid this indirect recovery step, but they make a stronger measurement claim: their dollar scale must be stable. In particular, the difference between two item-query responses should agree with the compensation elicited directly for the corresponding offers.

We formalize this claim as a self-consistency hypothesis. For each fixed preference utterance and item set, every item receives its own unrestricted, dollar-denominated utility. The audit is feature-agnostic but numeraire-dependent: it treats each item description as an atomic alternative rather than a feature vector, while listed price identifies a common cardinal scale across query types. We impose no linearity, smoothness, separability, or additivity over item features. The maintained economic restriction is quasi-linearity in the monetary numeraire: an offer's utility is its item's utility minus its listed price (Mas-Colell et al., 1995; Roughgarden, 2016). Self-consistency requires the expected responses from item and offer-pair queries to agree with this single utility. Even failure to reject would establish representability only for the audited finite items, queries, and prompt semantics, not generalization to unseen items or prompts.

Section 6.3 formalizes the self-consistency hypothesis H_{SC} and derives two necessary local implications. We test the hypothesis jointly, by measuring departure

from the best-fitting shared quasi-linear dollar utility, and locally, through P1 and P2. P1 compares a direct offer-pair estimate with a sum along a path of offer-pair queries; P2 compares an offer-pair estimate with the price-adjusted difference of two item-query estimates. A violation of either property provides evidence against H_{SC} for the queries involved, although satisfying both does not establish the full hypothesis. Because exact tests can detect operationally small disagreements, we report both statistical evidence and effect magnitudes in dollars and relative to listed prices. Applying the audit to controlled flight, apartment, and hotel examples across six models reveals persistent disagreement, particularly between query types. The audit measures internal coherence rather than accuracy for the person who supplied the preference description: rejection provides evidence against a shared quasi-linear dollar utility for the audited means, while failure to reject does not establish human fidelity.

Our contributions are:

- We formulate the audit over finite sets of text-described items, assigning one unrestricted dollar value to every item and imposing only quasi-linearity in listed price; no feature-based utility form is assumed.
- We develop a reusable audit protocol for numerical LLM preference interfaces with a monetary numeraire: a joint bootstrap test of whether one such utility fits all query means, together with local P1/P2 diagnostics reporting statistical evidence, price-scaled magnitude, and supported preference reversals.
- We demonstrate the audit across three controlled domains and six models, finding persistent disagreement across query types.

6.2 Related Work

LLM-derived and LLM-mediated preference signals. The systems closest to our setting elicit preferences from an LLM as input to a downstream selection or optimization procedure. LISTEN maps a natural-language preference description to explicit utilities, batch choices, or a selected item for multi-objective selection (Jovine et al., 2025); LILO fits Gaussian-process utilities from LLM-generated scalar or pairwise feedback within a Bayesian-optimization loop (Kobalczyk et al., 2025); and PEBOL converts free-text user replies into item-level evidence for Bayesian utility beliefs that drive acquisition (Austin et al., 2024). In the rental-listing setting of Eichelbeck et al. (2026), an automated LLM interview initializes a prior over feature-utility weights that subsequent user comparisons update. Beyond personalized decision support, LLM feedback over candidate text responses can provide pairwise labels for training preference models (Bai et al., 2022); reward models assign scalar scores and are commonly evaluated on preferred–rejected response pairs (Lambert et al., 2024). These systems use observation models different from ours and are not empirically audited here, but they similarly turn LLM-derived judgments into scalar signals used for ranking or optimization. Translating specified optimization problems into solver form (Ramamonjison et al., 2023; AhmadiTeshnizi et al., 2024) or extracting constraints from natural language (Lawless et al., 2023; Sanguinetti et al., 2025) addresses a different task: the objective is specified rather than elicited.

Feedback form and shared scalar representation. LILO compares its default binary pairwise labels with a scalar variant scoring each outcome in $[0, 1]$, but the comparison simultaneously changes the response scale, observation model, and points selected by the sequential policy (Kobalczyk et al., 2025). Pointwise scores on bounded or rubric-anchored scales (Zheng et al., 2023; Kim et al., 2023) and

pairwise judgments over the same kinds of text-described items both appear in LLM feedback and evaluation. These feedback forms do not automatically share a cardinal scale: interpreting pairwise choices through scalar rewards requires a specified choice model, while bounded scores require a specified scale interpretation. Reusens et al. (2026), for example, infer attribute-level willingness to pay by fitting an additive multinomial-logit model to price-varying binary LLM choices over hotel rooms.

Our item-query and offer-pair responses are instead expressed directly in dollars. Listed price supplies the external numeraire, and the maintained assumption of quasi-linearity in that numeraire specifies how the two query types should relate. We therefore hold the monetary scale and compared items fixed and test their agreement before downstream fitting, assigning each audited item an unrestricted utility rather than imposing a feature-based model.

Utility representation and consistency audits. The assumption that responses are consistent with a utility function inherits from work on learning from comparisons expressed by human decision-makers. Transitive and complete preferences over a countable item set are represented by a utility function (Debreu, 1954; Kreps, 1988). Humans do report intransitive judgments (Tversky, 1969), but violations concentrate among items of near-equal utility (Luce, 1956; Fishburn, 1991), and once response noise is modeled, gross violations of transitivity are rare (Regenwetter et al., 2011). This literature concerns ordinal coherence; our audit additionally requires that elicited dollar magnitudes cohere.

Our global fit and path diagnostics are also related to HodgeRank, which fits a global node potential to cardinal pairwise measurements and characterizes cyclic inconsistency (Jiang et al., 2011). Our setting combines item-query level anchors with offer-pair measurements, assigns the measurements monetary offer semantics,

and tests whether the two query types agree under quasi-linearity. LLM-judge work documents ordering, scoring, and presentation biases (Zheng et al., 2023; Wang et al., 2024; Haldar and Hockenmaier, 2025), while recent audits test logical conditions such as transitivity and commutativity (Feng et al., 2026; Liu et al., 2024). Those tests are primarily ordinal or logical. A transitive sign ordering does not identify coherent dollar magnitudes (Appendix 6.8.2); our audit tests cardinal integrability and agreement between item and offer-pair queries.

6.3 Utilities and LLM Estimates

6.3.1 Items, Offers, and Utility

Fix a domain containing a set of items $\mathcal{X} = \{x_1, \ldots, x_d\}$, a natural-language preference utterance, an LLM configuration (comprised of model choice and settings), and prompt templates. An item is an outcome represented to the LLM by a textual description; in our experiments, items are flights, apartment rentals, and hotel rooms. The prompts include the utterance when asking the LLM about these items. Section 6.4 defines the concrete choices used in our experiments.

We hypothesize the LLM responses are self-consistent: their population means can be described by a utility function U. Let $U(x)$ denote item x's utility under this hypothesized function, denominated in a reference currency (US dollars in our experiments). If no such utility reproduces all query means, then the responses are not internally consistent with a single dollar-denominated utility. Provided the person's preferences admit this representation, at least one query type must misestimate that utility on at least one audited query. Conversely, failure to reject self-consistency does not establish that the utility accurately represents the person's preferences.

An *offer* is a tuple $a = (x, p)$ consisting of item x and its listed price p. We assume quasi-linearity in money:

$$U(a) = U(x) - p.$$

This is a standard model for valuations with payments (Mas-Colell et al., 1995; Roughgarden, 2016). We make no other assumption about the functional form of U: the item utilities are unrestricted and need not be linear, smooth, separable, or additive in item features.

For offers $a = (x, p)$ and $b = (y, q)$, their target-minus-source utility difference is

$$U(b) - U(a) = [U(y) - q] - [U(x) - p].$$

6.3.2 Two Query Types

We use two prompt types; Appendix 6.9.3 gives their exact wording. An *item query* names an item x without a listed price and asks the maximum amount the user would pay for it. We interpret that amount as $U(x)$ relative to not buying the item.

An *offer-pair query* names source offer $a = (x, p)$ and target offer $b = (y, q)$ and asks how much the target price q would have to change to make the user indifferent. A positive response means that b could become more expensive; a negative response means that it must become cheaper. Thus the sign implies a pairwise preference.

Stylized prompts are: **Item query:** "Given the user's preference description and item x, what is the maximum price they would pay for x?" **Offer-pair query:** "Offer A is item x at price p, and Offer B is item y at price q. What signed amount should be added to B's price to make the user indifferent between the offers? A positive amount means B could cost more; a negative amount means B must become cheaper." Every prompt requests one numerical dollar amount; the exact wording is included with the released query specification.

Item queries are indexed by $x \in \mathcal{X}$, and offer-pair queries by an ordered pair of offers (a, b). Let $\mathcal{C}$ be the union of these two index sets. For any query $c \in \mathcal{C}$, let $Y^i(c)$ be the response from its ith LLM call. Thus $Y^i(x)$ denotes an item-query response because its argument is an item, whereas $Y^i((a, b))$ denotes an offer-pair response because its argument is an ordered pair of offers.

Within a fixed query, we treat finite responses as independent and identically distributed across calls. Distributions may differ across queries, including in their variances, and need not be Gaussian, symmetric, or unimodal. For n repeated calls to query c, define

$$\overline{Y}(c) := \frac{1}{n} \sum_{i=1}^{n} Y^i(c). \tag{6.1}$$

When a finite collection is enumerated by $c = 1, \ldots, Q$, we use the equivalent subscript notation Y_c^i and $\overline{Y}_c$.

6.3.3 Self-Consistency Hypothesis

Fix a finite query collection $\mathcal{Q} \subset \mathcal{C}$ of cardinality Q before observing any responses. For a candidate utility U, define the response implied by query c as

$$m_c(U) := \begin{cases} U(x), & c = x, \\ U(b) - U(a), & c = (a, b). \end{cases} \tag{6.2}$$

The first case is an item query and the second is an offer-pair query. The *self-consistency hypothesis* for this collection is

$$\begin{aligned} H_{\mathrm{SC}} : \quad & \exists\, U \text{ such that} \\ \mathbb{E}[Y^i(c)] = m_c(U) \quad & \text{for every } c \in \mathcal{Q}. \end{aligned} \tag{6.3}$$

Its alternative is the logical negation: no single U satisfies every equality, or equivalently every candidate U leaves at least one query mean unexplained. Expectations are over stochastic completions, conditional on the fixed audit inputs.

Best-fitting utility and discrepancies. Whether or not the audited null holds, define the *best-fitting population utility*

$$U^\star \in \arg\min_U \frac{1}{Q} \sum_{c \in \mathcal{Q}} \{\mathbb{E}[Y^i(c)] - m_c(U)\}^2. \tag{6.4}$$

Define each query's persistent discrepancy from this fit as

$$\delta(c) := \mathbb{E}[Y^i(c)] - m_c(U^\star).$$

$U^\star$ is the utility assignment that minimizes the average squared discrepancy over $\mathcal{Q}$; it is not an additional assumption about the person's true utility. The minimized objective in Equation 6.4 is the average of $\delta(c)^2$. Under H_{SC}, every discrepancy is zero. Under the alternative, at least one is nonzero. Thus the null says that one utility assignment absorbs all persistent query means; the alternative says that some persistent mean disagreement remains after fitting the best possible shared utility.

