

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 NLMOPTIMIZER: A NEUROSYMBOLIC FRAMEWORK AND BENCHMARK FOR OPERATIONS RESEARCH OPTI- MIZATION PROBLEMS FROM NATURAL LANGUAGE

006  
007     **Anonymous authors**  
008  
009  
010  
011  
012

Paper under double-blind review

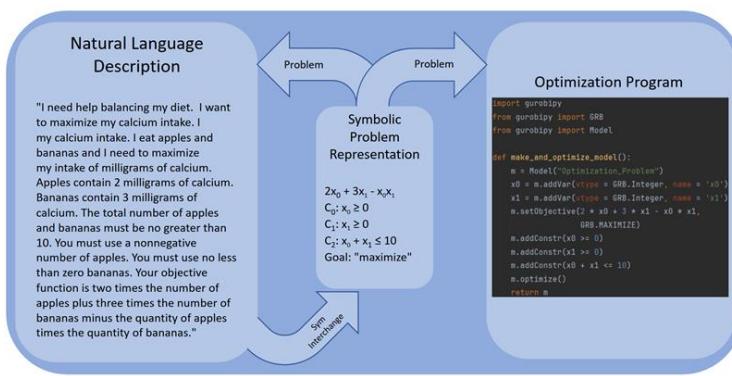
## ABSTRACT

013     Large Language Models (LLMs) are increasingly applied to structured reasoning  
014     tasks, but remain prone to generating outputs that are both syntactically coherent  
015     and semantically invalid, posing a serious challenge for the domain of mathemati-  
016     cal optimization. In particular, applications to operations research (OR) problems,  
017     where problem descriptions are often ambiguous, context-rich, and semantically  
018     dense, are compromised by these issues and a dearth of publicly available datasets  
019     appropriately designed for both training and benchmarking model performance.  
020     In this paper, we address these issues by first introducing **NLMOptimizer**, a neu-  
021     rosymbolic framework built on two classes: (i) the **Problem** class, which sys-  
022     tematically generates optimization problems; and (ii) the **SymInterchange** class,  
023     an exploratory suite of neurosymbolic methods intended to map word problems  
024     into structured, solver-executable forms. We then address the dearth of plausi-  
025     bly complex OR problems with the associated NLMOptimizer dataset, generated  
026     using **Problem**, which pairs structured natural-language descriptions with solver-  
027     checked mathematical programs across 1000 different linear (LP) and quadratic  
028     programs (QP) across integer, mixed-integer, and continuous types. We eval-  
029     uate four instruction-tuned LLMs (LLaMa-3.3, LLaMa-4-Scout, Gemini-1.5-Pro,  
030     GPT-OSS-120B) under zero-shot prompting and observe substantial degradation  
031     on our dataset, with the strongest model dropping from 66.6% end-to-end accu-  
032     racy on the NL4OPT benchmark dataset to 14.6% on NLMOptimizer. Our results  
033     indicate that (i) widely used benchmarks understate the difficulty of mapping nat-  
034     ural language to formal OR optimization problem structure, (ii) current LLMs  
035     struggle to represent even modestly more complex OR optimization problems than  
036     LPs with three variables, and (iii) progress will require methods that directly target  
037     representational fidelity without training models to fit fixed examples.

## 1 INTRODUCTION

040     Despite remarkable advances in large language models (LLMs) and neural reasoning systems, to-  
041     day’s artificial intelligence (AI) still struggles to integrate the depth and rigor of symbolic math-  
042     ematics with the flexibility of natural language understanding. This gap is especially salient in  
043     mathematical optimization, where problem descriptions are often ambiguous, context-rich, and se-  
044     mantically dense, yet require precise formalization for solver execution. Operations research (OR),  
045     a branch of applied mathematics central to decision making in domains such as health care, logis-  
046     tics, and network security, exemplifies this challenge. In particular, optimization requires a faithful  
047     *problem representation* whose entities, relations, and algebraic constraints can be manipulated reli-  
048     ably. The Natural Language for Optimization (NL4OPT) challenge (Ramamonjison et al., 2022a),  
049     which seeks to translate text descriptions of OR problems into mathematical formulations, is thus  
050     both a high-stakes application and a difficult representation learning task. Recent systems such as  
051     **OptGen** (Ramamonjison et al., 2022), **OptiMUS** (AhmadiTeshnizi et al., 2024), and **ComplexOR**  
052     (Xiao et al., 2024) have demonstrated progress using intermediate representations, chain-of-thought  
053     prompting, and modular task decomposition. However, they remain fundamentally limited in their  
054     ability to ensure semantic alignment, manipulate symbolic structures, and scale to solver-executable  
055     outputs. Further, the dominant NL4OPT benchmark (Ramamonjison et al., 2022a) is restricted to

054  
055  
056  
057  
058  
059  
060  
061  
062  
063  
064  
065  
066



067  
068  
069

Figure 1: Design pattern of **Problem** and **SymInterchange** classes.

070  
071

a small set of simple examples, failing to capture the diversity and structural richness of real-world OR problems (Xiao et al., 2024).

072  
073  
074  
075  
076  
077  
078  
079  
080  
081

Most recently, Mostajabdev et al. (2025) introduced **ORQA**, a hand-crafted benchmark that asks models to identify objectives, constraints, entities, and interrelations from realistic textual cases. Despite strong instruction-tuned models, accuracies plateau well below expert performance and degrade when reasoning is elicited by chain-of-thought prompts.<sup>1</sup> We argue that such failures are not idiosyncrasies of prompting, but symptoms of a deeper representational deficit and highlight the need for neurosymbolic approaches that treat representation as first-class: not only recognizing patterns in text, but also learning to map across natural language, symbolic abstractions, and solver-ready code in a way that is precise, generalizable, and transparent. We argue that many practical optimization problems admit a compact description as semi-algebraic sets and exponential-polynomial expressions over *E-rings*, and that the proper challenge is reconstructing this from natural language descriptions.

082  
083  
084  
085

We propose **NLMOptimizer**, a neurosymbolic framework that couples a scalable realistic OR optimization problem generator with a suite of methods to share and conform lightweight IR into executable code. Our contributions in this paper consist of:

086  
087  
088  
089  
090  
091

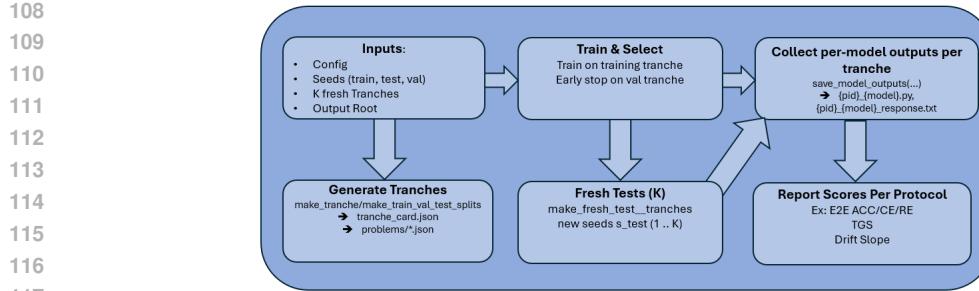
1. **Problem generation at scale.** We introduce the **Problem** class, which generates optimization problems, symbolically represented as semi-algebraic varieties defined over exponential rings  $(\mathbb{R}[x_1, \dots, x_n])^{\text{exp}}$  (Van Den Dries, 1984), and instantiate an initial dataset of 1000 linear and quadratic programs to enable head-to-head comparability with NL4OPT while increasing structural realism by including multiple resources types and larger variable counts.
2. **Empirical evaluation.** We benchmark four LLMs on both NL4OPT and 1000 **Problem**-generated examples. As with Mostajabdev et al. (2025), we observe that all models collapse under zero-shot prompting when faced with more realistic problems (e.g., continuous linear programs with 5+ variables), with Gemini-1.5-Pro End-to-End accuracy dropping from 66.6% (on NL4OPT) to 14.6% (on NLMOptimizer problems).

092  
093  
094  
095  
096

Together, these results demonstrate that current benchmarks dramatically underestimate task difficulty, that existing LLMs lack robust representations for optimization, and that principled symbolic interfaces are essential for advancing neurosymbolic reasoning and representation. While the total representational framework presented by **Problem** is scoped beyond polynomial terms, we restrict our empirical evaluation in this paper to LP and QP problems for its first release. This choice isolates and highlights representational difficulties without confounding it further by the introduction on non-standard polynomial objectives and constraints. This directly enables like-for-like comparison with NL4OPT, and avoids being able to dismiss observed degradations in performance due to significantly more challenging problems, as these degradations appear already at linear and quadratic cases with realistic scale and structure.

107

<sup>1</sup>See their Table 2 and human baseline, where the largest open model achieving 0.772 accuracy and a PhD baseline at 0.93, while CoT degrades with additional prompting.

Figure 2: **NLMOptimizer** Protocol With **Problem** For Dynamic Tranche Evaluation

## 2 RELATED WORK AND PRELIMINARIES

We assess that fully automated end-to-end processes without human-in-the-loop interventions risk incorrect outputs, particularly for more complex, domain-specific real-world use cases. To motivate our proposed framework, we first review several LLM-based efforts and their related datasets that pursued complementary approaches to natural language modeling of optimization problems.

### 2.1 PRIOR BENCHMARKING EFFORTS FOR NL-BASED OPTIMIZATION MODELING

Developed for the Natural Language for Optimization competition, the NL4OPT dataset was developed to increase the accessibility and usability of optimization solvers by allowing general users to interface with them through natural language descriptions (Ramamujison et al., 2022; Ramamujison et al., 2022b), and it has served as the benchmark dataset for the NL4OPT task. This dataset presents a total of 1101 annotated Linear Programming (LP) word problems across 6 different domains created using the Prodigy tool to manually create and annotate 600 problems, with the remaining 501 samples and annotations were generated using a NER model. **OptiMUS** (AhmadiTeshnizi et al., 2024) is a natural language (NL) to optimization code generator that employs LLM-based agents in a multi-step process including problem specification, variable and formulae generation, code generation, and debugging. The authors evaluate their approach over both existing and new benchmark datasets, and demonstrate improvement over existing NL-based solver code generators. **ComplexOR** (Xiao et al., 2024) uses a Chain-of-Experts approach with multiple experts (or agents) with individual tasks and goals. These experts include a conductor, terminology interpreter, modeling expert, programmer, and evaluator. Work is orchestrated via conductor, and tasks pass bidirectionally between experts based on feedback from the modeling expert and evaluator agents. In addition to this novel approach, the authors also present a more challenging benchmark dataset that more closely resembles real-world optimization problems, although presently the formal 37 datasets mentioned in the paper have yet to be released for review. Recent results with **ORQA** (Mostajabdeh et al., 2025) indicate that multiple-choice QA (MCQ) can diagnose gaps in *recognizing* optimization structure but underestimates the difficulty of *constructing* it: large models achieve respectable scores on reading-comprehension-like items yet fail sharply when entity-relation modeling is required, and chain-of-thought prompting often reduces accuracy. These outcomes align with a representation-centric view of LLM failures in OR. Prior datasets either rely on toy tasks, require code-generation and solver execution for scoring, or conflate notation bugs with modeling errors. **ORQA** improves on this by isolating model-component identification, but its MCQ format still cannot test constructive adequacy.

### 2.2 REPRESENTING AND SOLVING OPTIMIZATION PROBLEMS

An optimization problem captures decision-making under constraints with a goal to minimize, or dually, maximize a particular function. Constraints produce a set of options from which the decision-maker must choose. Whenever the set of options is empty, the problem is *infeasible*. For LLM systems that generate optimization routines, a symbolic formalism capturing the full spectrum of objective functions and constraint types is essential for expressivity and correctness.

Crucial for our purposes, semidefinite programming (SDP), a subclass of convex optimization problems where the feasible set is described by linear matrix inequalities, bridges the numerical algorithmic implementations of optimization and the core real algebraic geometric content. SDP captures non-linear algebraic constraints in a convex formulation, using auxiliary variables and moment representations, acting as the natural computational arena for embedding semialgebraic geometry into optimization solvers (Netzer, 2016). Formally, a program  $\mathbf{P}$  is *semidefinite* if there are some  $c \in \mathbb{R}^n$ , and symmetric matrices  $M_0, M_1, \dots, M_n \in \text{Sym}_N(\mathbb{R})$  such that  $\mathbf{P}$  is given by  $\min \vec{c} \cdot \vec{a}$  subject to  $M_0 + a_1 M_1 + \dots + a_n M_n \succeq 0$ , where  $M \succeq$  means that  $M$  is a semipositive definite matrix.

Following standard model theory conventions, given a first order language  $\mathcal{L}$ , an  $\mathcal{L}$ -structure  $\mathcal{M}$  is some domain  $M$  whose elements satisfies a set of sentences in  $\mathcal{L}$ . For example, given  $\mathcal{L} = \langle 0, 1, +, \cdot \rangle$ , any field  $F$  is defined over the language  $\mathcal{L}$  using the field axioms (Marker, 2002). For a  $D \subset M$ ,  $D$  is a *definable set* if there is an  $\mathcal{L}$ -formula  $\varphi$  such that  $a \in D$  if and only if  $\mathcal{M} \models \varphi(a)$ . In theory, optimization problems rely on feasible sets, while in practice feasible sets and the objective functions are precisely those that are definable in terms from the first-order language of ordered commutative rings  $\mathcal{L}_{\text{or}} = \langle 0, 1, +, \cdot, < \rangle$ . We can expand the language  $\mathcal{L}_{\text{or}}$  to include a function symbol  $E$ , or  $\exp$ , and provide sentences that allow us to interpret  $E$  as the standard real-valued exponential function. It is immediate that there is a bijective correspondence between the polynomials in any given ordered ring  $R[X_1, \dots, X_m]$  with indeterminates  $X_i$ , and the terms generated from  $\mathcal{L}_{\text{or}}$  when adding the constants of  $R$  to the  $\mathcal{L}_{\text{or}}$  (without any extensions of the language, the terms are precisely in correspondence with polynomials in  $\mathbb{Z}[X_1, \dots, X_m]$ ) (Marker, 2002). A field is  $F$  is said to be *real-closed* if there is a total order  $<$  on  $F$  such that every positive element of  $F$  has a square root in  $F$  and every polynomial of odd degree with coefficients in  $F$  has at least one root in  $F$ . The canonical real-closed field is the field of real numbers,  $\mathbb{R}$ . For any real-closed field  $F$ , a set  $\mathcal{W} \subset F^n$  is said to be *basic closed semialgebraic set* if it is a finite intersection of sets defined by polynomial inequalities of the form  $p_i(X_1, \dots, X_n) \geq 0$  for  $p_i \in F[X_1, \dots, X_n]$ , so that  $\mathcal{W} = \{\vec{x} \in F^n \mid \bigwedge_{i \in I_{\mathcal{W}}}^r p_i(\vec{x}) \geq 0\}$ , while a general *semialgebraic set* (alternatively, *semialgebraic variety*) is a finite Boolean combination of basic closed algebraic sets (Netzer, 2016; Marker, 2002). Following the standard definition (see Van Den Dries (1984)), an E-ring  $(R, E)$  augments a commutative ring  $R$  with an exponential map  $E : R \rightarrow R^{\times}$  satisfying  $E(x + y) = E(x) \cdot E(y)$ . E-polynomials are built inductively from  $R$  by adjoining symbols closed under  $+$ ,  $\cdot$ , and  $E(\cdot)$ .

### 3 METHODOLOGY

#### 3.1 NLMOPTIMIZER

Our preliminary investigation concerns the representation of optimization problems, and our experiments principally concern benchmarking with the **Problem** class. We consider the challenge posed by correct intermediate representations (IR) as a matter to be addressed by future development of the **SymInterchange** class, detailed in Appendix C.3. We recognize following Section 2.2 that feasible sets are identifiable with terms satisfying relations determined by the elements of E-rings, we developed a class titled **Problem**, which randomly selects for terms  $t := r \mid x \mid t + t \mid t * t \mid \exp(t)$ , where  $r \in \mathbb{R}$ , are terms subject to the natural  $\leq$  order relation on  $\mathbb{R}$ , subject to hyperparameter selection. Our pragmatic motivation for E-rings is summarized by Corollary B.11 (details in Appendix B).

**Corollary 3.1** (Pragmatic E-ring policy). *If a natural language to intermediate representation mapping stays within  $\mathbb{R}_{\text{or}}$ , feasibility and optimality are decidable, admitting effective QE. If an  $\text{NL} \rightarrow \text{IR}$  mapping stays within  $\mathbb{R}_{\text{an}, \exp}$ , we admit weak quantifier elimination.*

For our initial benchmarks, we set our generator to linear and quadratic programs, which are either classical programs, mixed integer, or integer programs, and have included a restricted implementation of the general problem class in Appendix C.1. This restriction is intentional: problems are guaranteed to admit QE by Cor. B.11, and it keeps algorithmic and solver complexity fixed while scaling representation complexity for the standard benchmark solver, Gurobi.<sup>2</sup> For our initial experiments, we generated a test batch of 1000 linear (LP) and quadratic programs (QP), consisting of a

<sup>2</sup>Consequently, the general E-ring formulation and code paths for  $\exp$  and  $\log$  terms are part of the framework design, but are not exercised in this preliminary empirical study.