Individual calls may fluctuate around their query mean. Equivalently,

$$Y^i(c) = m_c(U^\star) + \delta(c) + \varepsilon^i(c), \qquad \mathbb{E}[\varepsilon^i(c)] = 0. \tag{6.5}$$

Repeated calls reduce the sampling variation represented by $\varepsilon^i(c)$ but do not remove the persistent discrepancy $\delta(c)$. These discrepancies are relative to the shared-utility model, not necessarily biases relative to a person's true preferences.

Audit overview. We assess self-consistency globally, by asking whether one utility jointly explains a prespecified collection of query means, and locally, by comparing alternative estimates for the same offer pair. The global test summarizes joint lack of fit; P1 and P2 identify specific disagreements.

6.3.4 Global Test for Self-Consistency

Using the observed query means, estimate the best-fitting utility by

$$\widehat{U} \in \arg\min_{U} \frac{1}{Q} \sum_{c \in \mathcal{Q}} \{\overline{Y}(c) - m_c(U)\}^2. \tag{6.6}$$

We measure the joint lack of fit remaining in the observed means by

$$T = \left\{ \frac{1}{Q} \sum_{c \in \mathcal{Q}} \{\overline{Y}(c) - m_c(\widehat{U})\}^2 \right\}^{1/2}. \tag{6.7}$$

Thus T is the smallest *root mean squared error* (RMSE) achievable by assigning one free utility value to every audited item. An item query constrains one item value; an offer-pair query constrains the price-adjusted difference between two item values. Every prespecified query receives equal weight, so T is the RMSE for a uniformly selected query from $\mathcal{Q}$. This defines the reported effect size; it is not a claim that equal weighting is the most efficient way to estimate U.

As the number of repetitions grows, $\widehat{U}$ approaches $U^\star$. Under H_{SC}, T approaches zero because only finite-sample noise remains. Under the alternative, it approaches the RMSE of the persistent discrepancies $\delta(c)$. Larger departures from self-consistency therefore shift the distribution of T toward larger values. Lemma 6.2 gives the corresponding finite-sample expectation of the squared lack-of-fit statistic.

Because numerical responses may be non-Gaussian and have different variances across queries, we use a within-query nonparametric bootstrap to estimate the null distribution of T and obtain one p-value for $\mathcal{Q}$ (Efron and Tibshirani, 1993). The *observed RMSE* is T computed from the actual query sample means after fitting $\widehat{U}$; it is not error against human ground truth. Details are in Appendix 6.13.1.

6.3.5 Local Tests for Self-Consistency

Self-consistency implies that every way of estimating the utility difference between the same two offers agrees in expectation. The following comparisons test this

implication without first fitting a shared utility to all query means. Proposition 6.1 states both implications formally and Appendix 6.8.2 proves them.

P1: direct offer-pair estimate versus path sum. Choose a path $\mathcal{P} = (a = a_0, a_1, \ldots, a_k = b)$ from source offer a to target offer b, and ask separately about each adjacent pair. Section 6.4.2 specifies the prespecified paths used in our audit.

$$\sum_{j=1}^{k} \overline{Y}((a_{j-1}, a_j)). \tag{6.8}$$

Under H_{SC}, utility differences telescope, so this path sum and the direct offer-pair estimate have the same expectation:

$$\mathbb{E}\left[\sum_{j=1}^{k} \overline{Y}((a_{j-1}, a_j))\right] = U(b) - U(a) = \mathbb{E}[\overline{Y}((a, b))].$$

Define the P1 residual

$$\widehat{R}_{1,\mathcal{P}}(a, b) := \overline{Y}((a, b)) - \sum_{j=1}^{k} \overline{Y}((a_{j-1}, a_j)).$$

P2: offer-pair estimate versus item queries. For the same offers $a = (x, p)$ and $b = (y, q)$, an offer-pair query directly estimates $U(b) - U(a)$. Alternatively, two item queries give the estimate

$$[\overline{Y}(y) - q] - [\overline{Y}(x) - p]. \tag{6.9}$$

Under H_{SC}, its expectation is also $U(b) - U(a)$. Equivalently, adding the known price difference to the expected offer-pair response gives the item utility difference, $\mathbb{E}[Y^i((a, b))] + (q - p) = U(y) - U(x)$. Define the P2 residual

$$\widehat{R}_2(a, b) := \overline{Y}((a, b)) - \{[\overline{Y}(y) - q] - [\overline{Y}(x) - p]\}.$$

P2 is not an assumption that humans answer differently framed questions identically. It is an implication of treating both LLM query types as estimates of one U.

Statistical interpretation. For either property, the local null is that the two estimates being compared have the same population mean, so the corresponding residual has expectation zero. The local alternative is that their population means differ, so the residual has a nonzero expectation. This is a specific implication of H_{SC}: rejecting a local null rejects self-consistency for those queries, whereas failing to reject it does not establish global self-consistency.

Because each local residual is a prespecified linear combination of query averages from independent calls, we estimate its standard error from the constituent queries. We use a finite-sample normal approximation motivated by the central limit theorem and form the comparison-specific 95% confidence interval $\widehat{R} \pm 1.96\,\widehat{\mathrm{SE}}(\widehat{R})$. This approximation does not require equal response variances across queries, although it does require the sampling distribution of the residual to be reasonably approximated by a normal distribution. A local self-consistency rejection occurs when this interval excludes zero. Under the local null, the interval becomes concentrated around zero as calls are added; under a fixed alternative, it becomes concentrated around the nonzero population residual. We use this direct calculation locally because the residual is a prespecified linear combination. The global statistic instead requires refitting a shared utility and is calibrated by bootstrap.

Figure 6.2 summarizes the two local implications of self-consistency: P1 compares different paths of offer-pair queries, whereas P2 compares offer-pair and item-query estimates of the same utility difference.

6.3.6 Statistical Evidence and Practical Magnitude

The global and local tests concern exact equalities between population means. With enough repeated calls, either can detect a fixed disagreement too small to matter operationally. We therefore separate evidence against exact self-consistency, given by

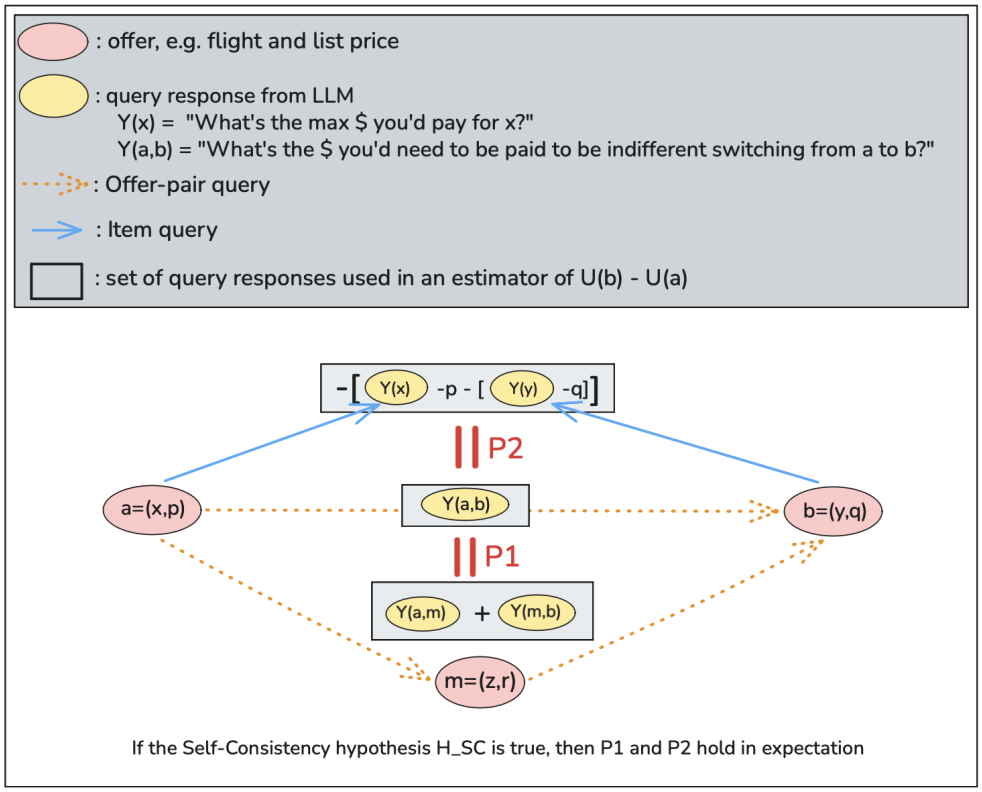


Figure 6.2: **Local implications of self-consistency.** For offers $a = (x, p)$ and $b = (y, q)$, all three constructions estimate $U(b) - U(a)$. P1 compares the direct offer-pair response $Y(a, b)$ with the path sum $Y(a, m) + Y(m, b)$. P2 compares $Y(a, b)$ with the price-adjusted item-query difference $[Y(y) - q] - [Y(x) - p]$. Self-consistency requires these estimates to agree in expectation, not on every LLM call.

rejection and its p-value or confidence interval, from practical magnitude, reported through the global RMSE T and local $|\widehat{R}|$. Section 6.4.3 specifies the normalizations, reversal criterion, and family-level reporting used in our experiments.

6.4 Audit Protocol

6.4.1 Domains and Preference Utterances

We construct finite sets of text-described items in three controlled domains: flights, apartments, and hotels. Flight and apartment descriptions vary continuous, ordinal,

Domain	Audited items	Audited offers	Utts.	Pairs/ utt.	Offer-pair queries	Item queries	All queries
Flights	19	21	3	6	90	24	114
Apartments	21	21	3	6	90	27	117
Hotels	3	9	3	4	60	9	69
Total	43	51	9	16	240	60	300

Table 6.1: **Audit design.** Pairs/utt. counts endpoint offer comparisons under one preference utterance. Each pair generates five offer-pair queries: one direct query and four steps on two two-step paths. Item-query estimates are reused across endpoint pairs but are elicited separately for each utterance. Each of the 300 query cells receives 15 completions, giving 4,500 calls per model.

and categorical item features; hotels combine three rating-defined room items with three listed prices to form nine offers. These features are used only to construct interpretable items, comparisons, and paths. The audit itself uses item identity, listed price, and query incidence; it does not fit utility as a function of item features. These controlled test cases make the P1 and P2 relationships explicit but are not representative samples of people or markets.

Before collecting responses, we fix three utterances per domain that vary emphasis on price and domain-specific features. An *audit group* fixes one model, domain, utterance, item set, and query collection; its utility fit and test statistic are computed separately. This gives nine groups per model. Appendices 6.9.1 and 6.9.2 summarize the audited design and give the preference utterances.

6.4.2 Queries, Models, and Sampling

The prespecified design in Table 6.1 contains 16 endpoint-comparison templates (six flight, six apartment, and four hotel), each evaluated under three domain-specific utterances, giving 48 comparison instances across nine audit groups.

Across the 48 endpoint pairs, the design yields 96 P1 and 48 P2 comparisons. Reuse explains why 60 item queries suffice, while some path-only intermediate items receive no item query. The two P1 residuals for a pair share its direct estimate, and

P2 residuals can share item estimates across pairs; individual comparison-specific intervals remain valid, but pooled residuals are not independent.