216 mix between integer (ILP/IQP), mixed-integer (MILP/MIQP), and real-valued problems (LP/QP),  
 217 which we have summarized in Table 3 in Appendix E.

218 Crucially, with the symbolic representation, we use standard regex methods to construct a default  
 219 instance of Gurobi code, which we then execute to ensure that the problem has a pregenerated and  
 220 annotated solution. Each generated problem is paired with a natural language representation. This  
 221 is achieved by inserting the parameters of the formal optimization model into pre-built natural  
 222 language templates. These templates are structured to mimic human-authored prompts, and include  
 223 redundant sentence options to introduce variation to natural language representations, incorporat-  
 224 ing elliptical references, anaphora, ratio constructions, and domain-specific jargon. Variable and  
 225 resource mentions are deliberately non-uniform to require coreference resolution rather than string  
 226 matching. Appendix D.0.1 enumerates the pattern classes and provides verbatim examples for each,  
 227 along with the sampling logic that yields lexical variety. Natural language prompts are assembled  
 228 from templates using a combination of string formatting and regular expression substitution, ensur-  
 229 ing that the resulting language is coherent and varied while remaining traceable to the underlying  
 230 mathematical representation of an optimization problem. The initial batch of scenarios covers: Of-  
 231 fice supply budgeting, household budgeting, horticulture, cybersecurity staffing and management,  
 232 military force structure, personnel management, and personal diet (such as macronutrient intake).  
 233 Variables and resources are assigned labels from sets of available labels based on the selected con-  
 234 text. Collectively, these statement sets are large enough to provide lexical diversity across problem  
 235 statements exceeding those provided by Xiao et al. (2024).

236 We settled on the current hyperparameters so that roughly half the problems would be real-valued,  
 237 25% mixed integer, and 25% integer programs, three-quarters would be linear programs, and be-  
 238 tween 1/3 and 1/2 would be feasible. Achieving the latter required adjusting the hyperparameters  
 239 used for generating the constraints. While we generate some instances of problems with hundreds  
 240 to thousands of constraints, the majority of problems contain fewer than 36 constraints. We detail in  
 241 Algorithm 1 the generation of a random instance of the Problem class, which we describe in detail  
 242 in Appendix C.1.

### 243 3.2 EXPERIMENTAL SETUP

244 We compare the NLMOptimizer dataset problems against the NL4OPT dataset (Ramamonjison  
 245 et al., 2022a) given the latter’s previous use as a benchmark against novel alternate datasets, its’  
 246 public availability, its’ suitability for zero-shot prompting, and for the number of problems available  
 247 (contra the NLP4LP dataset consisting of 61 MILPs, NL4OPT has more than 600). We test two  
 248 hypotheses: the first is that each LLM will perform worse on NLMOptimizer test problems than  
 249 the NL4OPT benchmark dataset (e.g. lower end-to-end accuracy and higher errors across all experi-  
 250 ments); the second is that expanded prompting to first reason out the specification of the optimization  
 251 problem in the symbolic interchange format can improve the quality and success of the Gurobi code  
 252 output from LLMs (that is, improved end-to-end accuracy and lower error rates). We test these two  
 253 hypotheses using the following models: LLaMa-3.3 (Grattafiori et al., 2024), LLaMa-4-Scout-17B-  
 254 16E-Instruct (AI, 2024), Gemini-1.5-Pro (Team et al., 2024), and GPT-OSS-120B (OpenAI et al.,  
 255 2025). LLaMa-3.3 has a maximum number of tokens of 8000, whereas the other two models had  
 256 a maximum number of tokens of 1000000. We tested each model and each problem set with two  
 257 prompts: a base prompt that establishes the LLMService as a team of operations researchers and  
 258 programmers tasked with producing Gurobi code to model and solve an optimization problem, and  
 259 a second prompt expanding on the base prompt by providing additional instructions to conform their  
 260 responses first to an IR format, before providing a code output. In both prompts, we feed the same  
 261 natural language description of an OR problem.

262 We summarize the problems we generated from our Problem class for our experiment in Ta-  
 263 ble 3, and in Figure 3 provide a cursory visualization of the differences in complexity between  
 264 the NL4OPT dataset and the NLMOptimizer problems we have included in our supple-  
 265 mentary materials. We note that we generated problems with substantially more variables than the  
 266 NL4OPT problems, as well as those subject to multiple types of resources constraints. Whereas  
 267 the NL4OPT dataset only contains problems constrained by one resource, such as budget, or  
 268 space, our problems could have multiple different types of resources, leading to substantially  
 269 more constraints while also producing realistic OR problems. We also explicitly cast constraints  
 indicating variable type, as well as strictly asserting non-negativity. In our summary statis-

270 tics table, we denote all instances on our initial problem set by '-n'. We included the json  
271 and additional generation content from the NL4OPT project (Ramamonjison et al, 2022) in the  
272 dataset folder included in our supplementary materials. The NLMOptimizer problem set can be  
273 found in the 'oproblems' subfolder of the data folder included in our supplementary materials.

Following Xiao et al. (2024), we gathered end-to-end Accuracy (ACC), Compile Error (CE), and Runtime Error (RE), as well as adjusted end-to-end accuracy and runtime error scores. These suffice because the task is not a detection/-classification problem, and the outcome of interest is of constructive adequacy, which these scores directly capture. We note that for our purposes, recording end-to-end accuracy as is standard with these studies suffices ultimately due to the computational complexity of ensuring faithful representations of the optimization problems up to isomorphism, as many of the problems we generated had both redundant constraints or were otherwise infeasible, and our intended. End-to-end accuracy therefore serves as an acceptable proxy for true fidelity in problem representation for a preliminary investigation, while future work ought to expand to tolerate parsimonious, isomorphic representations of extraneous, redundant constraints.

We implement Gurobi in Python 3.10.12 using Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (linux64 - Ubuntu 22.04.4 LTS). All experiments were run using vanilla models, with the LLaMa models hosted locally. We set temperature to 0, top\_p to 1, top\_k to 0, and the frequency and presence penalty to 0. We implemented a 180 second timeout for each model, and when executing code via multiprocessing, implemented a 10 second timer, listing any code that exceeded this threshold as having a runtime error. All local components of the experiments were run with CPU model: AMD Ryzen Threadripper PRO 3995WX 64-Cores, instruction set [SSE2—AVX—AVX2], with a Thread count: 64 physical cores, 128 logical processors, using up to 32 threads. We include all prompts in Appendix D.

## 4 EXPERIMENTAL RESULTS

Table 1: Summary statistics by model, dataset, and additional prompting. Datasets are either NLMOptimizer (NLM) or the NL4OPT (nl4opt) datasets. Additional prompting denoted by sym.

313	model&data&prompt. id	ACC	ACC-adj	CE	RE	RE-adj
314	llama3.3 NLM	0.056	0.056	0.015	0.590	0.590
315	llama4 NLM	0.008	0.049	0.123	0.857	0.729
316	gem NLM	0.161	0.161	0.015	0.101	0.101
317	gptoss NLM	0.031	0.031	0.037	0.212	0.212
318	llama3.3 NLM sym	0.023	0.023	0.015	0.817	0.817
319	llama4 NLM sym	0.001	0.012	0.120	0.878	0.818
320	gem NLM sym	0.146	0.146	0.015	0.071	0.071
321	gptoss NLM sym	0.082	0.082	0.026	0.139	0.139
322	llama3.3 nl4opt	0.593	0.593	0.000	0.024	0.024
323	llama 4 nl4opt	0.017	0.035	0.003	0.995	0.894
324	gem nl4opt	0.660	0.660	0.000	0.027	0.027
325	gptoss nl4opt	0.065	0.065	0.000	0.005	0.005
326	llama 3.3 nl4opt sym	0.597	0.597	0.000	0.112	0.112
327	llama 4 nl4opt sym	0.061	0.064	0.005	0.960	0.954
328	gem nl4opt sym	0.666	0.666	0.000	0.020	0.020
329	gptoss nl4opt sym	0.184	0.184	0.000	0.000	0.000

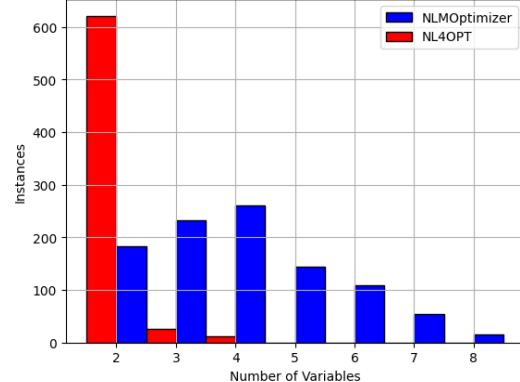


Figure 3: Complexity of the NL4OPT dataset versus our NLMOptimizer dataset with respect to number of variables in a problem instance.

tions of problems, while penalizing those adding

We display the summary statistics of experiments by the model, dataset, and flag for additional prompting in Table 1 (along with Figure 25 displayed in Appendix E), and the Wilcoxon-Signed rank summary statistics in Table 2. Additionally, we plot the relevant statistics for each model and problem type across our different prompting scenarios in Figures 4 to 10, along with the breakdown of each model by experimental set-up. In these figures, each curve summarizes descriptive outcomes across the full 1000 instances provided in the dataset included in our supplementary materials. As these are aggregate outcomes over the provided dataset, they depict deterministic trends rather than sampling uncertainty, which could in principle be derived from gener-

324  
 325  
 326  
 327  
 328  
 329  
 330  
 331  
 332  
 333  
 334  
 335  
 336  
 337  
 338  
 339  
 340  
 341  
 342  
 343  
 344  
 345  
 346  
 347  
 348  
 349  
 350  
 351  
 352  
 353  
 354  
 355  
 356  
 357  
 358  
 359  
 360  
 361  
 362  
 363  
 364  
 365  
 366  
 367  
 368  
 369  
 370  
 371  
 372  
 373  
 374  
 375  
 376  
 377

Finally, we provide expanded tables and figures in Appendix E, including a full audit of instance composition.

We make two immediate observations: across all four models, each performed appreciably worse with respect to end-to-end accuracy, compile error, and run-time errors, even after post hoc adjustment compared to the NL4OPT dataset, with poor performance observed across all problem types; independent of model, problem type, and prompt, end-to-end accuracy decline and errors increased for problems with more than 3 variables. Instances where model end-to-end accuracy improved tended to be spurious as models correctly concluded that the problems were infeasible, but otherwise failed to faithfully represent all named, albeit redundant and inconsistent constraints.

LLaMa-4-Scout-17B-16-Instruct was a particularly poorly performing model, in part because of near uniform failure to properly import Gurobi, which requires `import gurobipy`. Even when correcting for this, as noted by the adjusted end-to-end accuracy and adjusted RE scores, the run time errors remained high, and end-to-end accuracy remained below 10%. Improvements with respect to RE-adj did not lead to equivalent improvements in adjusted end-to-end accuracy, indicating that LLaMa-4 was especially challenged at both generating code and faithfully representing the optimization problem, independent of managing the proper imports.

Table 2: Wilcoxon signed rank one-sided hypothesis summary statistics (score, p-value)

	ACC	CE	RE
E1 (n=8)	(0,0.004)	(36,0.004)	(29,0.074)
E2 (n=8)	(12,0.230)	(5,0.250)	(18,0.527)
E2a (n=6632)	(61494,0.000)	(432,0.014)	(82956,1.000)

Scout, GPT-OSS-120B performed poorly on both datasets but principally due to a failure to reliably capture problem description in the executable Gurobi code format that was requested. GPT-OSS-120B improved the most when including additional prompts to follow an IR template.

In the case of additional prompting, the LLaMa-4 model managed to answer only 1 out of 1000 questions correctly without adjustment, after which it only managed to answer 12 questions correctly. LLaMa-3.3 also performed poorly on our problem set, but otherwise performed comparably to the Gemini model. Similar to LLaMa-4-

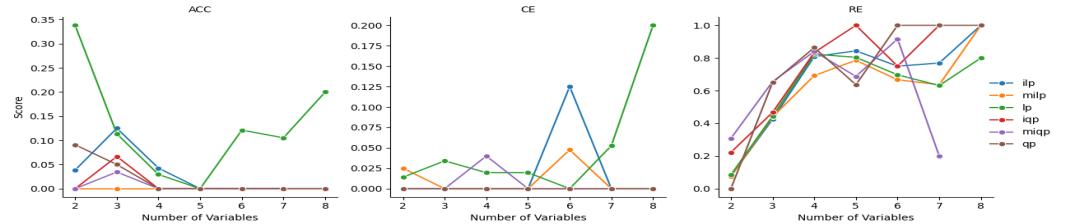


Figure 4: Distribution of scores for LLaMa-3.3 by number of variables.

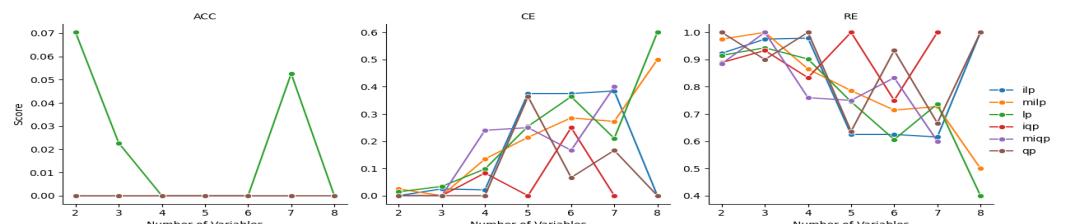


Figure 5: Distribution of scores for LLaMa-4 by number of variables.

The significance of these results for our two experiments, denoted by  $E1$ ,  $E2$  and  $E2a$  are detailed in Table 2, where we gather the Wilcoxon statistic and corresponding p-value for changes to ACC, CE, and RE results across all experiments. Further, while we can only compare the point-estimates

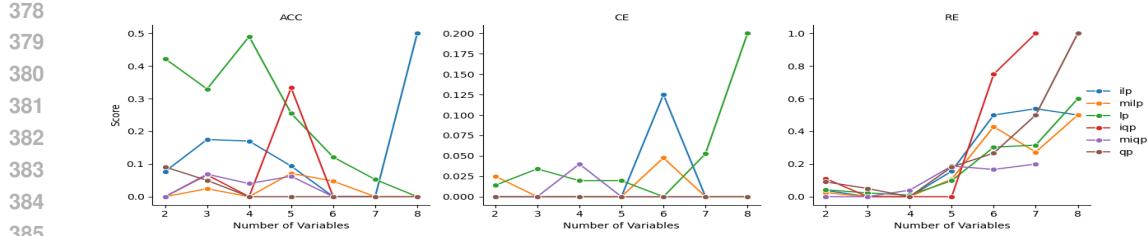


Figure 6: Distribution of scores for Gemini by number of variables.

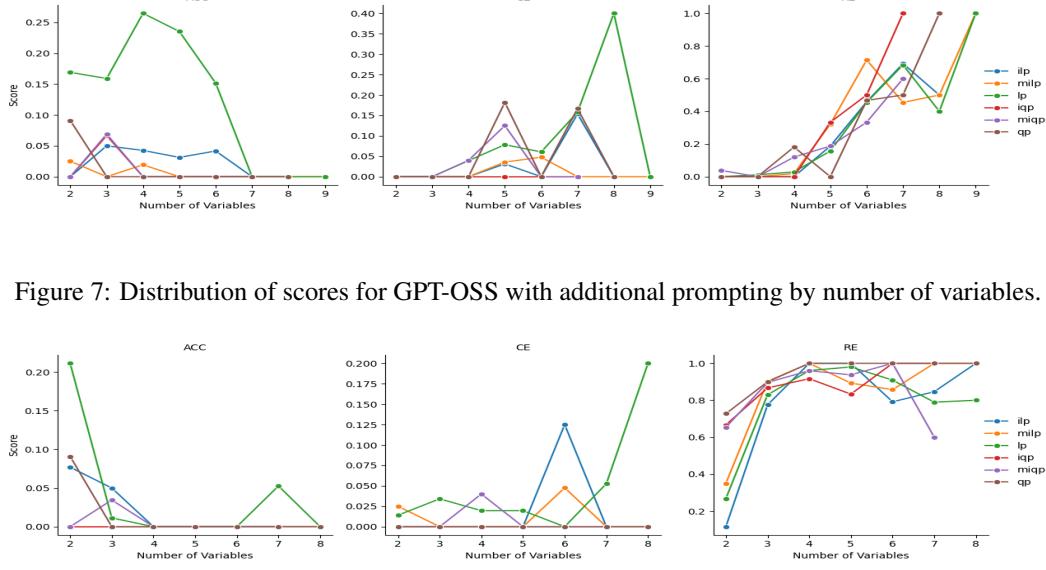


Figure 7: Distribution of scores for GPT-OSS with additional prompting by number of variables.

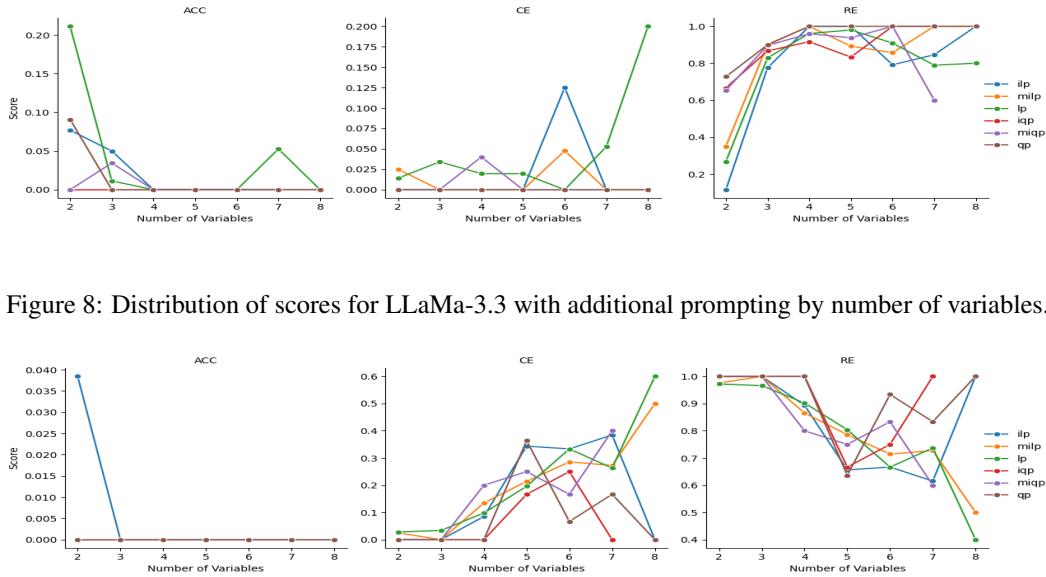


Figure 8: Distribution of scores for LLaMa-3.3 with additional prompting by number of variables.

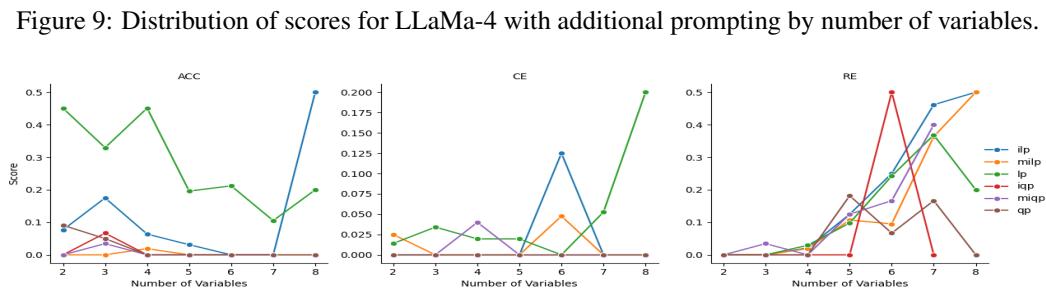


Figure 10: Distribution of scores for Gemini with additional prompting by number of variables.

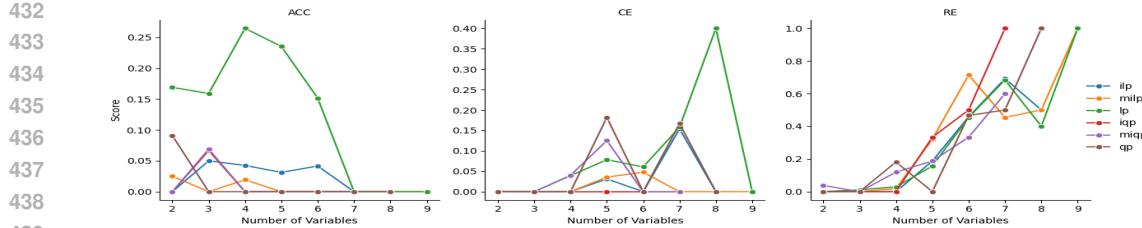


Figure 11: Distribution of scores for GPT-OSS with additional prompting by number of variables.

between models for experiment 1, we can also look at the difference between prompts on a problem-by-problem basis in experiment 2, with the problem-by-problem results displayed as  $E2a$ , and have developed a protocol (see Figure 2) for future dynamic evaluation. We list the sample sizes respectively for the test. In  $E1$  we match against the two datasets, in  $E2$  we match against the results of using the same base prompt for each model against an expanded prompt first specifying a preferred symbolic structure. For the first experiment: we reject the null-hypothesis against the alternative hypothesis that benchmarking NL4OPT against the original problems leads to an average accuracy difference below 0 (i.e. models will be less accurate with respect to the novel data presented by the Problem class), and we reject the null-hypothesis against the alternative hypothesis that benchmarking NL4OPT against the original problems leads to an average Compilation Error difference greater than 0 (i.e. models will have more compilation errors when trying to model the original problems in the Problem class); we fail to reject the null-hypothesis for run-time errors. For the second experiment, both with respect to the point-estimate as well as the problem-by-problem basis, we fail to reject the null-hypothesis against the alternative hypotheses where the additional symbolic prompting improves accuracy (or decreases compile time and run time errors) over the baseline prompting if we group by model. However, when comparing problems directly, we found statistically significant improvements with respect to accuracy and compilation errors when including additional prompting.

## 5 CONCLUSION

Our findings are consistent with the observations in Xiao et al. (2024) and Mostajabdeh et al. (2025): even the best models plateau well below expert performance on modeling-centric questions, finding further evidence that general-purpose instruction-tuned LLMs, even with explicit chain-of-thought or chain-of-experts implementations, fail on more realistic LP/QP instances across those respective ablation studies, and that errors concentrate in categories requiring identification of entities and relations, reinforcing our finding that the core bottleneck is problem representation. We conclude that the NL4OPT competition dataset is less appropriate as a training and benchmarking dataset than problems generated by our Problem class for the task of generating code that correctly solves an optimization problem. The problems in NL4OPT are too simple to be representative of actual optimization problems faced by real-world operations researchers. We further find support that the alternate benchmark datasets discussed in this paper are inadequate for the task of *representing* OR problems for training. We are not proposing a faster solver, nor that MCQ should be abandoned as a complementary benchmark for understanding. We advocate for the adoption of the **Problem** generator class over other alternatives for future model development on the grounds that ours covers a substantially larger class of problems that are relevant to both the real-world OR community, and that training, testing and validation can be done dynamically using the NLMOptimizer protocol so as to avoid overfitting a static dataset. Finally, while outside the scope of our experiments, we also explored initial human-guided interaction between LLMs using a Chain-of-Expert agents approach coupled with the **SymInterchange** class methods and real-time display of problem formulation. We found this successfully improved conversion of problems from natural language to executable Python code that also properly solved the problem for LLaMa-3.3 and Gemini-1.5-Pro. We encourage future research in the direction of multi-shot prompting LLM-agents to use both the Problem class and the corrective methods for the SymInterchange class.

486 REFERENCES  
487

488 Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling  
489 with (mi)lp solvers and large language models, 2024. URL <https://arxiv.org/abs/2402.10172>.

490

491 Meta AI. Introducing llama 4: Advancing multimodal intelligence, 2024. URL <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>.

492

493 Hedy Attouch, Jérôme Bolte, and Benar Fux Svaiter. Convergence of descent methods for semi-  
494 algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized  
495 gauss–seidel methods. *Mathematical programming*, 137(1):91–129, 2013.

496

497 Jean-Pierre Aubin. *Optima and equilibria: an introduction to nonlinear analysis*, volume 140.  
498 Springer Science & Business Media, 2002.

499

500 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*.  
501 Springer, 2006.

502 George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposi-  
503 tion: a synopsis. *ACM SIGSAM Bulletin*, 10(1):10–12, 1976.

504

505 Jan Denef and Lou Van den Dries. P-adic and real subanalytic sets. *Annals of Mathematics*, 128(1):  
506 79–138, 1988.

507

508 Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad  
509 Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan,  
510 Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Ko-  
511 rennev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava  
512 Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux,  
513 Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret,  
514 Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius,  
515 Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary,  
516 Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab  
517 AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco  
518 Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind That-  
519 tai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Kore-  
520 vaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra,  
521 Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Ma-  
522 hadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu,  
523 Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jong-  
524 soo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasudevan Alwala,  
525 Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid  
526 El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren  
527 Rantala-Yearly, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin,  
528 Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi,  
529 Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew  
530 Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Ku-  
531 mar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoy-  
532 chev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan  
533 Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan,  
534 Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ra-  
535 mon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Ro-  
536 hit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan  
537 Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell,  
538 Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparth, Sheng  
539 Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer  
Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman,  
Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mi-  
haylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor  
Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vitor Albiero, Vladan Petrovic, Weiwei

540 Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang  
 541 Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuwei Wang, Yaelle Gold-  
 542 schlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning  
 543 Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh,  
 544 Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria,  
 545 Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein,  
 546 Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, An-  
 547 drew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, An-  
 548 nie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel,  
 549 Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leon-  
 550 hardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu  
 551 Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Mon-  
 552 talvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao  
 553 Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia  
 554 Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide  
 555 Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le,  
 556 Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily  
 557 Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smo-  
 558 thers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni,  
 559 Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia  
 560 Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan,  
 561 Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harri-  
 562 son Rudolph, Helen Suk, Henry Aspégren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj,  
 563 Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James  
 564 Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jen-  
 565 nifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang,  
 566 Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Jun-  
 567 jie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy  
 568 Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang,  
 569 Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell,  
 570 Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa,  
 571 Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias  
 572 Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L.  
 573 Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike  
 574 Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari,  
 575 Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan  
 576 Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong,  
 577 Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent,  
 578 Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar,  
 579 Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Ro-  
 580 driguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy,  
 581 Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin  
 582 Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon,  
 583 Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ra-  
 584 maswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha,  
 585 Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal,  
 586 Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satter-  
 587 field, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj  
 588 Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo  
 589 Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook  
 590 Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Ku-  
 591 mar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihăilescu, Vladimir Ivanov,  
 592 Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiao-  
 593 jian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia,  
 Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao,  
 Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhao-  
 duo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL  
<https://arxiv.org/abs/2407.21783>.

594 Paul A Jensen and Jonathan F Bard. *Operations research models and methods*. John Wiley & Sons,  
 595 2002.

596

597 Cédric Josz and Daniel K Molzahn. Lasserre hierarchy for large scale polynomial optimization in  
 598 real and complex variables. *SIAM Journal on Optimization*, 28(2):1017–1048, 2018.

599

600 David Marker. *Model theory: an introduction*. Springer, 2002.

601

602 Mahdi Mostajabdeh, Timothy Tin Long Yu, Samarendra Chandan Bindu Dash, Rindra Rama-  
 603 monjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Evaluating llm  
 604 reasoning in the operations research domain with orqa. In *Proceedings of the AAAI Conference  
 605 on Artificial Intelligence*, volume 39, pp. 24902–24910, 2025.

606

607 Tim Netzer. Real algebraic geometry and its applications. 2016. URL <https://arxiv.org/abs/1606.07284>.

608

609 OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin  
 610 Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler  
 611 Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sébastien Bubeck, Che Chang, Kai  
 612 Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin  
 613 Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam  
 614 Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec  
 615 Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina  
 616 Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc,  
 617 James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin,  
 618 Jordan Liss, Lily Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCal-  
 619 lum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu,  
 620 Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ash-  
 621 ley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic  
 622 Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo  
 623 Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotstetd, Hadi Salman, Suvansh  
 624 Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singh, Yang Song,  
 625 Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric  
 626 Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery,  
 627 Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech  
 628 Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-  
 629 120b & gpt-oss-20b model card, 2025. URL <https://arxiv.org/abs/2508.10925>.

630

631 Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghad-  
 632 dar, Shiqi He, Mahdi Mostajabdeh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang.  
 633 Nl4opt competition: Formulating optimization problems based on their natural language descrip-  
 634 tions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht (eds.), *Proceedings of the  
 635 NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*,  
 636 pp. 189–203. PMLR, 28 Nov–09 Dec 2022a. URL <https://proceedings.mlr.press/v220/ramamonjison23a.html>.

637

638 Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghad-  
 639 dar, Shiqi He, Mahdi Mostajabdeh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang.  
 640 Nl4opt competition: Formulating optimization problems based on their natural language descrip-  
 641 tions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht (eds.), *Proceedings of the  
 642 NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*,  
 643 pp. 189–203. PMLR, 28 Nov–09 Dec 2022b. URL <https://proceedings.mlr.press/v220/ramamonjison23a.html>.

644

645 . Ramamonjison et al. Augmenting operations research with auto-formulation of optimization mod-  
 646 els from problem descriptions. In *Proceedings of the 2022 Conference on Empirical Methods  
 647 in Natural Language Processing: Industry Track*, pp. 29–62, Abu Dhabi, UAE, December 2022.  
 648 Association for Computational Linguistics. URL <https://aclanthology.org/2022.emnlp-industry.4>.

649

650 Abraham Seidenberg. A new decision method for elementary algebra. *Annals of Mathematics*, 60  
 651 (2):365–374, 1954.