We use one canonical wording per query type. Each independent prompt contains the utterance and relevant item or offers, but no earlier path questions or answers. Thus the audit concerns stateless rather than conversational elicitation. Appendix 6.9.3 gives the exact prompts.

Models and sampling. We audit Claude Opus 4.8, Gemini 3.5 Flash, GPT-5.5, GPT-OSS 120B, Llama 3.3 70B, and Qwen 3.6 27B, analyzing each separately. Each query receives 15 calls, as summarized in Table 6.1. Appendix 6.9.4 reports service providers, model identifiers, sampling controls, token limits, and reasoning settings. Analyses condition on finite numerical responses; parse failures are reported in Appendix 6.10.

6.4.3 Inference and Reporting

Each audit group provides one bootstrap p-value. Within each model, we apply a Bonferroni correction, rejecting a group only when $p \leq 0.05/9 = 0.0056$ (Dunn, 1961). This controls the probability of any false rejection at 5% without assuming group independence. We report whether the all-nine claim is rejected and how many groups reject after correction.

We report T_g in dollars and relative to the mean of the endpoint listed prices across that group's offer-pair queries, with queries weighted as they appear in the design. This is one group-level scale. For local residuals, we instead use a comparison-specific scale and report dollars and

$$\frac{|\widehat{R}|}{s_{ab}}, \qquad s_{ab} := \frac{p+q}{2},$$

the absolute residual as a fraction of endpoint offers a and b's mean listed price;

intermediate path prices do not enter this scale. For example, 0.10 means disagreement equal to 10% of that mean endpoint price. A *95% supported ordinal reversal* has estimates whose confidence intervals (CIs) exclude zero in opposite directions, so the two query constructions support opposite preferred offers beyond sampling uncertainty. A *local rejection* instead has a P1 or P2 residual CI excluding zero; it can reject equality without reversing the implied preference. Empirical complementary cumulative distribution function (CCDF) curves show the fraction of residual magnitudes exceeding each threshold. These intervals and lower bounds are comparison-specific, not simultaneous confidence statements, and describe local prevalence and magnitude rather than an additional family-level claim. Appendix 6.13.2 defines standard errors (SEs), CIs, and lower bounds; Appendix 6.13.1 gives bootstrap and multiplicity details.

6.5 Audit Results

We first test whether all query means admit one utility and then use P1 and P2 to identify disagreements across paths and query types. Statistical rejection addresses exact self-consistency; RMSE, price-normalized residuals, and supported reversals describe practical magnitude.

6.5.1 Does One Utility Fit the Query Means?

Every model rejects the joint claim that all nine audit groups are self-consistent after Bonferroni correction. Five models reject all nine groups, while Qwen rejects four. Because each fit assigns one unrestricted utility to every audited item, these rejections are not failures of a particular feature-based or parametric utility model. Under the maintained quasi-linear price relation, they provide evidence that no

Model	Flights $ (%)	Apartments $ (%)	Hotels $ (%)
Claude Opus 4.8	16.9 (4.6%)	38.3 (1.6%)	109.2 (19.9%)
Gemini 3.5 Flash	7.7 (2.1%)	47.5 (2.0%)	104.0 (18.9%)
GPT-5.5	14.6 (4.0%)	51.7 (2.2%)	105.7 (19.2%)
GPT-OSS 120B	10.9 (3.0%)	73.9 (3.2%)	127.6 (23.2%)
Llama 3.3 70B	18.1 (4.9%)	142.3 (6.1%)	198.0 (36.0%)
Qwen 3.6 27B	18.4 (5.0%)	65.3 (2.8%)	247.2 (44.9%)

Table 6.2: **Magnitude of lack of fit.** Cells average three audit-group RMSEs and report dollars (percentage of the group-level endpoint-price scale; Section 6.4.3). Each fit assigns one free utility per item. Hotels have 18.9–44.9% RMSE versus 1.6–6.1% elsewhere, but query geometry and fit rank differ by domain.

assignment of item utilities reproduces all audited query means. Group-level results are in Appendix Tables 6.9 and 6.10.

Table 6.2 reports the remaining error after fitting the best possible utility. Average RMSE ranges from 1.6–6.1% of the endpoint-price scale for flights and apartments and 18.9–44.9% for hotels. These are descriptive rather than controlled domain comparisons because the domains differ in query geometry and fit rank. The global audit establishes disagreement but does not identify its source or which responses better represent the person's preferences.

6.5.2 Where Does Self-Consistency Fail?

P2 provides the clearest local failure. Across models, 41.7–87.5% of P2 residual confidence intervals exclude zero (Table 6.3), and Figure 6.3 shows that many disagreements are substantial relative to offer prices. Supported reversals occur for 2.1–12.5% of the 48 prespecified comparisons per model. In each such comparison, a two-offer selector based on item-query means chooses the opposite offer from one based on the offer-pair mean. Thus query type can affect both the estimated utility difference and, in some audited cases, the resulting choice. The audit does not establish which choice is more faithful to the person; domain-level results are in

	P1 ($n = 96$)		**P2** ($n = 48$)	
Model	**Rev.**	**Reject.**	**Rev.**	**Reject.**
Claude Opus 4.8	4 (4.2%)	66 (68.8%)	6 (12.5%)	41 (85.4%)
Gemini 3.5 Flash	1 (1.0%)	51 (53.1%)	4 (8.3%)	37 (77.1%)
GPT-5.5	0 (0.0%)	37 (38.5%)	1 (2.1%)	40 (83.3%)
GPT-OSS 120B	0 (0.0%)	38 (39.6%)	2 (4.2%)	31 (64.6%)
Llama 3.3 70B	2 (2.1%)	53 (55.2%)	5 (10.4%)	42 (87.5%)
Qwen 3.6 27B	0 (0.0%)	6 (6.2%)	1 (2.1%)	20 (41.7%)

Table 6.3: **Local self-consistency audit.** Entries are counts (percentages). P1 compares a direct offer-pair estimate with a path sum; P2 compares it with the difference of two item-query estimates. Rev. denotes a supported reversal, whose estimate CIs exclude zero in opposite directions; Reject. denotes a residual CI excluding zero.

Appendix Table 6.8.

P1 and path composition. P1 failures are less uniform across models: rejection rates range from 6.2% to 68.8%, supported reversals are rare, and price-normalized magnitudes are generally smaller than for P2. These results provide evidence against unrestricted composition of offer-pair estimates in the audited design, but not a robust cross-model ordinal effect.

A supplementary stress test divides four smooth single-feature changes into increasingly fine paths. For every model, mean price-normalized P1 error is higher at $k = 8$ and $k = 16$ than at $k = 2$ (Appendix Figure 6.4 and Table 6.7). In these four paths, simple local steps do not guarantee agreement with the direct estimate. The small design and nonmonotone curves do not establish a general scaling law.

6.6 Implications for Preference Learning

The large and statistically significant inconsistencies that we observe do not by themselves show that preference information elicited from LLMs is unusable; instead, these inconsistency show that algorithm designers should be careful when using preference information from LLMs. This information depends on the way in which

questions are asked and the choice of LLM. When possible, algorithm designers should corroborate an LLM's preference judgements with real human judgements.

Our audit can be repeated with new items and price changes to check consistency in application domains before numerical elicitation is applied. After performing such an audit, an algorithm designers may may wish to choose an LLM with better self-consistency, using the metrics we develop, because using an LLMs with poor self-consistency risks results that depend arbitrarily on the specific form of questions asked.

Our audit also points toward human-subject research. Self-consistency is necessary for the shared-utility model but not sufficient for fidelity: an LLM whose responses admit a single quasi-linear dollar utility may still assign values that a person would not endorse, and our design cannot detect this. Measuring that gap requires eliciting the same item and offer-pair queries from people who supply the preference description, which would identify which query construction, if either, better recovers stated human values. A second direction is to calibrate the inconsistencies we report against human ones. Humans are themselves inconsistent, and framing effects in stated willingness to pay are well documented, so the relevant question for a designer is not whether LLM responses depart from a single utility but whether they depart more than the human judgments they stand in for. Running our protocol on human subjects and LLMs over the same items would answer this, and would also show whether the failures concentrate in the same comparisons.

6.7 Conclusion

We test whether heterogeneous numerical LLM answers estimate one utility. Across all six models, the joint claim that all nine audit groups are self-consistent is rejected after Bonferroni correction. P2 disagreements are frequent, often large relative to

price, and sometimes reverse the preferred offer; P1 and path-length results caution against unrestricted composition but are more model- and design-dependent. The audit does not establish human fidelity; it identifies when query types or paths cannot be treated as interchangeable under a shared quasi-linear dollar utility.

Limitations

This study audits internal coherence, not human fidelity, and uses no human data. Controlled items, hand-written utterances, three domains, six models, fixed prompts and provider versions, stateless calls, and 15 completions are not representative; real users, larger rankings, paraphrases, model versions, and conversational histories may differ. The quasi-linear dollar model excludes wealth effects, binding budgets, and cases without finite compensation; P2 covers price-free item queries, not all utility prompts. Analyses condition on parseable outputs and do not compare numerical with binary elicitation. The path-length check uses only four smooth single-feature paths and is not a scaling law.