648 Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer,  
 649 Damien Vincent, Zhufeng Pan, Shibo Wang, Soroosh Mariooryad, Yifan Ding, Xinyang Geng,  
 650 Fred Alcober, Roy Frostig, Mark Omernick, Lexi Walker, Cosmin Paduraru, Christina Sorokin,  
 651 Andrea Tacchetti, Colin Gaffney, Samira Daruki, Olcan Sercinoglu, Zach Gleicher, Juliette Love,  
 652 Paul Voigtlaender, Rohan Jain, Gabriela Surita, Kareem Mohamed, Rory Blevins, Junwhan Ahn,  
 653 Tao Zhu, Kornraphop Kawintiranon, Orhan Firat, Yiming Gu, Yujing Zhang, Matthew Rahtz,  
 654 Manaal Faruqui, Natalie Clay, Justin Gilmer, JD Co-Reyes, Ivo Penchev, Rui Zhu, Nobuyuki  
 655 Morioka, Kevin Hui, Krishna Haridasan, Victor Campos, Mahdis Mahdieh, Mandy Guo, Samer  
 656 Hassan, Kevin Kilgour, Arpi Vezer, Heng-Tze Cheng, Raoul de Liedekerke, Siddharth Goyal,  
 657 Paul Barham, DJ Strouse, Seb Noury, Jonas Adler, Mukund Sundararajan, Sharad Vikram, Dmitry  
 658 Lepikhin, Michela Paganini, Xavier Garcia, Fan Yang, Dasha Valter, Maja Trebacz, Kiran Vo-  
 659 drahalli, Chulayuth Asawaroengchai, Roman Ring, Norbert Kalb, Livio Baldini Soares, Sid-  
 660 dhartha Brahma, David Steiner, Tianhe Yu, Fabian Mentzer, Antoine He, Lucas Gonzalez, Bibo  
 661 Xu, Raphael Lopez Kaufman, Laurent El Shafey, Junhyuk Oh, Tom Hennigan, George van den  
 662 Driessche, Seth Odoom, Mario Lucic, Becca Roelofs, Sid Lall, Amit Marathe, Betty Chan, San-  
 663 tiago Ontanon, Luheng He, Denis Teplyashin, Jonathan Lai, Phil Crone, Bogdan Damoc, Lewis  
 664 Ho, Sebastian Riedel, Karel Lenc, Chih-Kuan Yeh, Aakanksha Chowdhery, Yang Xu, Mehran  
 665 Kazemi, Ehsan Amid, Anastasia Petrushkina, Kevin Swersky, Ali Khodaei, Gwoon Chen, Chris  
 666 Larkin, Mario Pinto, Geng Yan, Adria Puigdomenech Badia, Piyush Patil, Steven Hansen, Dave  
 667 Orr, Sebastien M. R. Arnold, Jordan Grimstad, Andrew Dai, Sholto Douglas, Rishika Sinha, Vikas  
 668 Yadav, Xi Chen, Elena Gribovskaya, Jacob Austin, Jeffrey Zhao, Kaushal Patel, Paul Komarek,  
 669 Sophia Austin, Sebastian Borgeaud, Linda Friso, Abhimanyu Goyal, Ben Caine, Kris Cao, Da-  
 670 Woon Chung, Matthew Lamm, Gabe Barth-Maron, Thais Kagohara, Kate Olszewska, Mia Chen,  
 671 Kaushik Shivakumar, Rishabh Agarwal, Harshal Godhia, Ravi Rajwar, Javier Snaider, Xerxes  
 672 Dotiwalla, Yuan Liu, Aditya Barua, Victor Ungureanu, Yuan Zhang, Bat-Orgil Batsaikhan, Ma-  
 673 teo Wirth, James Qin, Ivo Danihelka, Tulsee Doshi, Martin Chadwick, Jilin Chen, Sanil Jain,  
 674 Quoc Le, Arjun Kar, Madhu Gurumurthy, Cheng Li, Ruoxin Sang, Fangyu Liu, Lampros Lam-  
 675 prou, Rich Munoz, Nathan Lintz, Harsh Mehta, Heidi Howard, Malcolm Reynolds, Lora Aroyo,  
 676 Quan Wang, Lorenzo Blanco, Albin Cassirer, Jordan Griffith, Dipanjan Das, Stephan Lee, Jakub  
 677 Sygnowski, Zach Fisher, James Besley, Richard Powell, Zafarali Ahmed, Dominik Paulus, David  
 678 Reitter, Zalan Borsos, Rishabh Joshi, Aedan Pope, Steven Hand, Vittorio Selo, Vihan Jain, Nikhil  
 679 Sethi, Megha Goel, Takaki Makino, Rhys May, Zhen Yang, Johan Schalkwyk, Christina But-  
 680 terfield, Anja Hauth, Alex Goldin, Will Hawkins, Evan Senter, Sergey Brin, Oliver Woodman,  
 681 Marvin Ritter, Eric Noland, Minh Giang, Vijay Bolina, Lisa Lee, Tim Blyth, Ian Mackinnon,  
 682 Machel Reid, Obaid Sarvana, David Silver, Alexander Chen, Lily Wang, Loren Maggiore, Os-  
 683 car Chang, Nithya Attaluri, Gregory Thornton, Chung-Cheng Chiu, Oskar Bunyan, Nir Levine,  
 684 Timothy Chung, Evgenii Eltyshev, Xiance Si, Timothy Lillicrap, Demetra Brady, Vaibhav Ag-  
 685 garwal, Boxi Wu, Yuanzhong Xu, Ross McIlroy, Kartikeya Badola, Paramjit Sandhu, Erica Mor-  
 686 eira, Wojciech Stokowiec, Ross Hemsley, Dong Li, Alex Tudor, Pranav Shyam, Elahe Rahim-  
 687 toroghi, Salem Haykal, Pablo Sprechmann, Xiang Zhou, Diana Mincu, Yujia Li, Ravi Addanki,  
 688 Kalpesh Krishna, Xiao Wu, Alexandre Frechette, Matan Eyal, Allan Dafoe, Dave Lacey, Jay  
 689 Whang, Thi Avrahami, Ye Zhang, Emanuel Taropa, Hanzhao Lin, Daniel Toyama, Eliza Ruther-  
 690 ford, Motoki Sano, HyunJeong Choe, Alex Tomala, Chalence Safranek-Shrader, Nora Kassner,  
 691 Mantas Pajarskas, Matt Harvey, Sean Sechrist, Meire Fortunato, Christina Lyu, Gamaleldin El-  
 692 sayed, Chenkai Kuang, James Lottes, Eric Chu, Chao Jia, Chih-Wei Chen, Peter Humphreys,  
 693 Kate Baumli, Connie Tao, Rajkumar Samuel, Cicero Nogueira dos Santos, Anders Andreassen,  
 694 Nemanja Račićević, Dominik Grawe, Aviral Kumar, Stephanie Winkler, Jonathan Caton, Andrew  
 695 Brock, Sid Dalmia, Hannah Sheahan, Iain Barr, Yingjie Miao, Paul Natsev, Jacob Devlin, Feryal  
 696 Behbahani, Flavien Prost, Yanhua Sun, Artiom Myaskovsky, Thanumalayan Sankaranarayana  
 697 Pillai, Dan Hurt, Angeliki Lazaridou, Xi Xiong, Ce Zheng, Fabio Pardo, Xiaowei Li, Dan Hor-  
 698 gan, Joe Stanton, Moran Ambar, Fei Xia, Alejandro Lince, Mingqiu Wang, Basil Mustafa, Al-  
 699 bert Webson, Hyo Lee, Rohan Anil, Martin Wicke, Timothy Dozat, Abhishek Sinha, Enrique  
 700 Piqueras, Elahe Dabir, Shyam Upadhyay, Anudhyan Boral, Lisa Anne Hendricks, Corey Fry,  
 701 Josip Djolonga, Yi Su, Jake Walker, Jane Labanowski, Ronny Huang, Vedant Misra, Jeremy  
 Chen, RJ Skerry-Ryan, Avi Singh, Shruti Rijhwani, Dian Yu, Alex Castro-Ros, Beer Chang-  
 702 pinyo, Romina Datta, Sumit Bagri, Arnar Mar Hrafnkelsson, Marcello Maggioni, Daniel Zheng,  
 703 Yury Sulsky, Shaobo Hou, Tom Le Paine, Antoine Yang, Jason Riesa, Dominika Rogozinska,  
 704 Dror Marcus, Dalia El Badawy, Qiao Zhang, Luyu Wang, Helen Miller, Jeremy Greer, Lars Lowe  
 705 Sjos, Azade Nova, Heiga Zen, Rahma Chaabouni, Mihaela Rosca, Jiepu Jiang, Charlie Chen,

702 Rubo Liu, Tara Sainath, Maxim Krikun, Alex Polozov, Jean-Baptiste Lespiau, Josh Newlan,  
 703 Zeynep Cankara, Soo Kwak, Yunhan Xu, Phil Chen, Andy Coenen, Clemens Meyer, Katerina  
 704 Tsihlas, Ada Ma, Juraj Gottweis, Jinwei Xing, Chenjie Gu, Jin Miao, Christian Frank, Zeynep  
 705 Cankara, Sanjay Ganapathy, Ishita Dasgupta, Steph Hughes-Fitt, Heng Chen, David Reid, Keran  
 706 Rong, Hongmin Fan, Joost van Amersfoort, Vincent Zhuang, Aaron Cohen, Shixiang Shane Gu,  
 707 Anhad Mohananey, Anastasija Ilic, Taylor Tobin, John Wieting, Anna Bortsova, Phoebe Thacker,  
 708 Emma Wang, Emily Caveness, Justin Chiu, Eren Sezener, Alex Kaskasoli, Steven Baker, Katie  
 709 Millican, Mohamed Elhawaty, Kostas Aisopos, Carl Lebsack, Nathan Byrd, Hanjun Dai, Wen-  
 710 hao Jia, Matthew Wiethoff, Elnaz Davoodi, Albert Weston, Lakshman Yagati, Arun Ahuja, Isabel  
 711 Gao, Golan Pundak, Susan Zhang, Michael Azzam, Khe Chai Sim, Sergi Caelles, James Keel-  
 712 ing, Abhanshu Sharma, Andy Swing, YaGuang Li, Chenxi Liu, Carrie Grimes Bostock, Yamini  
 713 Bansal, Zachary Nado, Ankesh Anand, Josh Lipschultz, Abhijit Karmarkar, Lev Proleev, Abe It-  
 714 tycheriah, Soheil Hassas Yeganeh, George Polovets, Aleksandra Faust, Jiao Sun, Alban Rustemi,  
 715 Pen Li, Rakesh Shivanna, Jeremiah Liu, Chris Welty, Federico Lebron, Anirudh Baddepudi, Se-  
 716 bastian Krause, Emilio Parisotto, Radu Soricu, Zheng Xu, Dawn Bloxwich, Melvin Johnson,  
 717 Behnam Neyshabur, Justin Mao-Jones, Renshen Wang, Vinay Ramasesh, Zaheer Abbas, Arthur  
 718 Guez, Constant Segal, Duc Dung Nguyen, James Svensson, Le Hou, Sarah York, Kieran Mi-  
 719 lan, Sophie Bridgers, Wiktor Gworek, Marco Tagliasacchi, James Lee-Thorp, Michael Chang,  
 720 Alexey Guseynov, Ale Jakse Hartman, Michael Kwong, Ruizhe Zhao, Sheleem Kashem, Eliz-  
 721 abeth Cole, Antoine Miech, Richard Tanburn, Mary Phuong, Filip Pavetic, Sebastien Cevey,  
 722 Ramona Comanescu, Richard Ives, Sherry Yang, Cosmo Du, Bo Li, Zizhao Zhang, Mariko  
 723 Iinuma, Clara Huiyi Hu, Aurko Roy, Shaan Bijwadia, Zhenkai Zhu, Danilo Martins, Rachel Sa-  
 724 putro, Anita Gergely, Steven Zheng, Dawei Jia, Ioannis Antonoglou, Adam Sadovsky, Shane  
 725 Gu, Yingying Bi, Alek Andreev, Sina Samangooei, Mina Khan, Tomas Kociský, Angelos Filos,  
 726 Chintu Kumar, Colton Bishop, Adams Yu, Sarah Hodkinson, Sid Mittal, Premal Shah, Alexandre  
 727 Moufarek, Yong Cheng, Adam Bloniarz, Jaehoon Lee, Pedram Pejman, Paul Michel, Stephen  
 728 Spencer, Vladimir Feinberg, Xuehan Xiong, Nikolay Savinov, Charlotte Smith, Siamak Shakeri,  
 729 Dustin Tran, Mary Chesus, Bernd Bohnet, George Tucker, Tamara von Glehn, Carrie Muir, Yiran  
 730 Mao, Hideto Kazawa, Ambrose Slone, Kedar Soparkar, Disha Shrivastava, James Cobon-Kerr,  
 731 Michael Sharman, Jay Pavagadhi, Carlos Araya, Karolis Misiunas, Nimesh Ghelani, Michael  
 732 Laskin, David Barker, Qiuqia Li, Anton Briukhov, Neil Houlsby, Mia Glaese, Balaji Laksh-  
 733 minarayanan, Nathan Schucher, Yunhao Tang, Eli Collins, Hyeontaek Lim, Fangxiaoyu Feng,  
 734 Adria Recasens, Guangda Lai, Alberto Magni, Nicola De Cao, Aditya Siddhant, Zoe Ashwood,  
 735 Jordi Orbay, Mostafa Dehghani, Jenny Brennan, Yifan He, Kelvin Xu, Yang Gao, Carl Saroufim,  
 736 James Molloy, Xinyi Wu, Seb Arnold, Solomon Chang, Julian Schrittweiser, Elena Buchatskaya,  
 737 Soroush Radpour, Martin Polacek, Skye Giordano, Ankur Bapna, Simon Tokumine, Vincent  
 738 Hellendoorn, Thibault Sottiaux, Sarah Cogan, Aliaksei Severyn, Mohammad Saleh, Shantanu  
 739 Thakoor, Laurent Shefey, Siyuan Qiao, Meenu Gaba, Shuo yiin Chang, Craig Swanson, Biao  
 740 Zhang, Benjamin Lee, Paul Kishan Rubenstein, Gan Song, Tom Kwiatkowski, Anna Koop, Ajay  
 741 Kannan, David Kao, Parker Schuh, Axel Stjerngren, Golnaz Ghiasi, Gena Gibson, Luke Vilnis,  
 742 Ye Yuan, Felipe Tiengo Ferreira, Aishwarya Kamath, Ted Klimenko, Ken Franko, Kefan Xiao,  
 743 Indro Bhattacharya, Miteyan Patel, Rui Wang, Alex Morris, Robin Strudel, Vivek Sharma, Peter  
 744 Choy, Sayed Hadi Hashemi, Jessica Landon, Mara Finkelstein, Priya Jhakra, Justin Frye, Megan  
 745 Barnes, Matthew Mauger, Dennis Daun, Khuslen Baatarsukh, Matthew Tung, Wael Farhan, Hen-  
 746 ryk Michalewski, Fabio Viola, Felix de Chaumont Quirhy, Charline Le Lan, Tom Hudson, Qingze  
 747 Wang, Felix Fischer, Ivy Zheng, Elspeth White, Anca Dragan, Jean baptiste Alayrac, Eric Ni,  
 748 Alexander Pritzel, Adam Iwanicki, Michael Isard, Anna Bulanova, Lukas Zilka, Ethan Dyer,  
 749 Devendra Sachan, Srivatsan Srinivasan, Hannah Muckenhirk, Honglong Cai, Amol Mandhane,  
 750 Mukarram Tariq, Jack W. Rae, Gary Wang, Kareem Ayoub, Nicholas FitzGerald, Yao Zhao,  
 751 Woohyun Han, Chris Alberti, Dan Garrette, Kashyap Krishnakumar, Mai Gimenez, Anselm Lev-  
 752 skaya, Daniel Sohn, Josip Matak, Inaki Iturrate, Michael B. Chang, Jackie Xiang, Yuan Cao,  
 753 Nishant Ranka, Geoff Brown, Adrian Hutter, Vahab Mirrokni, Nanxin Chen, Kaisheng Yao,  
 754 Zoltan Egyed, Francois Galilee, Tyler Liechty, Praveen Kallakuri, Evan Palmer, Sanjay Ghe-  
 755 mawat, Jasmine Liu, David Tao, Chloe Thornton, Tim Green, Mimi Jasarevic, Sharon Lin, Victor  
 Cotruta, Yi-Xuan Tan, Noah Fiedel, Hongkun Yu, Ed Chi, Alexander Neitz, Jens Heitkaemper,  
 Anu Sinha, Denny Zhou, Yi Sun, Charbel Kaed, Brice Hulse, Swaroop Mishra, Maria Georgaki,  
 Sneha Kudugunta, Clement Farabet, Izhak Shafran, Daniel Vlasic, Anton Tsitsulin, Rajagopal  
 Ananthanarayanan, Alen Carin, Guolong Su, Pei Sun, Shashank V, Gabriel Carvajal, Josef Broder,  
 Iulia Comsa, Alena Repina, William Wong, Warren Weilun Chen, Peter Hawkins, Egor Filonov,