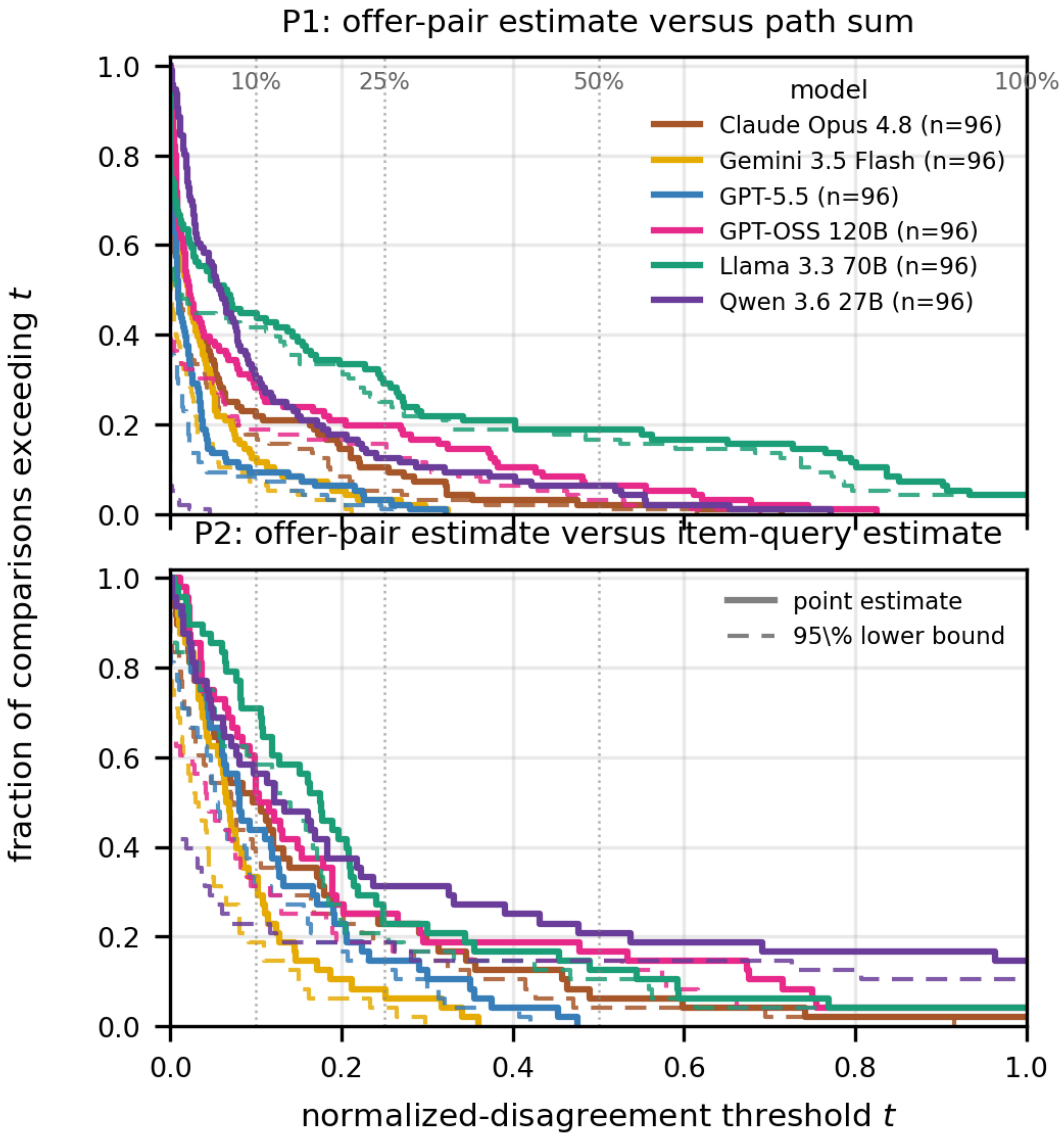


Figure 6.3: **Local disagreement by query construction.** P1 compares a direct offer-pair estimate with a path sum; P2 compares it with the price-adjusted difference of two item-query estimates. At threshold t, the curve gives the fraction with $|\widehat{R}|/[(p+q)/2] \geq t$; thus, $t = 0.10$ means disagreement of at least 10% of the endpoint offers' mean price. Axes end at $t = 1$; larger residuals still contribute to the curve height there. Solid curves use observed residuals; dashed curves use comparison-specific 95% lower bounds, not simultaneous confidence bands.

Appendices to Chapter 6

6.8 Formal basis of the audit

6.8.1 Why the queries estimate item utilities and offer-utility differences

Section 6.3.2 gives the prompt semantics. Here we derive their targets under quasi-linearity. Let v be the direct query's target amount. Utility is unchanged for our purposes if the same constant is added to every outcome, so we subtract the utility of buying nothing from every outcome and use buying nothing as the zero-utility reference. Hence $U(x)$ is the utility gain from obtaining x relative to buying nothing. This normalization is imposed in our measurement model, not stated in the LLM prompt. We interpret the prompt's "maximum amount" as the price at which the item and the outside option are equally desirable. Indifference at that price then gives

$$U(x) - v = 0, \qquad \text{so} \qquad v = U(x).$$

This normalization anchors the otherwise arbitrary utility level, and self-consistency therefore requires $\mathbb{E}[\overline{Y}(x)] = U(x)$.

Let r be the signed amount added to the target price in an offer-pair query from $a = (x, p)$ to $b = (y, q)$. Indifference after that adjustment gives

$$\begin{aligned} U(y) - (q + r) &= U(x) - p, \\ r &= [U(y) - q] - [U(x) - p] \\ &= U(b) - U(a). \end{aligned}$$

Thus the offer-pair query targets the signed target-minus-source utility difference between the two offers.

6.8.2 Why P1 and P2 follow from self-consistency

Proposition 6.1 (Local implications of self-consistency)**.** *Suppose H_{SC} holds. For any offers a, b and any path $\mathcal{P} = (a = a_0, a_1, \ldots, a_k = b)$,*

$$\mathbb{E}[\overline{Y}((a,b))] = \mathbb{E}\left[\sum_{j=1}^{k} \overline{Y}((a_{j-1}, a_j))\right].$$

Moreover, if $a = (x, p)$ and $b = (y, q)$, then

$$\mathbb{E}[\overline{Y}((a,b))] = [\mathbb{E}[\overline{Y}(y)] - q] - [\mathbb{E}[\overline{Y}(x)] - p].$$

Consequently, the population P1 and P2 residuals are zero.

Proof. For P1, let $\mathcal{P} = (a = a_0, a_1, \ldots, a_k = b)$ be any path from source offer a to target offer b. Because each offer-pair estimate has expected value equal to its utility difference under H_{SC},

$$\begin{aligned}
\mathbb{E}\left[\sum_{j=1}^{k} \overline{Y}((a_{j-1}, a_j))\right] &= \sum_{j=1}^{k} \mathbb{E}[\overline{Y}((a_{j-1}, a_j))] \\
&= \sum_{j=1}^{k} [U(a_j) - U(a_{j-1})] \\
&= U(b) - U(a) \\
&= \mathbb{E}[\overline{Y}((a,b))].
\end{aligned}$$

The middle sum telescopes. For P2, the definitions give

$$\begin{aligned}
\mathbb{E}[\overline{Y}((a,b))] &= [U(y) - q] - [U(x) - p] \\
&= [\mathbb{E}[\overline{Y}(y)] - q] \\
&\quad - [\mathbb{E}[\overline{Y}(x)] - p].
\end{aligned}$$

Thus a nonzero expected P1 or P2 discrepancy is sufficient to reject self-consistency on those queries.

Graph interpretation. Represent offers as nodes and offer-pair estimates as directed edges. On each connected component, expected offer-pair estimates can be represented by one offer utility exactly when every cycle sum is zero. To see this, fix a reference offer and define the utility of another offer by summing edge means along a path from the reference. Zero cycle sums make the definition path-independent; conversely, utility differences telescopically sum to zero around every cycle. The utility is unique up to an additive constant. P1 audits selected cycles formed by one direct edge and one decomposed path; it does not claim to enumerate every cycle in a larger item graph.

When the representation holds for all relevant pairs, signs inherit the ordering of scalar utilities. Antisymmetry follows for reversed edges, and transitivity follows because $U(y) > U(x)$ and $U(z) > U(y)$ imply $U(z) > U(x)$. These ordinal consequences follow from the cardinal representation. The converse is not true: a transitive sign ordering does not identify coherent dollar magnitudes. Item-query estimates add level measurements, and P2 tests whether those levels agree with the offer graph.

6.9 Experimental reproducibility details

6.9.1 Domains, items, and query counts

Table 6.4 summarizes the items, offers, and query counts used in the audit. A *source offer* is the fixed endpoint from which a *target offer* is compared.

Source offers and hotel grid. The flight source offers are `gnd_cheap_fast_delta` ($220, 5.0 h, Delta), `gnd_mid_united` ($360, 6.5 h, United), and `gnd_expensive_slow_american` ($520, 8.0 h, American). The

	Flights	Apartments	Hotels
Audited items	19	21	3
Audited offers	21	21	9 (3×3)
Numeraire	price	rent	price
Item features	travel time (continuous), airline (categorical)	area, commute (continuous), bedrooms (ordinal), building type (categorical)	rating (ordinal)
Audited utterances	1–3	1–3	1–3
Source offers	3 named	3 named	selected grid offers
Endpoint pairs / utterance	6	6	4
P1 comparisons / utterance	12	12	8
Offer-pair queries	90	90	60
Item queries	24	27	9
Total queries	114	117	69
Queries / group Q_g	38	39	23
Utility-fit rank r_g	19	21	3
Residual df $Q_g - r_g$	19	18	20

Table 6.4: **Audit design by domain.** Each source-target pair contributes one direct offer-pair query, four path-step queries, and two P1 comparisons. Across domains there are 300 queries; 15 completions per query give 4,500 planned calls per model. Listed price is excluded from the item features and serves as the monetary numeraire. Item features construct the controlled design but are not covariates in the utility fit.

apartment source offers are `gnd_cheap_studio_walkup` ($1,400/month, 380 ft^2, 15 min commute, studio, walk-up), `gnd_mid_1br_brownstone` ($2,200/month, 720 ft^2, 30 min, 1-bedroom, brownstone), and `gnd_expensive_2br_highrise` ($3,400/month, 1,050 ft^2, 45 min, 2-bedroom, high-rise). Hotels use the Cartesian product of prices $\{\$100, \$550, \$1{,}000\}$ per night and ratings $\{0.5, 2.8, 5.0\}$ stars. Thus the hotel design contains three hotel-room items, distinguished by rating, and nine offers after crossing each room type with each listed price. All nine offers enter the audit. The released query specification records every source, target, and intermediate offer used in each comparison.

6.9.2 Audited preference utterances

Flights. 1 (price focused). "Keeping the fare low is my main priority. I'm generally willing to accept a less convenient flight if it saves me money."

2 (time focused). "Keeping the trip short is my main priority. I'm willing to pay somewhat more for a shorter flight, but price still matters."
3 (price and travel time). "I'm trying to balance price and travel time. I prefer a cheaper flight, but I would pay somewhat more for a meaningfully shorter trip."

Apartments. 1 (rent focused). "Keeping my monthly rent low is my main priority. I'm willing to compromise on space, building type, and commute to save money."
2 (space focused). "Having more space is my main priority. I'm willing to pay somewhat more for a larger apartment, while still staying within a reasonable rental budget."
3 (rent, square footage, and commute). "I'm looking for an affordable apartment, but I would pay somewhat more for additional space or a meaningfully shorter commute."

Hotels. 1 (value-seeker). "I'm looking for good value. I prefer a higher-rated hotel when the improvement seems worth the additional nightly cost."
2 (price focused). "Keeping the nightly price low is my main priority. I usually prefer the cheaper hotel, although a major difference in quality could still matter."
3 (price and rating). "I care about both nightly price and hotel quality. I would pay somewhat more for a clearly better-rated hotel, but not without limit."

6.9.3 Prompts and exact query specification

Every query uses the same canonical wording for its query type; only the preference utterance and item or offer content vary as specified above.

Requested model string	Service/API	Collected	Temp./top-p	Seed	Token limit	Reasoning/ thinking
`claude-opus-4-8`	Anthropic Messages	Jul. 27	omitted/ omitted	omitted (unsupported)	2,000 output	adaptive; no control sent
`gemini-3.5-flash`	Google Generative AI	Jul. 27	1/1	10000–10014 requested	2,000 output	no control sent
`gpt-5.5`	OpenAI Chat Completions	Jul. 27	omitted/1	10000–10014 requested	8,000 completion	no effort setting; hidden tokens share limit
`openai/gpt-oss-120b`	Groq Chat Completions	Jul. 27	1/1	10000–10014	2,000 output	no control sent
`llama-3.3-70b-versatile`	Groq Chat Completions	Jul. 28	1/1	10000–10014	2,000 output	no control sent
`qwen/qwen3.6-27b`	Groq Chat Completions	Jul. 28	1/1	10000–10014	2,000 output	effort `none`; format `hidden`

Table 6.5: **Model and API configuration.** The first column gives the exact string sent to the provider. The APIs did not return or we did not retain a separate immutable snapshot identifier, so the collection date is the available version provenance. "Omitted" means the parameter was not sent and remained at the provider default. Gemini and OpenAI seeds were requested; the APIs did not guarantee determinism. GPT-5.5's limit includes hidden reasoning tokens; other rows report output-token limits.

6.9.4 Model and API configuration

Table 6.5 records the API configuration used for the audit. The runner requested temperature 1 and seeds $10000, \ldots, 10014$ where supported. Its nominal response limit was 2,000 tokens. For GPT-5.5, the OpenAI adapter translated this to an 8,000-token completion limit so that hidden reasoning and the visible answer shared sufficient space. These limits are upper bounds and do not represent matched reasoning budgets across providers.

Other generation and analysis settings. Each request asked for one response. Stop sequences and log probabilities were disabled; frequency and presence penalties and other provider-specific generation controls were not sent and therefore remained at provider defaults. Groq calls explicitly used `stream=false`; the other clients used non-streaming calls. The global test used $B = 2000$ bootstrap draws with analysis seed 20260713. The path-length intervals used $B = 5000$ draws with seed 20260709. All reported confidence intervals are 95%, and the model-level familywise

testing level is $\alpha = 0.05$, giving the Bonferroni cutoff $0.05/9$ for nine audit groups.

Pilot development and frozen settings. The final prompts, temperature, 15-call sample size, seed schedule, model-specific reasoning settings, bootstrap sizes, and testing level were fixed before the corresponding full runs were analyzed; this was not an external preregistration. Separate pilot responses are excluded from every reported audit result. For GPT-OSS, 90-call pilots at response-token ceilings of 400, 1,200, and 2,000 produced finite-parse rates of 67.8%, 93.3%, and 100%, respectively, motivating the 2,000-token ceiling. We manually revised the original prompt into the v2 wording and compared five small offer-pair prompt variants. The final output-first signed wording was selected for unambiguous sign semantics, numerical parseability, and avoidance of nonfinite or grossly out-of-range answers relative to the stated domain price ranges, not for P1/P2 residuals or rejection rates. Qwen reasoning-mode pilots used 2,000- and 4,000-token ceilings; the reported Qwen condition was then fixed as its documented non-thinking mode with a 2,000-token ceiling. The released pilot manifests and separate result directories retain these development runs.

Computational environment. All model inference used the hosted APIs in Table 6.5; provider-side serving hardware was not exposed, and no local accelerator was used for inference. Collection and analysis were orchestrated on an Apple M5 MacBook Pro (10 CPU cores, 16 GB memory, arm64) running macOS 26.4.1 and Python 3.12.6. Relevant package versions were NumPy 2.4.1, Matplotlib 3.10.8, OpenAI 2.32.0, Anthropic 0.109.2, Groq 1.0.0, Google Gen AI 1.65.0, and Google Generative AI 0.8.6. The reported responses were collected July 27–28, 2026. The main audit contains 27,000 successful recorded completions and the path-length stress check contains 11,160, for 38,160 total; this count excludes any provider

requests retried before a response was recorded.

6.10 Supplementary nonfinite-output diagnostics

Table 6.6 treats parseability as a separate measurement requirement. All Llama 3.3, Qwen 3.6, GPT-5.5, Opus 4.8, and Gemini 3.5 Flash responses yielded finite parsed values. GPT-OSS produced one empty offer-pair response. Two Gemini responses contained a trailing underscore but were unambiguously parsed as 30; all other finite responses consisted only of a signed number, optionally formatted as currency.

Model	**Calls**	**Finite parse**	**Number-only**	$\lvert Y\rvert \geq \$10{,}000$
Claude Opus 4.8	4500	4500 (100.0%)	4500 (100.0%)	0
Gemini 3.5 Flash	4500	4500 (100.0%)	4498 (99.96%)	0
GPT-5.5	4500	4500 (100.0%)	4500 (100.0%)	0
GPT-OSS 120B	4500	4499 (99.98%)	4499 (99.98%)	0
Llama 3.3 70B	4500	4500 (100.0%)	4500 (100.0%)	0
Qwen 3.6 27B	4500	4500 (100.0%)	4500 (100.0%)	0

Table 6.6: **Numerical-output parseability.** Counts are over the same 4,500 calls per model. Finite parse records responses from which the parser extracted a finite dollar value. Number-only additionally requires the whole visible response to be a signed number, optionally formatted as currency. The final column checks for extreme parsed magnitudes; none reached \$10,000.

6.11 Supplementary path-length stress check

Figure 6.4 and Table 6.7 report the path-length stress check. The four path instances are two apartment square-footage paths and two flight travel-time paths, each under a single canonical utterance. For each model and k, the reported quantity is mean absolute P1 residual $|\widehat{R}_{1,k}|$, normalized by the fixed mean price of the endpoint offers $s_{ab} = (p + q)/2$, across the four path instances. Confidence intervals are nonparametric bootstrap intervals over path instances, not per-sample LLM-call

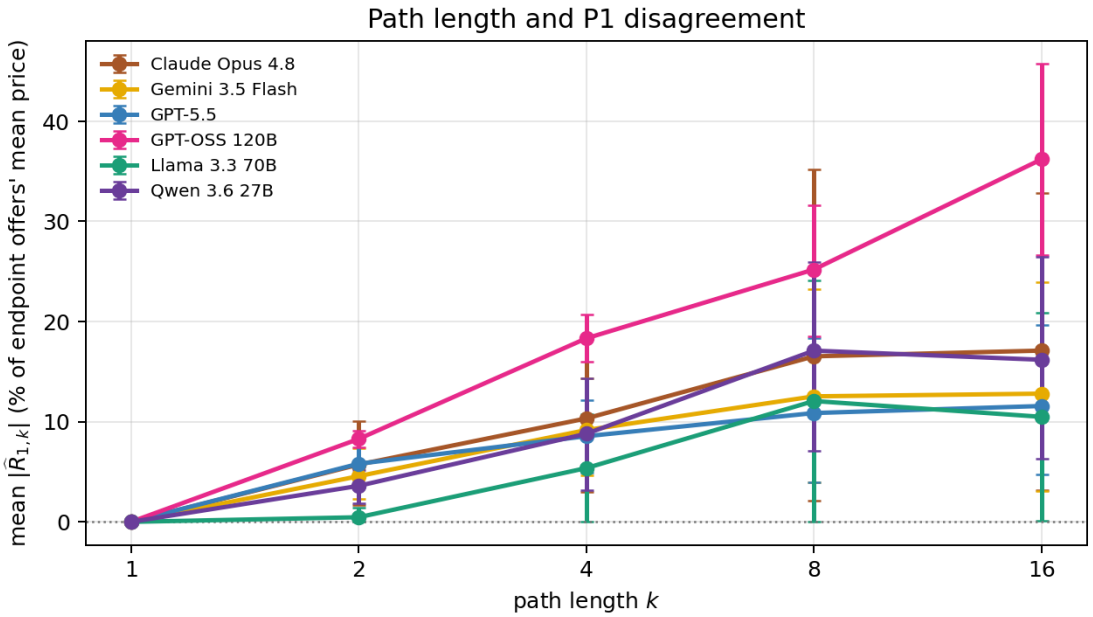


Figure 6.4: **Path length and P1 disagreement.** Each point is the mean absolute P1 residual across four prespecified smooth single-feature paths, expressed as a percentage of the endpoint offers' mean price. For every model, mean error at $k = 8$ and $k = 16$ exceeds that at $k = 2$, although the curves are not all monotone. Bars are bootstrap 95% intervals over the four paths; this stress test does not establish a general scaling law.

intervals. The v2 experiment contains 1,860 calls per model; all 11,160 responses produced finite parsed values.

6.12 P2 results by domain

Table 6.8 separates the pooled P2 results by domain. Domain counts are descriptive; the audit was not designed to estimate population differences between domains.

Model	k	n	**Mean $\lvert\widehat{R}_{1,k}\rvert$ [95% CI]**
Claude Opus 4.8	1	4	0.0 [0.0, 0.0]
Claude Opus 4.8	2	4	5.7 [1.7, 10.1]
Claude Opus 4.8	4	4	10.3 [3.0, 18.4]
Claude Opus 4.8	8	4	16.5 [2.1, 35.2]
Claude Opus 4.8	16	4	17.1 [3.2, 32.8]
Gemini 3.5 Flash	1	4	0.0 [0.0, 0.0]
Gemini 3.5 Flash	2	4	4.6 [2.3, 7.4]
Gemini 3.5 Flash	4	4	9.2 [4.7, 14.4]
Gemini 3.5 Flash	8	4	12.5 [3.9, 23.2]
Gemini 3.5 Flash	16	4	12.8 [3.1, 24.0]
GPT-5.5	1	4	0.0 [0.0, 0.0]
GPT-5.5	2	4	5.8 [3.4, 8.2]
GPT-5.5	4	4	8.6 [4.9, 12.1]
GPT-5.5	8	4	10.9 [3.9, 18.3]
GPT-5.5	16	4	11.6 [4.8, 19.6]
GPT-OSS 120B	1	4	0.0 [0.0, 0.0]
GPT-OSS 120B	2	4	8.3 [7.5, 9.1]
GPT-OSS 120B	4	4	18.3 [16.0, 20.7]
GPT-OSS 120B	8	4	25.2 [18.5, 31.6]
GPT-OSS 120B	16	4	36.2 [26.7, 45.7]
Llama 3.3 70B	1	4	0.0 [0.0, 0.0]
Llama 3.3 70B	2	4	0.5 [0.0, 1.4]
Llama 3.3 70B	4	4	5.4 [0.0, 10.7]
Llama 3.3 70B	8	4	12.1 [0.0, 24.1]
Llama 3.3 70B	16	4	10.5 [0.1, 20.9]
Qwen 3.6 27B	1	4	0.0 [0.0, 0.0]
Qwen 3.6 27B	2	4	3.6 [1.9, 5.2]
Qwen 3.6 27B	4	4	8.8 [3.1, 14.4]
Qwen 3.6 27B	8	4	17.1 [7.1, 26.0]
Qwen 3.6 27B	16	4	16.2 [6.3, 26.4]

Table 6.7: **Path-length stress-check summary.** Mean $\lvert\widehat{R}_{1,k}\rvert$ is reported as a percentage of the endpoint offers' mean price, with bootstrap 95% intervals over four path instances. The diagnostic is separate from the main audit and is meant to quantify practical dollar-scale error as local decompositions get finer. The $k = 1$ rows are the direct-comparison baseline and therefore have zero decomposition residual by construction.