Lucia Loher, Christoph Hirnschall, Weiyi Wang, Jingchen Ye, Andrea Burns, Hardie Cate, Diana Gage Wright, Federico Piccinini, Lei Zhang, Chu-Cheng Lin, Ionel Gog, Yana Kulizhskaya, Ashwin Sreevatsa, Shuang Song, Luis C. Cobo, Anand Iyer, Chetan Tekur, Guillermo Garrido, Zhuyun Xiao, Rupert Kemp, Huaixiu Steven Zheng, Hui Li, Ananth Agarwal, Christel Ngani, Kati Goshvadi, Rebeca Santamaria-Fernandez, Wojciech Fica, Xinyun Chen, Chris Gorgolewski, Sean Sun, Roopal Garg, Xinyu Ye, S. M. Ali Eslami, Nan Hua, Jon Simon, Pratik Joshi, Yelin Kim, Ian Tenney, Sahitya Potluri, Lam Nguyen Thiet, Quan Yuan, Florian Luisier, Alexandra Chronopoulou, Salvatore Scellato, Praveen Srinivasan, Minmin Chen, Vinod Koverkathu, Valentin Dalibard, Yaming Xu, Brennan Saeta, Keith Anderson, Thibault Sellam, Nick Fernando, Fantine Huot, Junehyuk Jung, Mani Varadarajan, Michael Quinn, Amit Raul, Maigo Le, Ruslan Habalov, Jon Clark, Komal Jalan, Kalesha Bullard, Achintya Singhal, Thang Luong, Boyu Wang, Sujeewan Rajayogam, Julian Eisenschlos, Johnson Jia, Daniel Finchelstein, Alex Yakubovich, Daniel Balle, Michael Fink, Sameer Agarwal, Jing Li, Dj Dvijotham, Shalini Pal, Kai Kang, Jaclyn Konzelmann, Jennifer Beattie, Olivier Dousse, Diane Wu, Remi Crocker, Chen Elkind, Siddhartha Reddy Jonnalagadda, Jong Lee, Dan Holtmann-Rice, Krystal Kallarackal, Rosanne Liu, Denis Vnukov, Neera Vats, Luca Invernizzi, Mohsen Jafari, Huanjie Zhou, Lilly Taylor, Jennifer Prendki, Marcus Wu, Tom Eccles, Tianqi Liu, Kavya Kopparapu, Francoise Beaufays, Christof Angermueller, Andreea Marzoca, Shourya Sarcar, Hilal Dib, Jeff Stanway, Frank Perbet, Nejc Trdin, Rachel Sterneck, Andrey Khorlin, Dinghua Li, Xihui Wu, Sonam Goenka, David Madras, Sasha Goldshtain, Willi Gierke, Tong Zhou, Yixin Liu, Yannie Liang, Anais White, Yunjie Li, Shreya Singh, Sanaz Bahargam, Mark Epstein, Sujoy Basu, Li Lao, Adnan Ozturk, Carl Crous, Alex Zhai, Han Lu, Zora Tung, Neeraj Gaur, Alanna Walton, Lucas Dixon, Ming Zhang, Amir Globerson, Grant Uy, Andrew Bolt, Olivia Wiles, Milad Nasr, Ilia Shumailov, Marco Selvi, Francesco Piccinno, Ricardo Aguilar, Sara McCarthy, Misha Khalman, Mrinal Shukla, Vlado Galic, John Carpenter, Kevin Villela, Haibin Zhang, Harry Richardson, James Martens, Matko Bosnjak, Shreyas Rammohan Belle, Jeff Seibert, Mahmoud Alnahlawi, Brian McWilliams, Sankalp Singh, Annie Louis, Wen Ding, Dan Popovici, Lenin Simicich, Laura Knight, Pulkit Mehta, Nishesh Gupta, Chongyang Shi, Saaber Fatehi, Jovana Mitrovic, Alex Grills, Joseph Paganoda, Tsendsuren Munkhdalai, Dessie Petrova, Danielle Eisenbud, Zhishuai Zhang, Damion Yates, Bhavishya Mittal, Nilesh Tripuraneni, Yannis Assael, Thomas Brovelli, Prateek Jain, Mihajlo Velimirovic, Canfer Akbulut, Jiaqi Mu, Wolfgang Macherey, Ravin Kumar, Jun Xu, Haroon Qureshi, Gheorghe Comanici, Jeremy Wiesner, Zhitao Gong, Anton Ruddock, Matthias Bauer, Nick Felt, Anirudh GP, Anurag Arnab, Dustin Zelle, Jonas Rothfuss, Bill Rosgen, Ashish Shenoy, Bryan Seybold, Xinjian Li, Jayaram Mudigonda, Goker Erdogan, Jiawei Xia, Jiri Simsa, Andrea Michi, Yi Yao, Christopher Yew, Steven Kan, Isaac Caswell, Carey Radabaugh, Andre Elisseeff, Pedro Valenzuela, Kay McKinney, Kim Paterson, Albert Cui, Eri Latorre-Chimoto, Solomon Kim, William Zeng, Ken Durden, Priya Ponnappalli, Tiberiu Sosea, Christopher A. Choquette-Choo, James Manyika, Brona Robenek, Harsha Vashisht, Sebastien Pereira, Hoi Lam, Marko Velic, Denese Owusu-Afriyie, Katherine Lee, Tolga Bolukbasi, Alicia Parrish, Shawn Lu, Jane Park, Balaji Venkatraman, Alice Talbert, Lambert Rosique, Yuchung Cheng, Andrei Sozanschi, Adam Paszke, Praveen Kumar, Jessica Austin, Lu Li, Khalid Salama, Bartek Perz, Wooyeol Kim, Nandita Dukkipati, Anthony Baryshnikov, Christos Kaplanis, XiangHai Sheng, Yuri Chervonyi, Caglar Unlu, Diego de Las Casas, Harry Askham, Kathryn Tunyasuvunakool, Felix Gimeno, Siim Poder, Chester Kwak, Matt Miecnikowski, Vahab Mirrokni, Alek Dimitriev, Aaron Parisi, Dangyi Liu, Tomy Tsai, Toby Shevlane, Christina Kouridi, Drew Garmon, Adrian Goedeckemeyer, Adam R. Brown, Anitha Vijayakumar, Ali Elqursh, Sadegh Jazayeri, Jin Huang, Sara Mc Carthy, Jay Hoover, Lucy Kim, Sandeep Kumar, Wei Chen, Courtney Biles, Garrett Bingham, Evan Rosen, Lisa Wang, Qijun Tan, David Engel, Francesco Pongetti, Dario de Cesare, Dongseong Hwang, Lily Yu, Jennifer Pullman, Srini Narayanan, Kyle Levin, Siddharth Gopal, Megan Li, Asaf Aharoni, Trieu Trinh, Jessica Lo, Norman Casagrande, Roopal Vij, Loic Matthey, Bramandia Ramadhana, Austin Matthews, CJ Carey, Matthew Johnson, Kremena Goranova, Rohin Shah, Shereen Ashraf, Kingshuk Dasgupta, Rasmus Larsen, Yicheng Wang, Manish Reddy Vuyyuru, Chong Jiang, Joana Ijazi, Kazuki Osawa, Celine Smith, Ramya Sree Boppana, Taylan Bilal, Yuma Koizumi, Ying Xu, Yasemin Altun, Nir Shabat, Ben Bariach, Alex Korchemniy, Kiam Choo, Olaf Ronneberger, Chimezie Iwuanyanwu, Shubin Zhao, David Soergel, Cho-Jui Hsieh, Irene Cai, Shariq Iqbal, Martin Sundermeyer, Zhe Chen, Elie Bursztein, Chaitanya Malaviya, Fadi Biadsy, Prakash Shroff, Inderjit Dhillon, Tejasji Latkar, Chris Dyer, Hannah Forbes, Massimo Nicosia, Vitaly Nikolaev, Somer Greene, Marin Georgiev, Pidong Wang, Nina Martin, Hanie Sedghi, John Zhang, Praseem Banzal, Doug Fritz, Vikram Rao, Xuezhi Wang, Jiageng Zhang, Viorica Pa-

810 trucean, Dayou Du, Igor Mordatch, Ivan Jurin, Lewis Liu, Ayush Dubey, Abhi Mohan, Janek  
 811 Nowakowski, Vlad-Doru Ion, Nan Wei, Reiko Tojo, Maria Abi Raad, Drew A. Hudson, Vaishakh  
 812 Keshava, Shubham Agrawal, Kevin Ramirez, Zhichun Wu, Hoang Nguyen, Ji Liu, Madhavi Se-  
 813 wak, Bryce Petrini, DongHyun Choi, Ivan Philips, Ziyue Wang, Ioana Bica, Ankush Garg, Jarek  
 814 Wilkiewicz, Priyanka Agrawal, Xiaowei Li, Danhao Guo, Emily Xue, Naseer Shaik, Andrew  
 815 Leach, Sadh MNM Khan, Julia Wiesinger, Sammy Jerome, Abhishek Chakladar, Alek Wenjiao  
 816 Wang, Tina Ornduff, Folake Abu, Alireza Ghaffarkhah, Marcus Wainwright, Mario Cortes, Fred-  
 817 erick Liu, Joshua Maynez, Andreas Terzis, Pouya Samangouei, Riham Mansour, Tomasz Kepa,  
 818 Fran ois-Xavier Aubet, Anton Algymr, Dan Banica, Agoston Weisz, Andras Orban, Alexandre  
 819 Senges, Ewa Andrejczuk, Mark Geller, Niccolo Dal Santo, Valentin Anklin, Majd Al Merey,  
 820 Martin Baeuml, Trevor Strohman, Junwen Bai, Slav Petrov, Yonghui Wu, Demis Hassabis, Koray  
 821 Kavukcuoglu, Jeff Dean, and Oriol Vinyals. Gemini 1.5: Unlocking multimodal understanding  
 822 across millions of tokens of context, 2024. URL <https://arxiv.org/abs/2403.05530>.

823 Lou Van Den Dries. Exponential rings, exponential polynomials and exponential functions. *Pacific  
 824 Journal of Mathematics*, 1984.

825 Lou Van den Dries. *Tame topology and o-minimal structures*, volume 248. Cambridge university  
 826 press, 1998.

827 Lou Van den Dries and Chris Miller. Geometric categories and o-minimal structures. 1996.

828 Lou van den Dries, Angus Macintyre, and David Marker. The elementary theory of restricted ana-  
 829 lytic fields with exponentiation. *Annals of Mathematics*, 140(1):183–205, 1994.

830 Alex J Wilkie. Model completeness results for expansions of the ordered field of real numbers by  
 831 restricted pfaffian functions and the exponential function. *Journal of the American Mathematical  
 832 Society*, 9(4):1051–1094, 1996.

833 Alex J Wilkie. Schanuel’s conjecture and the decidability of the real exponential field. In *Algebraic  
 834 Model Theory*, pp. 223–230. Springer, 1997.

835 Alex J Wilkie. O-minimality. *Documenta Mathematica, Extra Volume ICM*, 1:633–636, 1998.

836 Alex J Wilkie. A theorem of the complement and some new o-minimal structures. *Selecta Mathe-  
 837 matica*, 5(4):397–421, 1999.

838 Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin  
 839 Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet  
 840 complex operations research problems. In *The Twelfth International Conference on Learning  
 841 Representations*, 2024. URL <https://openreview.net/forum?id=HobyL1B9CZ>.

842

## 843 A APPENDIX

844 A full reproduction of the Problem class will not fit in an Appendix, but has been included in the  
 845 Supplementary materials. However, we have provided details for the hyperparameters we used for  
 846 generating the 1000 problems used for benchmarking, the experimental set-up, the natural language  
 847 problem generation, and some additional breakdown of the performance of the LLMs.

## 848 B THEORETICAL BACKGROUND

849 Our choice of E-rings as the principle representation for OR optimization problems relies on a  
 850 pragmatic argument that this is a suitable formal object for representing a wide class of problems  
 851 of interest that are largely decidable, and such that intermediate representations of definable sets are  
 852 amenable to neurosymbolic to determine if problems are solvable.

853 Throughout this Appendix, we expand on the material in Section 2.2, providing the readers primers  
 854 on quantifier elimination (QE), o-minimality, and E-rings, along with corresponding results that  
 855 provide decidability guarantees.

864 B.1 OPTIMIZATION FORMALLY UNDERSTOOD  
865866 We first summarize the primary scope of optimization problems considered in the main body of the  
867 paper.868 *Linear programs* involve a linear objective function over a polyhedron defined by linear equalities  
869 and inequalities, whereas *quadratic programs* allow for the objective function to be a quadratic  
870 function. Convex optimization occupies a central place in this landscape, characterized by problems  
871 where both the objective function and feasible region are convex (Jensen & Bard, 2002). Convexity  
872 guarantees the existence of global minima and underpins the success of scalable first- and second-  
873 order methods, however it is not enough to reveal the algebraic structure of optimization problems  
874 or to build a symbolic framework for constraint representation for the full scope of optimization  
875 problems that concern us (Aubin, 2002). For LLM systems that generate optimization routines,  
876 a symbolic formalism capturing the full spectrum of objective functions and constraint types is  
877 essential for expressivity and correctness.878 B.2 LANGUAGES AND DEFINABILITY  
879

880 Following Marker (2002), we recall the definition of languages, structure, and definability.

881 **Definition B.1.** *A first-order language  $\mathcal{L}$  is given by specifying the following data:*882  
883 1. *a set of function symbols  $\mathcal{F}$  and positive integers  $n_f$  for each  $f \in \mathcal{F}$  describing the arity of  
884 the function;*  
885  
886 2. *a set of relation symbols  $\mathcal{R}$  and positive integers  $n_R$  for each  $R \in \mathcal{R}$ ;*  
887  
888 3. *a set of constant symbols  $\mathcal{C}$* 889 *An  $\mathcal{L}$ -structure  $\mathcal{M}$  is given by the following data:*890  
891 1. *a nonempty set  $M$  called the universe, domain, or underlying set of  $\mathcal{M}$ ;*  
892  
893 2. *a function  $f^{\mathcal{M}} : M^{n_f} \rightarrow M$  for each  $f \in \mathcal{F}$ ;*  
894  
895 3. *a set  $R^{\mathcal{M}} \subset M^{n_R}$  for each  $R \in \mathcal{R}$ ;*  
896  
897 4. *an element  $c^{\mathcal{M}} \in M$  for each  $c \in \mathcal{C}$ .*898  *$f^{\mathcal{M}}$ ,  $R^{\mathcal{M}}$  and  $c^{\mathcal{M}}$  are referred to as the interpretations of their respective symbols. The set of  $\mathcal{L}$ -terms  
899 is the smallest set  $\mathcal{T}$  such that*900  
901 1.  *$c \in \mathcal{T}$  for each  $c \in \mathcal{C}$ ;*  
902  
903 2. *each variable symbol  $v_i \in \mathcal{T}$  for  $i=1,2,3,\dots$ ;*  
904  
905 3. *if  $t_1, \dots, t_{n_f} \in \mathcal{T}$  and  $f \in \mathcal{F}$ , then  $f(t_1, \dots, t_{n_f}) \in \mathcal{T}$ .*906 *We say  $\phi$  is an atomic-formula if  $\phi$  is either*907  
908 1.  *$t_1 = t_2$  for terms  $t_1, t_2$ ;*  
909  
910 2.  *$R(t_1, \dots, t_{n_R})$  for terms  $t_1, \dots, t_{n_R}$ .*911 *The set of  $\mathcal{L}$ -formulas is the smallest set  $\mathcal{W}$  containing the atomic formulas and closed under logical  
912 connectives and quantifiers, i.e.*913  
914 1. *if  $\phi \in \mathcal{W}$ , then  $\neg\phi \in \mathcal{W}$*   
915  
916 2. *if  $\phi, \psi \in \mathcal{W}$ , then  $\phi \wedge \psi$  and  $\phi \vee \psi$  are in  $\mathcal{W}$ ;*  
917  
918 3. *if  $\phi$  in  $\mathcal{W}$ , then  $\forall v_i \phi \in \mathcal{W}$  and  $\exists v_i \phi \in \mathcal{W}$ .*919 *Given an  $\mathcal{L}$ -structure  $\mathcal{M}$ , a subset  $X \subseteq M^n$  is definable if  $X = \{x \in M^n : \varphi(x)\}$  for some  
920 first-order  $\mathcal{L}$ -formula  $\varphi$  with parameters from  $\mathcal{M}$ .*

918 Let  $\mathcal{L}_{\text{or}} := \{0, 1, +, \cdot, \leq\}$  be the language of ordered rings. Let  $\mathcal{L}_{\text{exp}} := \mathcal{L}_{\text{or}} \cup \{\exp\}$  be the  
 919 language of ordered exponential rings (*E-rings* in the sense used in this paper). We write  $\mathbb{R}_{\text{or}}$  for the  
 920 real field in  $\mathcal{L}_{\text{or}}$  and  $\mathbb{R}_{\text{exp}}$  for  $(\mathbb{R}; 0, 1, +, \cdot, \leq, \exp)$ . For restricted analytic functions, write  $\mathbb{R}_{\text{an}}$ , and  
 921 for the expansion by the (unrestricted) exponential function,  $\mathbb{R}_{\text{an}, \exp}$ .

922 Further, we recall the chain of definitions for real-closed fields and semi-algebraic sets:  
 923

925 **Definition B.2.** *For any of the languages  $\mathcal{L}$  whose underlying universe is a field  $F$ , we say  $F$  is real-  
 926 closed if there is a total order  $<$  on  $F$  such that every positive element of  $F$  has a square root in  $F$  and  
 927 every polynomial of odd degree with coefficients in  $F$  has at least one root in  $F$ . The canonical real-  
 928 closed field is the field of real numbers,  $\mathbb{R}$ . For any real-closed field  $F$ , a set  $\mathcal{W} \subset F^n$  is said to be  
 929 basic closed semialgebraic set if it is a finite intersection of sets defined by polynomial inequalities of  
 930 the form  $p_i(X_1, \dots, X_n) \geq 0$  for  $p_i \in F[X_1, \dots, X_n]$ , so that  $\mathcal{W} = \{\vec{x} \in F^n \mid \bigwedge_{i \in I_{\mathcal{W}}} p_i(\vec{x}) \geq 0\}$ ,  
 931 while a general semialgebraic set is a finite Boolean combination of basic closed algebraic sets  
 932 (Netzer, 2016; Marker, 2002).*

933  
 934  
 935 With respect to our semi-algebraic set, this means  $\varphi(a)$  satisfies the implied (in)equalities expressed  
 936 in formula  $\varphi$ .  
 937

938 In theory, optimization problems rely on feasible sets, and in practice feasible sets and the objective  
 939 functions are often precisely those that are definable in terms from the first-order language of ordered  
 940 commutative rings  $\mathcal{L}_{\text{or}} = \langle 0, 1+, \cdot, < \rangle$ . In our expansion of the language  $\mathcal{L}_{\text{or}}$  to include a function  
 941 symbol  $E$ , or  $\exp$ , we want to capture semi-algebraic sets that definable in logarithmic-exponential  
 942 polynomial terms, and provide sentences that allow us to interpret  $E$  as the standard real-valued  
 943 exponential function. We will summarize below the model theoretic reasoning as to why this is  
 944 both desirable for optimization by SDP, as well as how o-minimal geometry allows us to retain the  
 945 decidability necessary to determine feasibility.

946 As mentioned before, it is immediate that there is a bijective correspondence between the polynomials  
 947 in any given ordered ring  $R[X_1, \dots, X_m]$  with indeterminates  $X_i$ , and the terms generated from  
 948  $\mathcal{L}_{\text{or}}$  when adding the constants of  $R$  to the  $\mathcal{L}_{\text{or}}$  (without any extensions of the language, the terms  
 949 are precisely in correspondence with polynomials in  $\mathbb{Z}[X_1, \dots, X_m]$ ) (Marker, 2002).