Model	Domain	n	Supported reversal	Residual CI excludes 0	25%-price lower bound
Claude Opus 4.8	flights	18	5 (27.8%)	16 (88.9%)	2 (11.1%)
Claude Opus 4.8	apartments	18	1 (5.6%)	14 (77.8%)	0 (0.0%)
Claude Opus 4.8	hotels	12	0 (0.0%)	11 (91.7%)	7 (58.3%)
Gemini 3.5 Flash	flights	18	2 (11.1%)	13 (72.2%)	0 (0.0%)
Gemini 3.5 Flash	apartments	18	2 (11.1%)	14 (77.8%)	0 (0.0%)
Gemini 3.5 Flash	hotels	12	0 (0.0%)	10 (83.3%)	2 (16.7%)
GPT-5.5	flights	18	1 (5.6%)	18 (100.0%)	2 (11.1%)
GPT-5.5	apartments	18	0 (0.0%)	14 (77.8%)	1 (5.6%)
GPT-5.5	hotels	12	0 (0.0%)	8 (66.7%)	4 (33.3%)
GPT-OSS 120B	flights	18	0 (0.0%)	15 (83.3%)	0 (0.0%)
GPT-OSS 120B	apartments	18	2 (11.1%)	6 (33.3%)	2 (11.1%)
GPT-OSS 120B	hotels	12	0 (0.0%)	10 (83.3%)	7 (58.3%)
Llama 3.3 70B	flights	18	4 (22.2%)	15 (83.3%)	2 (11.1%)
Llama 3.3 70B	apartments	18	1 (5.6%)	16 (88.9%)	0 (0.0%)
Llama 3.3 70B	hotels	12	0 (0.0%)	11 (91.7%)	8 (66.7%)
Qwen 3.6 27B	flights	18	0 (0.0%)	5 (27.8%)	0 (0.0%)
Qwen 3.6 27B	apartments	18	0 (0.0%)	4 (22.2%)	0 (0.0%)
Qwen 3.6 27B	hotels	12	1 (8.3%)	11 (91.7%)	9 (75.0%)

Table 6.8: **P2 results by domain.** A supported reversal means that the offer-pair and item-query estimates have 95% intervals excluding zero in opposite directions. The final column counts comparisons whose 95% lower bound on absolute P2 disagreement is at least 25% of the two offers' mean listed price. Across models, supported reversals occur in all three domains, and every model–domain combination contains supported cardinal disagreement.

6.13 Statistical details

6.13.1 Audit-group self-consistency tests

For each model, an audit group contains all queries for one domain and one utterance. With three domains and three selected utterances per domain, there are nine audit groups. The utility is conditional on the utterance and defined over that domain's item set, so utility values are neither shared nor directly comparable across groups.

As defined in Section 6.3.2, enumerating the queries gives Y_c^i for completion i, $\overline{Y}_c = n_c^{-1} \sum_i Y_c^i$, and $m_c(U)$ for the answer implied by utility assignment U.

For audit group g, $\widehat{U}_g$ and T_g are the group-specific versions of the least-squares utility and RMSE defined in Section 6.3.5. The fit uses the observed query means $\overline{Y}(c)$, and the minimum ranges over every assignment of one utility value to each audited item. It is obtained by ordinary least squares: an item query selects one item value, while an offer-pair query takes the difference of two item values and applies a known price adjustment. This is not a linear utility model over item attributes, and T_g does not select particular P1 or P2 comparisons in advance.

We give each query mean equal weight because T_g is intended to measure the RMSE for a uniformly selected query from the prespecified set that represents one model–domain–utterance use case. Equal weighting is not generally the most efficient estimator of U. If query variances were accurately known and efficient estimation under a correctly specified model were the objective, inverse-variance weighting could be used; it would define a different best-fitting summary by giving more influence to low-variance queries. Audit groups are fit separately so that each has its own conditional utility and dollar-scale summary.

The generic notation also separates persistent disagreement from finite-sample variation. Let $U_g^\star$ be the group-specific version of the population least-squares

utility assignment in Equation 6.4 and write

$$Y_c^i = m_c(U_g^\star) + \delta_c + \varepsilon_c^i, \qquad \mathbb{E}[\varepsilon_c^i] = 0.$$

For each fixed query c, we assume $\varepsilon_c^1, \ldots, \varepsilon_c^{n_c}$ are independent and identically distributed, with finite variance $\mathrm{Var}(\varepsilon_c^i) = \sigma_c^2$. We allow σ_c^2 to differ across queries and treat calls belonging to different queries as independent. Here $\delta_c = \mathbb{E}[Y_c^i] - m_c(U_g^\star)$ is the persistent discrepancy for query c; it is the enumerated form of $\delta(c)$ from Section 6.3.3. The null hypothesis holds exactly when every δ_c is zero.

Now write

$$S_g = Q_g T_g^2 = \sum_{c=1}^{Q_g} \{\overline{Y}_c - m_c(\widehat{U}_g)\}^2.$$

If the utility values were known rather than fitted, the sampling-noise contribution to $\mathbb{E}[S_g]$ would be the sum of the query-mean variances. In fact, the utilities are fitted from these same noisy means, so the fit absorbs some sampling variation.

The corresponding linear regression is simple. Form a vector $\mathbf{y}$ by leaving each item-query mean unchanged and adding the known adjustment $(q - p)$ to each offer-pair-query mean; this known shift does not change any fitted residual or S_g. Create a design matrix X with one column per audited item. An item query about item x has a row containing 1 in the column for x and 0 elsewhere. An offer-pair query from $a = (x, p)$ to $b = (y, q)$ has -1 in the column for x, $+1$ in the column for y, and 0 elsewhere. Consequently, fitting the unrestricted item utilities is exactly

$$\widehat{\mathbf{u}} \in \arg\min_{\mathbf{u}} \|\mathbf{y} - X\mathbf{u}\|_2^2.$$

The matrix

$$H = XX^+,$$

where X^+ is the Moore–Penrose pseudoinverse, maps the adjusted query means to their fitted values: $X\widehat{\mathbf{u}} = H\mathbf{y}$. This projection matrix is commonly called the ordinary-least-squares *hat matrix*. Its diagonal entry $h_c = H_{cc}$ measures query

c's leverage: holding the other query means fixed, increasing query c's observed mean by one unit changes its own fitted mean by h_c units. The leverages satisfy $\sum_c h_c = r_g$, where r_g is the rank of the utility fit.

This notation makes explicit how much sampling variation can be absorbed by estimating the utilities from the same query means used to compute S_g. The following lemma records its finite-sample baseline.

Lemma 6.2 (Expected squared lack of fit)**.** *If query means are independent and* $\mathrm{Var}(Y_c^i) = \sigma_c^2$*, then*

$$\mathbb{E}[S_g] = \sum_{c=1}^{Q_g}(1 - h_c)\frac{\sigma_c^2}{n_c} + \sum_{c=1}^{Q_g}\delta_c^2. \tag{6.10}$$

Proof. Let $\Sigma = \mathrm{diag}(\sigma_1^2/n_1, \ldots, \sigma_{Q_g}^2/n_{Q_g})$. Because $U_g^\star$ is the least-squares population fit, $\boldsymbol{\delta}$ is orthogonal to the columns of X, so $H\boldsymbol{\delta} = 0$. Hence

$$S_g = \|\boldsymbol{\delta} + (I - H)\overline{\boldsymbol{\varepsilon}}\|_2^2.$$

Taking expectations eliminates the cross term and gives

$$\mathbb{E}[S_g] = \|\boldsymbol{\delta}\|_2^2 + \mathrm{tr}\{(I - H)\Sigma(I - H)\}.$$

Since $I - H$ is symmetric and idempotent, the trace term equals $\mathrm{tr}\{(I - H)\Sigma\} = \sum_c(1 - h_c)\sigma_c^2/n_c$, proving Equation 6.10. Under self-consistency, the expected squared statistic therefore contains only the sampling variation left after fitting U. Under the alternative, the squared discrepancies that no utility assignment can absorb add directly to that expectation. Increasing the number of completions reduces the first term but not the second. We include this calculation to interpret the statistic's finite-sample baseline; the bootstrap test does not estimate a common variance or use this formula as its reference distribution. Refitting the utility in every bootstrap draw automatically reproduces the sampling variation absorbed by the fit.

For intuition, if all queries have n completions with common response variance σ^2, and the utility fit has rank r_g, then

$$\mathbb{E}[T_g^2] = \frac{Q_g - r_g}{Q_g}\frac{\sigma^2}{n} + \frac{1}{Q_g}\sum_{c=1}^{Q_g}\delta_c^2.$$

The expectation is stated for T_g^2 because the expectation of the square root T_g has no equally simple form. These identities require finite variances and independent query means, but not equal variances or Gaussian responses. We do not pool within-query variances into one common noise estimate because the response variance may differ across queries. Instead, the querywise bootstrap below preserves each query's empirical response distribution under the fitted null.

We estimate the reference distribution of T_g under the fitted null with a within-query nonparametric bootstrap (Efron and Tibshirani, 1993). We use this procedure because numerical LLM responses may be non-Gaussian and because their variances may differ across queries.

Observed-data stage. Using all original, successfully parsed completions, compute each query mean $\overline{Y}_c$, fit $\widehat{U}_g$ with Equation 6.6, and compute the observed RMSE T_g with Equation 6.7. For each query, also form a fixed resampling pool by subtracting $\overline{Y}_c$ from each of its original completions. These centered values retain the query's observed response variability but have mean zero. Thus, the observed RMSE measures disagreement among the original query means after fitting $\widehat{U}_g$; it is not error relative to a person's true utility or prediction error for individual completions.

Bootstrap stage. For each bootstrap draw, independently resample with replacement from each query's fixed centered pool, using that query's original number of successfully parsed completions. Add the resampled values to $m_c(\widehat{U}_g)$, the answer

implied by the utility fitted to the original data. This produces a synthetic dataset whose query means satisfy the fitted self-consistency null apart from resampled finite-sample variation. Using only that synthetic dataset, refit the utility and recompute the RMSE, denoted $T_g^{*(b)}$. Refitting is necessary because the observed statistic also estimates the utility from the data and therefore absorbs some sampling variation. The original $\widehat{U}_g$ and centered pools define the null-generating distribution; the synthetic query means, fitted utility, and RMSE are new in every draw.

With $B = 2000$ bootstrap draws, the audit-group p-value is

$$\frac{1 + \sum_{b=1}^{B} \mathbb{I}\{T_g^{*(b)} \geq T_g\}}{B + 1},$$

where $T_g^{*(b)}$ is the refitted RMSE in bootstrap draw b and $\mathbb{I}\{\cdot\}$ is the indicator function. Thus, the p-value is the fraction of RMSE values generated under the fitted self-consistency null that are at least as large as the observed RMSE, with the standard finite-bootstrap correction.

Our model-level claim ranges over nine audit groups: all nine are self-consistent. Because rejecting any one group contradicts that claim, examining nine unadjusted 5% tests would increase the chance of a false positive. We therefore use a Bonferroni correction at $\alpha = 0.05$, rejecting a group only when $p \leq 0.05/9 = 0.0056$ (Dunn, 1961). The union bound guarantees that the probability of falsely rejecting any self-consistent group is at most 5%, without requiring independence. At least one rejection rejects the joint claim that all nine groups are self-consistent; the count describes the breadth of the evidence. An audit with one prespecified group needs no correction. Table 6.9 reports the unadjusted group p-values for transparency.

Model	Domain	Utterance	RMSE ($)	RMSE (%)	Boot. p
Claude Opus 4.8	apartments	1	28.2	1.2	< 0.001
Claude Opus 4.8	apartments	2	52.3	2.2	< 0.001
Claude Opus 4.8	apartments	3	34.5	1.5	< 0.001
Claude Opus 4.8	flights	1	10.7	2.9	< 0.001
Claude Opus 4.8	flights	2	26.5	7.2	< 0.001
Claude Opus 4.8	flights	3	13.5	3.7	< 0.001
Claude Opus 4.8	hotels	1	119.4	21.7	< 0.001
Claude Opus 4.8	hotels	2	90.7	16.5	< 0.001
Claude Opus 4.8	hotels	3	117.6	21.4	< 0.001
Gemini 3.5 Flash	apartments	1	48.0	2.1	< 0.001
Gemini 3.5 Flash	apartments	2	50.2	2.2	< 0.001
Gemini 3.5 Flash	apartments	3	44.2	1.9	< 0.001
Gemini 3.5 Flash	flights	1	4.4	1.2	< 0.001
Gemini 3.5 Flash	flights	2	10.0	2.7	< 0.001
Gemini 3.5 Flash	flights	3	8.6	2.3	< 0.001
Gemini 3.5 Flash	hotels	1	127.1	23.1	< 0.001
Gemini 3.5 Flash	hotels	2	74.6	13.6	< 0.001
Gemini 3.5 Flash	hotels	3	110.4	20.1	< 0.001
GPT-5.5	apartments	1	35.0	1.5	< 0.001
GPT-5.5	apartments	2	78.4	3.4	< 0.001
GPT-5.5	apartments	3	41.7	1.8	< 0.001
GPT-5.5	flights	1	8.2	2.2	< 0.001
GPT-5.5	flights	2	22.7	6.2	< 0.001
GPT-5.5	flights	3	13.0	3.5	< 0.001
GPT-5.5	hotels	1	129.9	23.6	< 0.001
GPT-5.5	hotels	2	83.3	15.1	< 0.001
GPT-5.5	hotels	3	103.9	18.9	< 0.001

Table 6.9: **Audit-group self-consistency tests.** Each row fits one free utility value per audited item for a fixed model, domain, and utterance; numbers 1–3 refer to the domain-specific utterances in Appendix 6.9.2. RMSE is the smallest root mean squared error. The percentage uses the mean of endpoint listed prices across that group's offer-pair queries, weighted as they appear in the design; bootstrap p-values are unadjusted. The main text reports the number of audit groups rejected after applying Bonferroni correction.

6.13.2 Local residual uncertainty

This appendix documents the uncertainty calculations behind the local rejections in Table 6.3 and the lower-bound curves in Figure 6.3.

For query c, let n_c be the realized number of finite completions and let s_c be their sample standard deviation. The query average is $\overline{Y}(c)$, with standard error $s_c/\sqrt{n_c}$. Queries with no finite response are excluded from residual analyses and counted in Table 6.6.

Model	Domain	Utterance	RMSE ($)	RMSE (%)	Boot. p
GPT-OSS 120B	apartments	1	40.3	1.7	< 0.001
GPT-OSS 120B	apartments	2	111.5	4.8	< 0.001
GPT-OSS 120B	apartments	3	69.7	3.0	< 0.001
GPT-OSS 120B	flights	1	9.5	2.6	< 0.001
GPT-OSS 120B	flights	2	12.1	3.3	< 0.001
GPT-OSS 120B	flights	3	11.1	3.0	< 0.001
GPT-OSS 120B	hotels	1	128.6	23.4	0.0020
GPT-OSS 120B	hotels	2	104.3	19.0	< 0.001
GPT-OSS 120B	hotels	3	149.9	27.3	< 0.001
Llama 3.3 70B	apartments	1	139.8	6.0	< 0.001
Llama 3.3 70B	apartments	2	116.6	5.0	< 0.001
Llama 3.3 70B	apartments	3	170.6	7.3	< 0.001
Llama 3.3 70B	flights	1	18.3	5.0	< 0.001
Llama 3.3 70B	flights	2	17.1	4.7	< 0.001
Llama 3.3 70B	flights	3	18.8	5.1	< 0.001
Llama 3.3 70B	hotels	1	200.1	36.4	< 0.001
Llama 3.3 70B	hotels	2	197.3	35.9	< 0.001
Llama 3.3 70B	hotels	3	196.6	35.8	< 0.001
Qwen 3.6 27B	apartments	1	67.3	2.9	0.379
Qwen 3.6 27B	apartments	2	75.4	3.2	0.025
Qwen 3.6 27B	apartments	3	53.1	2.3	0.171
Qwen 3.6 27B	flights	1	21.1	5.8	0.010
Qwen 3.6 27B	flights	2	15.1	4.1	0.058
Qwen 3.6 27B	flights	3	19.1	5.2	< 0.001
Qwen 3.6 27B	hotels	1	276.5	50.3	0.0040
Qwen 3.6 27B	hotels	2	168.4	30.6	< 0.001
Qwen 3.6 27B	hotels	3	296.5	53.9	< 0.001

Table 6.10: **Audit-group self-consistency tests (continued).** Columns and reporting conventions are as in Table 6.9.

Each P1 or P2 residual is a signed sum of three query averages. Its constituent queries use disjoint LLM calls. Indexing those three queries by $j = 1, 2, 3$, we estimate

$$\widehat{\mathrm{SE}}(\widehat{R}) = \left\{ \sum_{j=1}^{3} \frac{s_j^2}{n_j} \right\}^{1/2}.$$

The signs do not affect this variance calculation. Listed-price adjustments are fixed and therefore add no sampling variance.

The threshold curves divide $|\widehat{R}|$ by the fixed mean-price scale $s_{ab} = (p + q)/2$. Consequently, $\mathrm{SE}(\widehat{R}/s_{ab}) = \mathrm{SE}(\widehat{R})/s_{ab}$. Reported 95% intervals are normal-approximation (Wald) intervals $\widehat{R} \pm 1.96\,\widehat{\mathrm{SE}}(\widehat{R})$. A local rejection has an interval excluding zero. Dashed curves plot the comparison-specific nonnegative magnitude

lower bounds

$$\frac{\max\{|\widehat{R}| - 1.96\,\widehat{\mathrm{SE}}(\widehat{R}), 0\}}{s_{ab}},$$

and are not simultaneous confidence bands.

Independence assumptions. Repeated completions within each fixed model–query pair are treated as independent and identically distributed; variances may differ across queries. We use temperature 1 and distinct seeds where available; the Opus 4.8 interface exposes neither. Different queries use separate prompts and calls. Two paths sharing the same source-target pair (a, b) share the direct $\overline{Y}((a, b))$ estimate, inducing positive correlation between their residuals; we do not model this dependence in the empirical threshold curves (they remain valid as empirical distributions of comparison-level residuals, but the points are not independent and identically distributed).

BIBLIOGRAPHY

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. `https://www.tensorflow.org/`, accessed 28$^{\text{th}}$ April 2022.

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models, 2024. URL `https://arxiv.org/abs/2402.10172`.

S. Asmussen and P. W. Glynn. *Stochastic Simulation: Algorithms and Analysis*, volume 57 of *Stochastic Modeling and Applied Probability*. Springer, New York, 2007.

David Austin, Anton Korikov, Armin Toroghi, and Scott Sanner. Bayesian optimization with LLM-based acquisition functions for natural language preference elicitation. In *18th ACM Conference on Recommender Systems*, RecSys '24, pages 74–83. ACM, October 2024. doi: 10.1145/3640457.3688142. URL `http://dx.doi.org/10.1145/3640457.3688142`.

John Aycock. A brief history of just-in-time. *ACM Computing Surveys*, 35(2): 97–113, 2003. ISSN 0360-0300. doi: 10.1145/857076.857077. URL `https://doi-org.proxy.library.cornell.edu/10.1145/857076.857077`.

Yuntao Bai et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Eric B Baum and Kenneth Lang. Query learning can work poorly when a human oracle is used. In *International joint conference on neural networks*, volume 8, page 8. Beijing China, 1992.

Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(1):5595–5637, 2017. ISSN 1532-4435.

Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. doi: 10.1007/BF01386316. URL `https://doi.org/10.1007/BF01386316`.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. 2018. URL `http://github.com/google/jax`. `http://github.com/google/jax`, Version 0.2.5, accessed 28$^{\text{th}}$ April 2022.

Bernd Bullnheimer. An examination scheduling model to maximize students' study time. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pages 78–91, Berlin, Heidelberg, 1998. Springer. ISBN 978-3-540-49803-2. doi: 10.1007/BFb0055882. URL `https://doi.org/10.1007/BFb0055882`.

Edmund K. Burke, Jakub Marecek, Andrew J. Parkes, and Hana Rudova. Decomposition, reformulation, and diving in university course timetabling. *Computers*

& Operations Research, 37(3):582–597, 2010. doi: 10.1016/j.cor.2009.02.023. URL https://doi.org/10.1016/j.cor.2009.02.023.

Xi-Ren Cao. Convergence of parameter sensitivity estimates in a stochastic experiment. *IEEE Transactions on Automatic Control*, 30(9):845–853, Sep. 1985. ISSN 1558-2523. doi: 10.1109/TAC.1985.1104076.

Michael W. Carter. OR practice—a survey of practical applications of examination timetabling algorithms. *Operations Research*, 34(2):193–202, March 1986. doi: 10.1287/opre.34.2.193. URL https://doi.org/10.1287/opre.34.2.193.

Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. Educational timetabling: Problems, benchmarks, and state-of-the-art results. *European Journal of Operational Research*, 308(1):1–18, 2023. doi: 10.1016/j.ejor.2022.07.011. URL https://doi.org/10.1016/j.ejor.2022.07.011.

Clara Chaplin, Stanley Gai, Samuel C. Gutekunst, Tsugunobu Miyake, Thiago Serra, Luke Snyder, Vy Tran, and Lucas Waddell. Final exam scheduling at Bucknell University: A case study and open-source tool. *INFORMS Journal on Applied Analytics*, 56(4):383–399, 2026. doi: 10.1287/inte.2025.0262. URL https://doi.org/10.1287/inte.2025.0262.

Jialei Chen, Yujia Xie, Kan Wang, Chuck Zhang, Mani A. Vannan, Ben Wang, and Zhen Qian. Active image synthesis for efficient labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3770–3781, nov 2021. doi: 10.1109/tpami.2020.2993221. URL https://doi.org/10.1109%2Ftpami.2020.2993221.

Zhenyu Cui, Michael C Fu, Jian-Qiang Hu, Yanchu Liu, Yijie Peng, and Lingjiong

Zhu. On the variance of single-run unbiased stochastic derivative estimators. *INFORMS Journal on Computing*, 32(2):390–407, 2020.

George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960. doi: 10.1287/opre.8.1.101. URL `https://doi.org/10.1287/opre.8.1.101`.

Gerard Debreu. Representation of a preference ordering by a numerical function. *Decision Processes*, pages 159–165, 1954.

Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

D. J. Eckman, S. G. Henderson, R. Pasupathy, and S. Shashaani. Simulation optimization library. 2020. `http://github.com/simopt-admin/simopt`, accessed 28th April 2022.

David J. Eckman and Shane G. Henderson. Biased gradient estimators in simulation optimization. In *2020 Winter Simulation Conference (WSC)*, pages 2935–2946, 2020a. doi: 10.1109/WSC48552.2020.9383938.

David J. Eckman and Shane G. Henderson. Biased gradient estimators in simulation optimization. In K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, editors, *Proceedings of the 2020 Winter Simulation Conference*, pages 2935–2946, Piscataway, New Jersey, 2020b. Institute of Electrical and Electronics Engineers, Inc.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, New York, 1993.

Michael Eichelbeck, Tim Voigt, and Matthias Althoff. Supporting high-stakes decision making through interactive preference elicitation in the latent space.

In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ra7CSHcVCv.