950 Expanding on E-rings in greater detail and following (Van Den Dries, 1984), an *E-ring* is a pair  
 951  $(R, E)$  where  $R$  is a ring with unity, denoted by 1, and a map  $E : (R, +) \rightarrow R_u^\times$ , such that  
 952  $E(0) = 1$ , and which maps the additive group structure of  $R$  to the multiplicative group of units of  
 953  $R$ , so that  $E(x+y) = E(x) \cdot E(y)$ . Given an E-ring  $(R, E)$ , the ring of *E-polynomial* in indeterminates  
 954  $X_1, \dots, X_m$  over  $R$ , is denoted by  $R[X_1, \dots, X_m]^E$ , and has the structure of a group ring over the  
 955 polynomial ring  $R[X_1, \dots, X_m]$ . The additive group structure of the E-polynomial ring is given by:

956  $R \oplus \bigoplus_{k=0}^{\infty} A_k$ , where  $A_k$  is recursively defined in terms of  $A_k$ , group homomorphisms  $E_k$ , and rings  
 957  $R_k$  such that  $R_k \subset R_{k+1}$  and  $E_{k+1}$  is a functional extension of  $E_k$ , which is a group homomorphism  
 958 from the additive group  $R_k$  to the group of units in  $R_{k+1}$ . In particular, each  $E_k : R_k \rightarrow R_{k+1}^\times$   
 959 is a group-homomorphism sending each  $r \in R_k$  written as  $r = r' + a$  such that  $r' \in R_{k-1}$  and  
 960  $a \in A_k$  to the element  $E_{k-1}(r') \exp(a)$  in  $R_{k+1}$ , such that  $R_{k+1} := R_k[\exp(A_k)]$ , the group ring  
 961 of  $\exp(A_k)$  over  $R_k$ . In turn,  $A_k$  is defined to be an  $R_k$ -submodule of  $R_{k+1}$  freely generated by  
 962  $\exp(a)$  with  $a \in A_k$  for  $a \neq 0$ , which establishes  $R_{k+1} = R_k \oplus A_{k+1}$  as additive groups. In turn,  
 963 the underlying polynomial ring  $R[X_1, \dots, X_m]^E$  is taken as the algebraic object  $\lim_{\rightarrow} R_k = \bigcup_{k \in \mathbb{N}} R_k$ .

964  
 965 Of particular interest, any polynomial  $p \in R[X_1, \dots, X_m]^E$  will correspond to a formal term  
 966  $t_p \in R \oplus \bigoplus_{k=0}^{\infty} A_k$ , which in turn is an element  $t_p \in R_k$ , where  $k$  is of *height* of  $t_p$ , with  $k$  the  
 967 integer where  $t_p \in R_k \setminus R_{k-1}$ . Intuitively, the height is the maximum number of embedded expon-  
 968 entiations appearing in the term  $t_p$ . Real exponential rings introduce a compositional algebraic  
 969 structure capable of representing optimization constraints involving both polynomials and expon-  
 970 entials in a uniform symbolic language. This algebraic expressiveness invites a deeper inquiry into the  
 971 geometric behavior of such constraints.

972 B.3 QUANTIFIER ELIMINATION AND DECIDABILITY IN REAL CLOSED FIELDS  
973974 We now describe two foundational theorems for our purposes: Theorems B.3 and B.4, (see Seiden-  
975 berg (1954) and Collins (1976) for full proofs and details).976 **Theorem B.3** (Tarski-Seidenberg). *The complete first-order theory  $\text{Th}(\mathbb{R}_{\text{or}})$  of real closed ordered  
977 fields admits quantifier elimination. In particular, any first-order sentence over  $(\mathbb{R}; 0, 1, +, \cdot, \leq)$  is  
978 effectively reducible to a quantifier-free one, hence  $\text{Th}(\mathbb{R}_{\text{or}})$  is decidable.*979 **Theorem B.4.** *There is an effective procedure, cylindrical algebraic decomposition (CAD), that  
980 performs quantifier elimination over  $\mathbb{R}_{\text{or}}$ . The worst-case complexity is doubly-exponential in the  
981 number of variables, and this is optimal in general.*982 Theorem B.3 implies that the image of any semi-algebraic set under a polynomial map remains  
983 semi-algebraic under coordinate projection, which is the geometric core of quantifier elimination  
984 over the reals (Seidenberg, 1954). Concretely, existential quantifiers, and thus universal quantifiers  
985 under the identification that  $\forall x \varphi(x) \equiv \neg \exists x \neg \varphi(x)$ , over real variables can be eliminated effectively,  
986 reducing feasibility questions for semi-algebraic constraints to equivalent quantifier-free formulas  
987 that symbolic procedures can manipulate. Algorithmically, Collins' cylindrical algebraic decom-  
988 position (CAD) provides a constructive QE method: it decomposes  $\mathbb{R}^n$  into finitely many *cells* on  
989 which every input polynomial has a constant sign, enabling projection and elimination steps that  
990 decide first-order sentences in the language of ordered rings. (Collins, 1976) While CAD is worst-  
991 case doubly exponential in the number of variables, and this is unavoidable in general, its cell or  
992 sample-point outputs yield exact certificates of feasibility and optimality for low-dimensional or  
993 low-degree instances. Further, guarantees of decidability provide motivation for reliance on approx-  
994 imation methods which can be run more efficiently and for real-world applications often suffice  
995 given tolerable imprecision found within real-world implementations.996 In addition to decidable guarantees for feasibility, real algebraic geometry is furnished with many  
997 decision-to-optimization reductions and critical-point methods that recover algebraic minimizers by  
998 eliminating variables (and multipliers) from KKT-style conditions or by optimizing on QE-produced  
999 cells (Basu et al., 2006). These facts are what justify using semi-algebraic structure as a target for  
1000 our IR and validators: feasibility remains a decidable, tame geometric problem; optimality can  
1001 be certified exactly for small cases; and larger cases still benefit from the same logical invariants  
1002 (e.g., unit/convexity checks) even when solved numerically (Basu et al., 2006). We summarize these  
1003 observations for practical purposes in Corollary B.5.1004 **Corollary B.5** (Decision-to-optimization reduction). *Let  $f, g_i, h_j$  be polynomial terms, and define  
1005 a decision predicate  $P(t) := (\exists x) \bigwedge_i g_i(x) \leq 0 \wedge \bigwedge_j h_j(x) = 0 \wedge f(x) \leq t$ . Then  $P$  is decidable.*1006 The proof of Corollary B.5 follows immediately by Theorem B.3. As a consequence of Corollary  
1007 B.5, the global minimum  $\min\{f(x) : g_i(x) \leq 0, h_j(x) = 0\}$  is computable exactly by bracketing  
1008  $t$  and deciding  $P(t)$ , or by eliminating  $x$  from the KKT/global optimality conditions.1009 **Remark B.6** (What QE means in practice). *For low degrees, i.e. small  $n$ , CAD-based QE supplies  
1010 exact feasibility certificates, algebraic optima, and sample points for minimizers. For larger degrees,  
1011 best practices rely on numerical solvers. Nonetheless, these problems still retain the logical form to  
1012 guide validation.*1014 B.4 O-MINIMAL EXPANSIONS AND EXPONENTIALS  
10151016 Let  $\mathcal{L}$  contain a relation symbol  $<$ , and let  $\mathcal{M}$  be a dense linear order with respect to  $<$ . Then  $\mathcal{M}$   
1017 is *o-minimal* iff every definable subset  $S \subset M$  (with parameters) is a finite union of points and  
1018 intervals. Following Van den Dries (1998), o-minimality generalizes the semialgebraic setting once  $\mathcal{L}$   
1019 furnishes an ordered ring structure, so that *tame* sets behave like semialgebraic sets with robust  
1020 geometric control (monotonicity, cell decomposition, dimension). Practically, the connective tissue to  
1021 optimization is that tame geometry underlies semidefinite *relaxations* for polynomial optimization,  
1022 notably Lasserre's hierarchy: minimizing a polynomial  $f$  on a compact basic semialgebraic set  $\mathcal{W}$   
1023 is approached by a sequence of SDPs over moment cones with SOS certificates (Josz & Molzahn,  
1024 2018). Each level optimizes a linear functional subject to PSD moment/localizing matrices, encod-  
1025 ing polynomial nonnegativity by SOS constraints and yielding increasingly tight lower bounds, with  
many refinements for sparsity and large scale. Conceptually, this casts global polynomial optimiza-

1026  
 1027  
 tion over semialgebraic sets as an SDP sequence that exploits positivity, duality, and moment/SOS  
 1028  
 1029  
 1030  
 geometry (Netzer, 2016; Josz & Molzahn, 2018).

1031  
 1032  
**Theorem B.7** (O-minimality of  $\mathbb{R}_{\text{an}}$ ). *The structure  $\mathbb{R}_{\text{an}}$  (real field expanded by all restricted analytic functions) is o-minimal. Consequently, definable sets admit cell decomposition and the usual tame geometric properties. (van den Dries et al., 1994)*

1033  
 1034  
 1035  
 1036  
 1037  
 1038  
 O-minimality yields a calculus for tame sets and functions: finite cell decompositions, stratifications, and dimension control that mirror the semialgebraic case and support symbolic preprocessing (projection, variable elimination) in optimization pipelines (Van den Dries, 1998). As explored in Attouch et al. (2013), tameness also underpins algorithmic analysis, as for tame (e.g., semialgebraic/definable) objectives, descent and proximal methods will still admit global convergence guarantees via the Kurdyka–Łojasiewicz inequality. In particular, definability ensures the KL property and yields convergence rates and stability for a wide class of first-order schemes.

1039  
 1040  
 1041  
 1042  
 1043  
 1044  
 1045  
 1046  
 Real-world models often require exponentials (entropy, logistic regression, log-barriers, softmax/soft constraints), and we want to retain tameness while adding  $\exp$  (Wilkie, 1998). The key bridge we identify builds upon Wilkie’s results for exponential fields (principally Theorems B.8 and B.9): expansions of the real field by suitable analytic functions, including the (unrestricted) exponential, remain o-minimal or at least admit strong model-theoretic control, and so definable sets with exponentials inherit the same geometric regularity (cell decomposition, dimension, finiteness properties) (Wilkie, 1996). This provides the fundamental justification for our choice of *E-rings* as the target for our IR. We can model exponential terms while preserving tame geometry for symbolic reasoning and quantifier simplification (van den Dries et al., 1994).

1047  
 1048  
**Theorem B.8** (Wilkie’s Theorem). *The real exponential field  $\mathbb{R}_{\text{exp}}$  is o-minimal, and certain expansions by restricted Pfaffian functions are model complete. (Wilkie, 1996)*

1049  
 1050  
 1051  
 1052  
 1053  
 1054  
 1055  
 Following Wilkie (1999), o-minimality of  $\mathbb{R}_{\text{exp}}$  gives tame geometry for formulas involving real polynomials and exponentials: definable sets admit cell decomposition and controlled combinatorics, which is precisely the regularity needed to design and analyze robust optimization layers (e.g., screening infeasibility, preserving convexity, bounding active set changes). This directly supports our intended natural language to IR workflow that must manipulate exponential constraints without leaving the tame world.

1056  
**Theorem B.9.** (van den Dries et al., 1994) *The expansion  $\mathbb{R}_{\text{an,exp}}$  is model complete and o-minimal.*

1057  
 1058  
 1059  
 1060  
 1061  
 1062  
 1063  
 1064  
 1065  
 Theorems B.7 and B.9 are forms of a *weak* QE result. Whereas *strong* QE means every first-order formula in the given language is equivalent to a quantifier-free one in the *same* language, as in real closed fields, by contrast, there are two practically important senses of *weak* QE that arise in o-minimal expansions such as  $\mathbb{R}_{\text{an,exp}}$ . First, the *model-completeness* sense, where every formula is equivalent to an *existential* formula (no universal blocks or alternation) though not necessarily quantifier-free. Second, the *language-expansion* sense, where formulas become quantifier-free after enlarging the language by naming standard definable primitives (e.g., restricted analytic/Pfaffian pieces,  $\exp$ ), even if they are not quantifier-free in the strict base language (Van Den Dries, 1984; Wilkie, 1996; Van den Dries & Miller, 1996; Marker, 2002).

1066  
 1067  
 1068  
 1069  
 1070  
 1071  
 1072  
 1073  
 1074  
 1075  
 1076  
 1077  
 1078  
 1079  
 Either sense of weak QE enables a solver architecture resembling those employed by mixed-integer linear and cone programs with cutting planes and oracles, as the solution to an optimization problem is reduced to a  $\Sigma_1$ -sentence of the form  $\exists \vec{x} [f(\vec{x}) \leq t \wedge \varphi(\vec{x})]$ , where  $f$  is a (definable) objective function,  $\varphi$  is the collected formula describing the constraints, and  $t$  is the definable candidate term for an optimal solution (in this particular setting,  $f$  and  $t$  are scalar valued, but this can be further amended for vector-valued functions, and general definable functions). In particular, existential checks can be approached with feasibility oracles, and the optimization loop then is recast as repeated queries checking whether a current discrete assignment or candidate bound term for the objective function returns a *witness* to the defining formula for the objective modulo constraints (a sample point with the corresponding algebraic data), or a small unsatisfiable explanation, from which we can then deploy branch-and-bound and cut-generation loops as needed. Future work involving formal verification can post-certify winners via Positivstellensatz or KKT-based checks (see Netzer (2016) for details), following common OR workflows validating incumbent solutions instead of proving global optimality at every node. Further, in the weak QE setting, the polynomial term constraints can be cast into an answer-set program (ASP), where they act as external atoms that are evaluated by an oracle given the current ASP assignment to discrete choices and parameters symbols.

1080 For our pipeline, this distinction guides how we validate and compile  $\text{NL} \rightarrow \text{IR}$  instances. In the  
 1081 *semialgebraic* fragment, strong QE yields full elimination and decidability, which we exploit for  
 1082 exact feasibility/optimality on small cases and for sound structural checks more generally. In the  
 1083 *exponential* fragment,  $\mathbb{R}_{\text{an},\text{exp}}$  gives o-minimal tameness and model-completeness (weak QE). In  
 1084 this general setting, feasibility reduces to existential forms, projections preserve definability, and  
 1085 quantifiers can be simplified once the IR names the right primitives. O-minimality/model completeness  
 1086 do *not* by themselves yield decidability. The decidability of  $\text{Th}(\mathbb{R}_{\text{exp}})$  is *open*. Assuming a  
 1087 suitable real form of Schanuel’s conjecture, Macintyre and Wilkie show that  $\text{Th}(\mathbb{R}_{\text{exp}})$  is decidable;  
 1088 conversely, a *weak Schanuel* statement is equivalent to decidability (Wilkie, 1997).

1089 **B.5 CONSEQUENCES FOR OR MODELING**

1090 Reiterating our motivation, many optimization models used in OR and ML are definable in the lan-  
 1091 guages described above expressing tame theories: LP, QP, Second Order Cone Programs (SOCP),  
 1092 and SDP feasible regions are semialgebraic, and common exponential terms (such as entropy, log-  
 1093 likelihoods, log-barriers, softmax) live naturally in expansions with  $\exp$ . Working in an o-minimal  
 1094 expansion preserves geometric regularity (chiefly cell decomposition, dimension theory, stratifica-  
 1095 tions), which in turn supports symbolic preprocessing (namely projection and thus elimination) and  
 1096 robust certification prior to numerical solution (Denef & Van den Dries, 1988; Van den Dries &  
 1097 Miller, 1996). For polynomial parts, Lasserre-type hierarchies provide systematic SDP relaxations  
 1098 and certificates via moments/SOS, allowing us to transform global polynomial optimization over  
 1099 compact semialgebraic sets into a convergent sequence of tractable SDPs (Josz & Molzahn, 2018;  
 1100 Netzer, 2016).

1101 **Proposition B.10** (Standard OR cones are definable). *The feasible sets of LPs and QPs are semi-  
 1102 algebraic (linear or quadratic equalities/ inequalities). SOCP constraints  $\|Ax + b\|_2 \leq c^\top x + d$   
 1103 are semialgebraic. SDP constraints  $\{X \succeq 0\}$  are semialgebraic via principal minors. Exponen-  
 1104 tial/power cones (e.g.,  $y \exp(x/y) \leq z$ ,  $y > 0$ ) are definable in  $\mathbb{R}_{\text{exp}}$  and hence in  $\mathbb{R}_{\text{an},\text{exp}}$ .*

1105 Prop. B.10 justifies aiming the natural language to IR mapping at a formal language whose first order  
 1106 terms are composed of polynomials and  $\exp$ : we keep the cones that appear in OR, while staying  
 1107 within an o-minimal setting. This yields closure under projection and cell decomposition—key  
 1108 for IR-level checks (unit consistency, convexity screens, feasibility typing) and safe simplifications  
 1109 before calling a solver. For purely polynomial instances, moment/SOS machinery supplies SDP-  
 1110 based lower bounds and, in small cases, exact certificates.

1111 Concretely, the E-ring perspective provides a first-order language in which constraints take the form

$$\varphi(\mathbf{x}) \equiv t_p(\mathbf{x}) \geq 0,$$

1112 where  $t_p$  is a formal term built from polynomials and (possibly)  $\exp$ . Then, via the established corre-  
 1113 spondence,  $t_p$  is in one-to-one correspondence with a polynomial in an E-ring understood as nested  
 1114 compositions of elements of  $\mathbb{R}[X_1, \dots, X_m]$  and  $\exp$ . In this language, *feasibility* corresponds to the  
 1115 existential truth of a sentence, while *optimality* (e.g., minimality) requires restricted universal quan-  
 1116 tification. O-minimality provides the weak quantifier-elimination and geometric tameness needed so  
 1117 that such transformations remain within the definable universe (Wilkie, 1996; van den Dries et al.,  
 1118 1994; Marker, 2002). We reiterate and gather this observation in the following Corollary:

1119 **Corollary B.11** (Pragmatic E-ring policy). *If an  $\text{NL} \rightarrow \text{IR}$  mapping stays within  $\mathbb{R}_{\text{or}}$  ( namely for  
 1120 LP/QP/SOCP/SDP), feasibility/optimality are decidable and admit effective QE. If an  $\text{NL} \rightarrow \text{IR}$   
 1121 mapping stays within  $\mathbb{R}_{\text{an},\text{exp}}$ , we admit weak quantifier elimination.*

1122 Corollary B.11 follows principally from Theorems B.3 – B.4. When exponentials appear in terms,  
 1123 we will be working in  $\mathbb{R}_{\text{an},\text{exp}}$ , which by Theorems B.8–B.9 yields tame geometry and model com-  
 1124 pleteness. Although full decidability is not known in general, but the o-minimal setting supports  
 1125 robust validation and quantifier *simplification*, as for fragments that appear within  $\mathcal{L}_{\text{or}}$ , we will have  
 1126 decidability guarantees. This justifies our choice to target the E-ring fragment for modeling and  
 1127 training.