Yuanning Feng, Sinan Wang, Zhengxiang Cheng, Yao Wan, and Dongping Chen. Are we on the right way to assessing LLM-as-a-judge?, 2026. URL https://arxiv.org/abs/2512.16041.

Peter C. Fishburn. Nontransitive preferences in decision theory. *Journal of Risk and Uncertainty*, 4(2):113–134, 1991.

Matthew T. Ford, Shane G. Henderson, and David J. Eckman. Automatic differentiation for gradient estimators in simulation. In *2022 Winter Simulation Conference (WSC)*, pages 3134–3145. IEEE, 2022.

Adam Foster, Martin Jankowiak, Matthew O'Meara, Yee Whye Teh, and Tom Rainforth. A unified stochastic gradient approach to designing Bayesian-optimal experiments, 2020.

Christian Fries. Fast stochastic forward sensitivities in monte-carlo simulations using stochastic automatic differentiation (with applications to initial margin valuation adjustments (mva)). *SSRN Electronic Journal*, 01 2017. doi: 10.2139/ssrn.3018165.

Christian Fries. Stochastic automatic differentiation: automatic differentiation for monte-carlo simulations. *Quantitative Finance*, 19:1–17, 01 2019. doi: 10.1080/14697688.2018.1556398.

M. C. Fu and J.-Q. Hu. *Conditional Monte Carlo: Gradient Estimation and Optimization Applications*. Kluwer, Boston, 1997.

M.C. Fu, S. Andradottir, J.S. Carson, F. Glover, C.R. Harrell, Yu-Chi Ho, J.P. Kelly, and S.M. Robinson. Integrating optimization and simulation: Research

and practice. In J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, editors, *Proceedings of the 2020 Winter Simulation Conference*, pages 610–616, Piscataway, New Jersey, 2000. Institute of Electrical and Electronics Engineers, Inc. doi: 10.1109/WSC.2000.899770.

Michael C. Fu. Stochastic gradient estimation. In Michael C. Fu, editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, chapter 5, pages 105–148. Springer New York, 2015.

Avery Funkhouser and Anna V. Blenda. Medical student preferences on examination scheduling at a US allopathic medical school—survey study. *Advances in Medical Education and Practice*, 15:945–955, 2024. doi: 10.2147/AMEP.S459701. URL `https://doi.org/10.2147/AMEP.S459701`.

Andrew Gelman and Donald B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992. doi: 10.1214/ss/1177011136.

Mike Giles and Paul Glasserman. Smoking adjoints: Fast Monte Carlo greeks. *Risk*, 19(1):88–92, 2006.

P. Glasserman. *Gradient Estimation Via Perturbation Analysis*. Kluwer, The Netherlands, 1991.

Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer, 2004.

Paul Glasserman. Adjoints and averaging. In *Stochastic Simulation Optimization for Discrete Event Systems: Perturbation Analysis, Ordinal Optimization, and Beyond*, pages 63–73. World Scientific, Singapore, 2013.

P. W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

Wei-Bo Gong and Yu-Chi Ho. Smoothed (conditional) perturbation analysis of discrete event dynamical systems. *IEEE Transactions on Automatic Control*, 32 (10):858–866, 1987.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Andreas Griewank. On automatic differentiation. In Masao Iri and Kunio Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108, Dordrecht, The Netherlands, 1989. Kluwer Academic Publishers.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, 2008. ISBN 978-0-89871-659-7. doi: 10.1137/1.9780898717761. URL `https://doi.org/10.1137/1.9780898717761`.

Aniket Haldar and Julia Hockenmaier. Rating roulette: Self-inconsistency in LLM-as-a-judge frameworks. In *Findings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. URL `https://aclanthology.org/2025.findings-emnlp.1361/`.

Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(47):1593–1623, 2014.

Miriam W. Huijser and Jan C. van Gemert. Active decision boundary annotation with deep generative models, 2017.

Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B Shah, and Will Tebbutt. A differentiable programming system to bridge machine learning and scientific computing. 2019. doi: 10.48550/ARXIV.1907.07587. https://arxiv.org/abs/1907.07587, accessed 28th April 2022.

Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial Hodge theory. *Mathematical Programming*, 127(1):203–244, 2011. doi: 10.1007/s10107-010-0419-x.

Adam S. Jovine, Tinghan Ye, Francis Bahk, Jingjing Wang, David B. Shmoys, and Peter I. Frazier. LISTEN to your preferences: An LLM framework for multi-objective selection, 2025. URL https://arxiv.org/abs/2510.25799.

Adam Shafik Jovine. SchedulingBenchmark. GitHub repository, 2025. URL https://github.com/AdamJovine/SchedulingBenchmark. Commit 16a0a586b7b1204a570d24dbbb6c3ce6bd95c6fc; accessed July 13, 2026.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models, 2023. URL https://arxiv.org/abs/2310.08491.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671. URL https://doi.org/10.1126/science.220.4598.671.

Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. BatchBALD: Efficient

and diverse batch acquisition for deep Bayesian active learning, 2019. URL https://arxiv.org/abs/1906.08158.

Katarzyna Kobalczyk, Zhiyuan Jerry Lin, Benjamin Letham, Zhuokai Zhao, Maximilian Balandat, and Eytan Bakshy. LILO: Bayesian optimization with interactive natural language feedback, 2025. URL https://arxiv.org/abs/2510.17671.

Quan Kong, Bin Tong, Martin Klinkigt, Yuki Watanabe, Naoto Akira, and Tomokazu Murakami. Active generative adversarial network for image classification, 2019.

David M. Kreps. *Notes on the Theory of Choice*. Underground Classics in Economics. Westview Press, 1988.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18(1):430–474, 2017. ISSN 1532-4435.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating reward models for language modeling, 2024. URL https://arxiv.org/abs/2403.13787.

Connor Lawless, Jakob Schoeffer, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahareh Sarrafzadeh. "I Want It That Way": Enabling interactive decision support using large language models and constraint programming, 2023. URL https://arxiv.org/abs/2312.06908.

Yinhong Liu, Zhijiang Guo, Tianya Liang, Ehsan Shareghi, Ivan Vulić, and Nigel Collier. Aligning with logic: Measuring, evaluating and improving

logical preference consistency in large language models, 2024. URL https://arxiv.org/abs/2410.02205.

R. Duncan Luce. Semiorders and a theory of utility discrimination. *Econometrica*, 24(2):178–191, 1956.

R. Duncan Luce. *Individual Choice Behavior: A Theoretical Analysis*. John Wiley and Sons, New York, 1959.

Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, 2015. URL /bib/maclaurin/maclaurinautograd/automl-short.pdf,https://indico.lal.in2p3.fr/event/2914/session/1/contribution/6/3/material/paper/0.pdf,https://github.com/HIPS/autograd. July 11th, Lille, France.

Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

William Moses and Valentin Churavy. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33*, pages 12472–12485, Red Hook, New York, 2020. Curran Associates, Inc.

Ahmad Muklason, Andrew J. Parkes, Ender Özcan, Barry McCollum, and Paul McMullan. Fairness in examination timetabling: Student preferences and extended formulations. *Applied Soft Computing*, 55:302–318, 2017. ISSN 1568-4946. doi: 10.1016/j.asoc.2017.01.026. URL https://doi.org/10.1016/j.asoc.2017.01.026.

Uwe Naumann. Optimal jacobian accumulation is np-complete. *Mathematical Programming*, 112(2):427–441, Apr 2008. ISSN 1436-4646. doi: 10.1007/s10107-006-0042-z. URL https://doi.org/10.1007/s10107-006-0042-z.

George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988. ISBN 978-0-471-82819-8. doi: 10.1002/9781118627372. URL https://doi.org/10.1002/9781118627372.

Ian Osband, Zheng Wen, Mohammad Asghari, Morteza Ibrahimi, Xiyuan Lu, and Benjamin Van Roy. Epistemic neural networks. *CoRR*, abs/2107.08924, 2021. URL https://arxiv.org/abs/2107.08924.

Omiros Papaspiliopoulos, Gareth O. Roberts, and Martin Skold. A general framework for the parametrization of hierarchical models. *Statistical Science*, 22(1): 59–73, 2007. doi: 10.1214/088342307000000014.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv preprint arXiv:1912.11554*, 2019.

R. L. Plackett. The analysis of permutations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 24(2):193–202, 1975. doi: 10.2307/2346567.

Rong Qu, Edmund K. Burke, Barry McCollum, Liam T. G. Merlot, and Sanja Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):55–89, 2009. doi: 10.1007/s10951-008-0077-5. URL https://doi.org/10.1007/s10951-008-0077-5.

Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions, 2023. URL https://arxiv.org/abs/2303.08233.

Michel Regenwetter, Jason Dana, and Clintin P. Davis-Stober. Transitivity of preferences. *Psychological Review*, 118(1):42–56, 2011.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning, 2021.

Manon Reusens, Sofie Goethals, Toon Calders, and David Martens. Would a large language model pay extra for a view? Inferring willingness to pay from subjective choices, 2026. URL https://arxiv.org/abs/2602.09802.

Tim Roughgarden. *Twenty Lectures on Algorithmic Game Theory*. Cambridge University Press, Cambridge, UK, 2016.

Manuela Sanguinetti, Alessandra Perniciano, Luca Zedda, Andrea Loddo, Cecilia Di Ruberto, and Maurizio Atzori. From user preferences to optimization constraints using large language models, 2025. URL https://arxiv.org/abs/2503.21360.

Raphael Schumann and Ines Rehbein. Active learning via membership query synthesis for semi-supervised sentence classification. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 472–481, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/K19-1044. URL https://aclanthology.org/K19-1044.

Steven G. Self and Kung-Yee Liang. Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions. *Journal*

of the American Statistical Association, 82(398):605–610, 1987. doi: 10.1080/01621459.1987.10478472.

Ei Shwe Sin and Nang Saing Moon Kham. Hyper heuristic based on great deluge and its variants for exam timetabling problem. *International Journal of Artificial Intelligence and Applications*, 3(1), 2012. doi: 10.5121/ijaia.2012.3112. URL `https://doi.org/10.5121/ijaia.2012.3112`.

Kenneth E. Train. *Discrete Choice Methods with Simulation*. Cambridge University Press, 2 edition, 2009.

Toan Tran, Thanh-Toan Do, Ian Reid, and Gustavo Carneiro. Bayesian generative active deep learning, 2019.

Amos Tversky. Intransitivity of preferences. *Psychological Review*, 76(1):31–48, 1969. doi: 10.1037/h0026750.

Bart van Merriënboer, Olivier Breuleux, Arnaud Bergeron, and Pascal Lamblin. Automatic differentiation in ml: Where we are and where we should be going. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, and Nicolò Cesa-Bianchi, editors, *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8771–8781, Red Hook, New York, 2018. Curran Associates, Inc.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL `https://aclanthology.org/2024.acl-long.511/`.

Wikipedia contributors. Automatic differentiation — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/wiki/Automatic_differentiation. "https://en.wikipedia.org/wiki/Automatic_differentiation", accessed 28th April 2022.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Tinghan Ye, Adam S. Jovine, Willem van Osselaer, Qihan Zhu, and David B. Shmoys. Cornell University uses integer programming to optimize final exam scheduling. *arXiv preprint arXiv:2409.04959*, 2024. URL https://arxiv.org/abs/2409.04959.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.