1128 We note that for terms in these languages With integer variables, the linear fragment (Presburger  
 1129 Arithmetic, i.e. an ILP) is decidable. If we allow for general polynomial relations over  $\mathbb{Z}$  though,  
 1130 this leads to undecidability as determined by Hilbert’s Tenth Problem. The proposed IR conforming  
 1131 to **Problem** therefore classifies integer problems and restricts to the linear case for exact certification.

1134 Finally, given a **Problem** instance in the semi-algebraic fragment, we may form  $P(t)$  as in Cor. B.5.  
 1135 Using CAD (or a specialized QE) to decide  $P(t)$  along a rational bisection on  $t$ , we can then ex-  
 1136 tract  $t^*$  and (optionally) a sample minimizer via the CAD cell containing the optimum. For larger  
 1137 instances, we may still use numerical solvers but keep IR-level feasibility/convexity/unit checks as  
 1138 first-order invariants. For the purposes of improving LLM-based systems then, reliably reconstr-  
 1139 ucting and building an IR that maps faithfully onto a **Problem** instance is a sufficient goal for operations  
 1140 research optimization from natural language.

## C CODE

### C.1 GENERATING A PROBLEM

We examine in greater detail the pseudo-code description provided in the main body of the paper, so that readers can examine the code we have provided in the supplementary materials with greater clarity.

Although we provide methods to pre-load problem configurations, by default we randomly generate problems from scratch with a number of fixed hyperparameters, both in the class `__init__` method, as well as within various functions.  $n_{vars}$ , the number of variables, and  $n_{res}$ , the number of resources are generated from a mixed model of three separate integral uniform distribution and hypergeometric distributions respectively, each determined by three hyperparameters. Because minimization problems and maximization problems can be freely converted between each other by multiplying the objective function by  $-1$ , we used a fair Bernoulli random variable to determine the problem goal.

We only generate arbitrary polynomials in the base E-ring in order to compare the complexity of our linear and quadratic programs with the linear programs found in the NL4OPT dataset. The objective function is generated by the `_gen_function` method of the problem class, and randomly selects monomials by the grading of the degree from combinations of the variables. Coefficients for each monomial term are either drawn uniformly from  $Unif_{\mathbb{Z}}(1, 10)$  or  $Unif_{\mathbb{Z}}(1, 10) + Unif(0, 1)$  depending on a fixed hyperparameter for whether we prefer integer coefficients only. We have opted to default solely to integer coefficients.

Constraints are organized into four cases: non-negativity, type, lower-bound, and upper-bound. The non-negativity constraints were implemented to ensure that all variables are to be non-negative. The type constraints are used to enforce cases where variables must be integral - if any variable is integral, the problem becomes mixed-integer unless all variables are integral, in which case it becomes an integer program. Lower- and upper-bound constraints were further sorted into several resources that are randomly determined by a class hyperparameter.

Lower- and upper-bound constraints consist of either two, three or all variables, and either determine a minimum or maximum proportion for those constraints or otherwise correspond to a minimum or maximum allocation of resources set by a budget cap that is by default randomly generated along with the scalar coefficients for each resource. The coefficients appearing in both budget constraint types are randomly generated by default with the `_gen_resources_dict` method unless users provide explicit resource functions when initializing a **Problem** instance. The default method first generates three random integers:  $a$  and  $b$  are uniformly selected from between 2 and 10, while  $t$  is drawn  $t \sim 2 + HGeo(a, b, n)$ ,  $n \sim Unif(a, a + b)$ . Afterward, with a fixed hyperparameter value of `resources_split_parameter`, which we default to  $.2$ , if a variable is a uniform random variable from the unit interval is greater than this hyperparameter, then the coefficients were drawn from a random integer matrix whose components were uniformly drawn between 1 and 3t; otherwise, coefficients were drawn from a real-valued random matrix whose components were drawn uniformly between 1 and 3t and rounded to the hundredth's place. Similarly, budget cap 'seeds'  $b_{res}$  are used when generating constraints as a random vector drawn uniformly from between L and U, where  $L \sim Unif_{\mathbb{Z}}(2tn_{vars}, 5tn_{vars})$  and  $U \sim Unif_{\mathbb{Z}}(6tn_{vars}, 20tn_{vars})$ . For the resource constraints, bounds are then uniformly drawn from  $Unif_{\mathbb{Z}}(\lfloor b_{res}/(3n_{vars}) \rfloor, \lfloor b_{res}/(n_{vars}) \rfloor)$  for lower bounds, and  $Unif_{\mathbb{Z}}(\lfloor b_{res}/(n_{vars}) \rfloor, b_{res})$  for upper bounds.

The number of pairs and triples that appear in the lower- and upper-bound constraints for each resource type are random variables, which depend on whether the goal is to maximize or minimize

1188 the objective function. We describe the minimize case below, noting that we reverse the parameters  
 1189 for the lower and upper bounds when the goal is to maximize the objective function. These variables  
 1190 are  $n_{lpair,res}$ ,  $n_{ltriangle,res}$ ,  $n_{upair,res}$ ,  $n_{utriangle,res}$  and in the minimization case are drawn as follows:  
 1191

$$\begin{aligned}
 1192 \quad r &\sim \text{Unif}_{\mathbb{Z}}(n_{vars}, \binom{n_{vars}}{2}) & s &\sim \text{Unif}_{\mathbb{Z}}(0, \binom{n_{vars}}{3}) \\
 1193 \quad t &\sim \text{Unif}_{\mathbb{Z}}(0, \binom{n_{vars}}{2}) & u &\sim \text{Unif}_{\mathbb{Z}}(0, \binom{n_{vars}}{3}) \\
 1194 \quad n_{lpair,res} &\sim HGeo(r, \binom{n_{vars}}{2}), \lfloor \frac{3r}{2} \rfloor \\
 1195 \quad n_{ltriangle,res} &\sim HGeo(s, \binom{n_{vars}}{3}), \lfloor \frac{3s}{2} \rfloor \\
 1196 \quad n_{upair,res} &\sim HGeo(t, \binom{n_{vars}}{2}), \lfloor \frac{3t}{2} \rfloor \\
 1197 \quad n_{utriangle,res} &\sim HGeo(u, \binom{n_{vars}}{2}), \lfloor \frac{3u}{2} \rfloor
 \end{aligned}$$

1200 Having now formed the symbolic representation of the problem, we now randomly select the semantic  
 1201 template for the natural language description of the problem. The options are conditioned on the `problem_type` attribute, and are drawn uniformly from one of three possible lists. Then, depending on the number of variables in the symbolic problem, we uniformly select from the semantic template determined by the `semantic_problem_type` attribute, we select corresponding variables from a list of possible semantic variable names with the `_gen_sym_vars` method. Similarly, we select resource names in a similar fashion with the `_gen_nl_resources_dict` method. After this, we construct the word problem through substitution and random selection of syntactically correct connecting phrases. We explain this in greater detail in the Data section of this Appendix.

1202 We also convert the symbolic description of the problem into a corresponding Python file that implements the problem in Gurobi. We default to Gurobi in this paper, as we have restricted ourselves for the time being to quadratic problems. However, when dealing with arbitrary terms drawn from an E-ring, we would default to an implementation in SymPy. This will require a future iteration of the Problem class to substitute the `gurobi_code` attribute with a generic `code` attribute name. With respect to the translation of the problem into Gurobi code, this amounts to properly providing the Problem's symbolic attributes for the objective function, variables, and constraints as they appear in attribute dictionaries. Finally, we run the code produced with the `_run_gurobi_code` method, storing it as the `problem_solution` attribute.

## 1227 C.2 RUNNING EXPERIMENTS

1228 We have included a detailed README file in our supplementary materials. We advise researchers  
 1229 consult this first before running any experiments. We also advise researchers to properly provide  
 1230 their own API keys for each respective model, or appropriately adjusting the LLM Service templates  
 1231 that we have provided in order to ensure smooth operation. We include two functions to run experiments  
 1232 on the NL4OPT data and the NLMOptimizer problems with the `baseline_llm_nlp4opt_qa.py`  
 1233 and `baseline_llm_problem_answer.py` files respectively. These functions run locally within the sup-  
 1234 plementary materials folder and interact with the Data and Dataset folders also included in the  
 1235 Supplementary materials folder.

1236 For the NL4OPT data, we convert the stored problem description into our Problem class format, pri-  
 1237 marily by preloading in the symbolic representation of the problem and natural language description  
 1238 of the problem, before also generating appropriate Gurobi code. All experiments are run through an  
 1239 interface between the Problem attributes and various LLM Services. We use two prompts for our  
 1240 experiments, one baseline, and one expanded to further prompt the LLM Service to conform to a  
 1241 consistent symbolic representation.

---

1242 **Algorithm 1** Problem Initialization

1243 **Require:** Sampling configuration (distributions for variables, constraints, objectives), desired se-

1244 mantic problem type, solver parameters, and optionally resource functions.

1245 **Ensure:** Symbolic problem, natural language description, solver-compatible code, and solver out-

1246 put.

1247 1: Sample number of variables using multi-branch distribution

1248 2: Sample number of resources using hypergeometric distribution

1249 3: Generate resource matrix and budget caps and store.

1250 4: Determine problem type by maximum Degree, maximum height, Variable type (continuous,

1251 mixed, or integer), and Goal (maximize or minimize)

1252 5: Generate term space for objective function

1253 6: Construct symbolic objective from randomly sampled monomials and coefficients

1254 7: Generate constraint set by : 1) Non-negativity, integrality, upper and lower bounds including

1255 tradeoff and budget constraints

1256 8: Assign semantic problem type based on variable type.

1257 9: Sample variable names and resource descriptions from semantic type

1258 10: Generate natural language mappings for Variables, Resources, Constraints, and Objective func-

1259 tion, and store.

1260 11: Compile symbolic and natural language problem descriptions

1261 12: Translate to solver-executable code and store (e.g., Gurobi)

1262 13: Run solver-executable code and store solution

1263 14: Return Problem object

---

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295 We use the following as to generate our two prompts, with the condition `include_sym` to trigger  
the expanded prompt:

1296  
1297

**Prompt Template**

```

1298 "PURPOSE: I need you to solve an optimization problem,
1299 outputting Gurobi code that captures the problem description
1300 and provides a solution, or otherwise indicates the problem
1301 is infeasible.
1302 CONTEXT: I have the following variables to consider:
1303 {problem.problem_variables} which have the following
1304 resources/attributes that I need to deal with:
1305 {problem.nl_resources_dict}
1306 ROLE: You are a consulting team of business analysts,
1307 operations researchers, and programmers who will convert my
1308 natural language description of an optimization problem into
1309 functional Gurobi code that answers my problem.
1310 INPUT: I need to {problem.goal} the following objective
1311 function : {problem.objective_function_statement} subject
1312 to the following constraints:
1313 for constraint in problem.problem_constraints: "*"
1314 {constraint}
1315 "OUTPUT:"
1316 if include_sym: "In order to convince me that the code you
1317 are producing is correct, I also need to have a symbolic
1318 representation of the problem showing me that you have
1319 converted the description above into an appropriate symbolic
1320 representation of the optimization problem. This consists of
1321 a pairs of variables in symbolic notation for the first item
1322 in the pair of the form 'x1', 'x2', and so on, and the second
1323 item of the pair being the natural language object appearing
1324 in the problem description; the objective function rendered
1325 as an algebraic term where all natural language objects are
1326 substituted for the corresponding symbolic variable; and
1327 the list of semi-algebraic constraints where the natural
1328 language object is substituted with its symbolic variable
1329 counterpart. Return this solution in a code bloc encased
1330 as '''json {dict({"sym_variables": [("x#i", "object#i")], "objective_function": "objective function description with sym variables", "constraints": ["constraint", ]})} ''' " fi
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
252
253
254
255
256
257
258
259
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
25310
25311
25312
25313
25314
25315
25316
25317
25318
25319
25320
25321
25322
25323
25324
25325
25326
25327
25328
25329
25330
25331
25332
25333
25334
25335
25336
25337
25338
25339
25340
25341
25342
25343
25344
25345
25346
25347
25348
25349
25350
25351
25352
25353
25354
25355
25356
25357
25358
25359
253510
253511
253512
253513
253514
253515
253516
253517
253518
253519
253520
253521
253522
253523
253524
253525
253526
253527
253528
253529
253530
253531
253532
253533
253534
253535
253536
253537
253538
253539
253540
253541
253542
253543
253544
253545
253546
253547
253548
253549
253550
253551
253552
253553
253554
253555
253556
253557
253558
253559
253560
253561
253562
253563
253564
253565
253566
253567
253568
253569
253570
253571
253572
253573
253574
253575
253576
253577
253578
253579
253580
253581
253582
253583
253584
253585
253586
253587
253588
253589
2535810
2535811
2535812
2535813
2535814
2535815
2535816
2535817
2535818
2535819
2535820
2535821
2535822
2535823
2535824
2535825
2535826
2535827
2535828
2535829
25358210
25358211
25358212
25358213
25358214
25358215
25358216
25358217
25358218
25358219
25358220
25358221
25358222
25358223
25358224
25358225
25358226
25358227
25358228
25358229
25358230
25358231
25358232
25358233
25358234
25358235
25358236
25358237
25358238
25358239
25358240
25358241
25358242
25358243
25358244
25358245
25358246
25358247
25358248
25358249
25358250
25358251
25358252
25358253
25358254
25358255
25358256
25358257
25358258
25358259
253582510
253582511
253582512
253582513
253582514
253582515
253582516
253582517
253582518
253582519
253582520
253582521
253582522
253582523
253582524
253582525
253582526
253582527
253582528
253582529
253582530
253582531
253582532
253582533
253582534
253582535
253582536
253582537
253582538
253582539
253582540
253582541
253582542
253582543
253582544
253582545
253582546
253582547
253582548
253582549
253582550
253582551
253582552
253582553
253582554
253582555
253582556
253582557
253582558
253582559
253582560
253582561
253582562
253582563
253582564
253582565
253582566
253582567
253582568
253582569
253582570
253582571
253582572
253582573
253582574
253582575
253582576
253582577
253582578
253582579
253582580
253582581
253582582
253582583
253582584
253582585
253582586
253582587
253582588
253582589
253582590
253582591
253582592
253582593
253582594
253582595
253582596
253582597
253582598
253582599
2535825100
253582511
253582512
253582513
253582514
253582515
253582516
253582517
253582518
253582519
253582520
253582521
253582522
253582523
253582524
253582525
253582526
253582527
253582528
253582529
253582530
253582531
253582532
253582533
253582534
253582535
253582536
253582537
253582538
253582539
253582540
253582541
253582542
253582543
253582544
253582545
253582546
253582547
253582548
253582549
253582550
253582551
253582552
253582553
253582554
253582555
253582556
253582557
253582558
253582559
253582560
253582561
253582562
253582563
253582564
253582565
253582566
253582567
253582568
253582569
253582570
253582571
253582572
253582573
253582574
253582575
253582576
253582577
253582578
253582579
253582580
253582581
253582582
253582583
253582584
253582585
253582586
253582587
253582588
253582589
253582590
253582591
253582592
253582593
253582594
253582595
253582596
253582597
253582598
253582599
2535825100
253582511
253582512
253582513
253582514
253582515
253582516
253582517
253582518
253582519
253582520
253582521
253582522
253582523
253582524
253582525
253582526
253582527
253582528
253582529
253582530
253582531
253582532
253582533
253582534
253582535
253582536
253582537
253582538
253582539
253582540
253582541
253582542
253582543
253582544
253582545
253582546
253582547
253582548
253582549
253582550
253582551
253582552
253582553
253582554
253582555
253582556
253582557
253582558
253582559
253582560
253582561
253582562
253582563
253582564
253582565
253582566
253582567
253582568
253582569
253582570
253582571
253582572
253582573
253582574
253582575
253582576
253582577
253582578
253582579
253582580
253582581
253582582
253582583
253582584
253582585
253582586
253582587
253582588
253582589
253582590
253582591
253582592
253582593
253582594
253582595
253582596
253582597
253582598
253582599
2535825100
253582511
253582512
253582513
253582514
253582515
253582516
253582517
253582518
253582519
253582520
253582521
253582522
253582523
253582524
253582525
253582526
253582527
253582528
253582529
253582530
253582531
253582532
253582533
253582534
253582535
253582536
253582537
253582538
253582539
253582540
253582541
253582542
253582543
253582544
253582545
253582546
253582547
253582548
253582549
2
```

1350 objectives and constraints into symbolic form, and optionally generate executable code (e.g., for  
 1351 Gurobi) to solve the problem.

1352 The class also includes functionality for loading, saving, updating, and verifying the consistency of  
 1353 these symbolic-natural language mappings, enabling seamless transformation between interpretable  
 1354 and computable representations of structured optimization tasks. We have included it in the Supple-  
 1355 mentary Files for researchers.

1356

## 1357 D DATA

1359

### 1360 D.0.1 NLMOPTIMIZER DATA

1361

1362 We have included 1000 problems generated by the Problem class. We have included both the Prob-  
 1363 lem class as well as the problems in our Supplementary files. Beyond simple slot-filling, our tem-  
 1364 plates incorporate anaphora, ellipsis, proportional statements, and domain jargon, in line with other  
 1365 efforts such as Xiao et al. (2024) each pattern family is listed with multiple surface variants and  
 1366 examples to support inspection and reuse.

1367

1368 Each generated problem is paired with a natural language representation. This is achieved by insert-  
 1369 ing the parameters of the formal optimization model into pre-built natural language templates. These  
 1370 templates are structured to mimic human-authored prompts, and include redundant sentence options  
 1371 to introduce variation to natural language representations. Natural language prompts are assembled  
 1372 from templates using a combination of string formatting and regular expression substitution, ensur-  
 1373 ing that the resulting language is coherent and lexically diverse, while remaining traceable to the  
 1374 underlying mathematical representation of an optimization problem.

1375

1376 The generation of natural language representations begins with the random selection of a problem  
 1377 context, `semantic_type`. The value of `semantic_type` determines the possible values for re-  
 1378 sources, values, and some template text, ensuring that generated text is context-consistent and gram-  
 1379 matically correct. An initial batch of available contexts covers: Office supply budgeting, Household  
 1380 budgeting, Horticulture, Cybersecurity, staffing and management, Military force structure, Person-  
 1381 nel management, Personal diet, and Macronutrient intake.

1382

1383 A natural language statement is selected from a set of alternative wordings to describe the problem  
 1384 context, which places the agent in the role of the solver. Variables and resources are assigned labels  
 1385 selected at random from sets of available labels for the selected context. Statements describing  
 1386 resource costs for each variable are generated after randomly selecting from a set of context-relevant  
 1387 sentence templates, before substitution with the corresponding variable names, resource names, and  
 1388 resource costs. For any non-negativity constraints, statements are generated by randomly selecting  
 1389 from multiple sets of sentence fragments and joining them to produce a single grammatically-correct  
 1390 sentence asserting non-negativity. Upper- and lower-bound constraint statements are then generated  
 1391 by random selection from a context-relevant set of available sentence templates, followed by term  
 1392 substitution with the relevant resource and variable names. We consider proportion constraints to be  
 1393 part of the lower-bound constraints.

1394

1395 We denote by `int_type` whether the optimization problem is integer, mixed-integer, or linear.  
 1396 `int_type` is a randomly determined by two initializing parameters that partition the unit interval.  
 1397 This was done so that users may opt to force a particular problem type if they so desired. The  
 1398 `int_type` determines the statements that can be generated where the permissible solution value  
 1399 types are described. As with other methods, these statements are formed by combining selecting  
 1400 randomly from multiple sets of sentence fragments and joining them to produce grammatically-  
 1401 correct sentences conditioned on the `int_type`. Similarly, the natural language objective function  
 1402 statement is generated by first iterating through the symbolic objective function, and substituting  
 1403 variables, values, and operators with their previously selected semantic equivalents in natural lan-  
 1404 guage, before randomly selecting from a set of sentence fragment templates that are used to construct  
 1405 a full natural language statement. Finally, we collect all of the statements generated by the above  
 1406 methods are collected into a single problem statement.

1407

1408 We illustrate this process with the following example. Consider the case of a 2-variable 1-resource  
 1409 integer optimization problem with a goal of maximization, and objective function:  $2 \cdot x_0 + 3 \cdot x_1 +$   
 1410  $x_0 \cdot x_1$  and three constraints:

1404     1.  $x_0 \geq 0$   
 1405     2.  $x_1 \geq 0$   
 1406     3.  $x_0 + x_1 \leq 10$

1408  
 1409     Once the symbolic representation of the problem has been generated, the context `diet0` (personal  
 1410     diet) is selected at random.

```
1411     def _select_semantic_problem(self):
1412         """
1413             Selects a semantic problem type for determining.
1414             Supports up to 12 vars.
1415         """
1416
1417
1418         if self.problem_type["is_integer"] == 1: # Integer
1419             semantic_problem_options = [
1420                 "office supplies",
1421                 "family budget",
1422                 "gardening",
1423                 "network defense",
1424                 "force structure",
1425                 "diet0",
1426                 "personnel",
1427             ]
1428
1429
1430         elif self.problem_type["is_integer"] == 0.5: # Mixed integer
1431             semantic_problem_options = [
1432                 "diet0",
1433                 "diet1",
1434                 "personnel",
1435             ]
1436
1437
1438         elif self.problem_type["is_integer"] == 0: # continuous/linear
1439             semantic_problem_options = [
1440                 "diet0",
1441                 "diet1",
1442                 "personnel",
1443             ]
1444
1445         return random.choice(semantic_problem_options)
```

1446     Figure 12: Method to select `semantic_type`.  
 1447

1448  
 1449     A dictionary mapping variables to names, `sym_vars`, is constructed by selecting a random sample  
 1450     of size `num_vars` from the `diet0` context-relevant list of possible variable names.

1451  
 1452  
 1453  
 1454  
 1455  
 1456  
 1457

```

1458
1459     elif self.semantic_problem_type == "diet0":
1460         possible_semantic_vars = [
1461             "eggs",
1462             "apple pies",
1463             "cherry pies",
1464             "blueberry pies",
1465             "chicken breasts",
1466             "chicken thighs",
1467             "chicken drumsticks",
1468             "pickles",
1469             "kale salads",
1470             "fruit salads",
1471             "apples",
1472             "lemons",
1473             "hamburgers",
1474             "cheeseburgers",
1475             "rotisserie chickens",
1476             "steaks",
1477             "ravioli",
1478             "milkshakes",
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511

```

Figure 13: A portion of the possible\_semantic\_vars list.

```

# Build sym vars
sym_vars = dict()
for i in range(self.num_vars):
    var = f"x{i}"
    varname = selected_vars[i]
    vartokens = tokenized_vars[varname]
    common_objects = [
        token for token in vartokens if token_counts[token] > 1
    ] # all repeated tokens
    differentiators = [
        token for token in vartokens if token not in common_objects
    ] # all non-repeated tokens
    # Only keep differentiators if there's at least one common object
    if not common_objects:
        differentiators = []

    item_dict = {
        "full_phrase": varname,
        "differentiators": differentiators,
        "common_objects": common_objects,
        "target": None,
    }
    sym_vars[var] = item_dict

return sym_vars

```

Figure 14: Method to produce the sym\_vars dict.

1512 For our example, suppose that "apples" and "bananas" have been randomly selected. Likewise,  
 1513 a dictionary mapping resources to names, `nl_resources_dict`, is constructed by selecting a  
 1514 random sample of size `num_resources` from the `diet0` context-relevant list of possible resource  
 1515 names.

```
1516
1517     elif self.semantic_problem_type == "diet0":
1518
1519         possible_resources = [
1520             "dollar cost",
1521             "grams of protein",
1522             "grams of carbohydrates",
1523             "grams of fat",
1524             "grams of fiber",
1525             "milligrams of calcium",
1526             "milligrams of iron",
1527             "tastiness rating",
1528             "healthiness rating",
1529             "sourness index",
1530             "umami index",
1531         ]
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
```

Figure 15: A portion of the `possible_resources` list.

```
1543
1544     selected_resources = random.sample(possible_resources, self.num_resources)
1545     nl_resources_dict = dict()
1546     for i in range(self.num_resources):
1547         r = f"r{i}"
1548         item_dict = {
1549             "description": selected_resources[i],
1550             "upper_bound": self.resources_dict["budget_caps"][i],
1551         }
1552         for j in range(self.num_vars):
1553             var = f"x{j}"
1554             weight = self.resources_dict["weights"][j, i]
1555             item_dict[var] = weight
1556             nl_resources_dict[r] = item_dict
1557
1558
1559
1560
1561
1562
1563
1564
1565
```

Figure 16: Method to produce `nl_resources_dict`.

For our example, say we select "milligrams of calcium". Next we want to generate value statements  
 for each variable and resource. We randomly select a template from available options for each  
 resource and input names and values to produce the full statements.

```

1566     elif resource_name in [
1567         "grams of protein",
1568         "grams of carbohydrates",
1569         "grams of fat",
1570         "grams of fiber",
1571         "milligrams of calcium",
1572         "milligrams of iron",
1573     ]:
1574         weight_statements = [
1575             f"{varname} each contain {weight} {resource_name}. ",
1576             f"{varname} contain {weight} {resource_name}. ",
1577             f"There are {weight} {resource_name} in {varname}. ",
1578         ]
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

```

Figure 17: Weight statement options for milligrams of calcium.

One possible output would be: "apples each contain 2 milligrams of calcium." Non-negativity constraints are then generated by selecting sentence fragments, joining them, and inputting variable names.

```

1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632     for constraint_name, constraint in self.constraints["non_negativity"].items():
1633         for var in self.sym_vars.keys():
1634             varname = self.sym_vars[var]["full_phrase"]
1635             if var + " >= 0" in constraint:
1636                 if random.random() > 0.5: # Positive statement
1637                     imperatives = [
1638                         "You must use ",
1639                         "You must have ",
1640                         "You have to use ",
1641                         "You have to have ",
1642                         "There must be ",
1643                         "There has to be ",
1644                     ]
1645                     operator_statements = [
1646                         "no less than zero ",
1647                         "no less than 0 ",
1648                         "zero or more ",
1649                         "0 or more ",
1650                         "zero or greater ",
1651                         "0 or greater ",
1652                         "greater than or equal to zero ",
1653                         "greater than or equal to 0 ",
1654                         "a non-negative number of ",
1655                         "no less than zero ",
1656                         "no less than 0 ",
1657                     ]
1658                     sym_constraint = (
1659                         random.choice(imperatives)
1660                         + random.choice(operator_statements)
1661                         + varname
1662                         + ". "
1663                     )
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

```

Figure 18: Non-negativity statement segment construction.

1674 For instance, this could produce the statement: "You must use no less than zero apples." Upper- and  
 1675 lower-bound statements are assembled by inputting the appropriate values into templates.  
 1676

```

1677     elif resource in [
1678         "grams of carbohydrates",
1679         "grams of fat",
1680         "grams of fiber",
1681         "milligrams of calcium",
1682         "milligrams of iron",
1683     ]:
1684         ub_statement = random.choice(
1685             [
1686                 f"You must get at most {bound} {resource} of from {joined_vars}. ",
1687                 f"You must get no more than {bound} {resource} of from {joined_vars}. ",
1688                 f"You need to get no more than {bound} {resource} of from {joined_vars}. ",
1689                 f"At most {bound} {resource} can come from {joined_vars}. ",
1690                 f"You cannot get more than {bound} {resource} from {joined_vars}. ",
1691                 f"You can get up to {bound} {resource} from {joined_vars}. ",
1692             ]
1693         )

```

Figure 19: Upper-bound statement segment construction.

1694 In the instance of our example upper bound constraint, this may generate the statement "You must  
 1695 get at most 10 milligrams of calcium from apples and bananas." Integer constraints will then be  
 1696 generated for each variable by joining sentence fragment templates with resource names, similar to  
 1697 non-negativity constraints.

```

1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727

```

```

1728
1729
1730
1731
1732
1733
1734     if constraint["bound"] == "int":
1735         if random.random() > 0.33:
1736             imperatives = [
1737                 "You must use ",
1738                 "You have to use ",
1739                 "You must have ",
1740                 "You have to have ",
1741                 "There must be ",
1742                 "There has to be ",
1743                 "You are restricted to ",
1744                 "You're restricted to ",
1745                 "You are limited to ",
1746                 "You're limited to ",
1747             ]
1748             int_statements = [
1749                 "a whole number of ",
1750                 "a whole number amount of ",
1751                 "an integer number of ",
1752                 "an integer amount of ",
1753                 "a non-fractional amount of ",
1754                 "a nonfractional number of ",
1755             ]
1756             sym_constraints[f"{constraint_name}"] = (
1757                 random.choice(imperatives)
1758                 + random.choice(int_statements)
1759                 + varname
1760                 + ", "
1761             )
1762         )
1763     )
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775     Figure 20: Integer type statement segment construction.
1776
1777
1778
1779
1780
1781

```

1782 This may produce the statement: "You must use an integer amount of apples." For the objective  
 1783 function, we again select randomly from multiple sentence fragments to start the statement, then  
 1784 loop over the terms in the objective function to produce a natural language representation of the  
 1785 entire function and join that to our initial sentence fragment to produce the full statement.  
 1786

```

1787     def _gen_objective_function_statement(self):
1788
1789         objective_function_statement = random.choice(
1790             [
1791                 f"Your overall goal is to {self.goal} ",
1792                 f"Your goal is to {self.goal} ",
1793                 f"You need to {self.goal} ",
1794                 f"You need to {self.goal} the value of ",
1795                 f"We need to {self.goal} ",
1796                 f"We need to {self.goal} the value of ",
1797                 f"Altogether, we want to {self.goal} ",
1798             ]
1799         )
1800
1801         for vartuple in self.objective_function_data["term_list"]:
1802             if not isinstance(vartuple[0], str):
1803                 coef = round(vartuple[0], 2)
1804                 coef = str(coef)
1805             operator = random.choice(
1806                 [
1807                     " times the number of ",
1808                     " multiplied by the number of ",
1809                     " times the amount of ",
1810                     " multiplied by the amount of ",
1811                     " times the quantity of ",
1812                     " multiplied by the quantity of ",
1813                     " times the total number of ",
1814                     " times the quantity of ",
1815                 ]
1816             )
1817             varname = self.get_varname(vartuple[1])
1818             trailing_operator = random.choice([" plus ", " added to "])
1819             objective_function_statement += (
1820                 coef + operator + varname + trailing_operator
1821             )
1822             objective_function_statement = (
1823                 objective_function_statement[: -len(trailing_operator)] + ". "
1824             )
1825
1826
1827         return objective_function_statement
1828
1829
1830
1831
1832
1833
1834
1835

```

Figure 21: Method to produce natural language objective function statements.

For instance: "Your overall goal is to maximize 2 times the number of apples plus three times the  
 1832 number of bananas plus apples times the number of bananas." Lastly, all of the above will be joined  
 1833 into a singular statement. This begins by selecting a context-relevant initial problem statement  
 1834 establishing context.  
 1835

```

1836     self.semantic_problem_type == "diet0":
1837         problem_statement = random.choice([
1838             "I need you to help me develop a diet plan that provides me the correct balance of nutrients and other metrics.",
1839             "Help me refine my diet plan to provide the right balance of nutrients and other metrics.",
1840             "I need help determining a diet plan that balances nutrients and other metrics correctly.",
1841             "My diet is unbalanced, and I need to find a way to adjust my intake of several foods to get the right mix of nutrients.",
1842             "I need to optimize my nutrient intake."])
1843

```

Figure 22: Problem statement initial sentence list.

1843 Variable and resource statements are then appended iteratively.

```

1844
1845     # Add sym vars
1846     problem_statement += random.choice([
1847         "These are the foods that I need to adjust my intake of: ",
1848             "These are all of the foods that need to be balanced in my diet: ",
1849             "These are the foods from which I need to get all of my nutrients: ",
1850             "You need to find the right intake of ",
1851             "I eat ",
1852             "The only things I eat are ",
1853             "All I eat are ",
1854             "I only eat "
1855             "My diet must consist entirely of "])
1856
1857     i = 0
1858     for var in self.problem_variables:
1859         if i < self.num_vars-2:
1860             problem_statement += var + ", "
1861         elif i == self.num_vars-1:
1862             problem_statement += var + random.choice([" and ", ", and "])
1863         elif i == self.num_vars:
1864             problem_statement += var + ","
1865         i+=1
1866
1867     # Add sym resources
1868     problem_statement += random.choice([
1869         "I need to get the right balance of ",
1870             "I need to optimize my intake of ",
1871             "I need to balance my ",
1872             "I care about optimizing "])
1873
1874     i = 1
1875     for res in self.resources:
1876         if i < self.num_resources-2:
1877             problem_statement += res + ", "
1878         elif i == self.num_resources-1:
1879             problem_statement += res + random.choice([" and ", ", and "])
1880         elif i == self.num_resources:
1881             problem_statement += res + ","
1882         i+=1

```

Figure 23: Problem statement initial sentence list.

1871 And finally, each constraint statement and the objective statement are simply appended to the  
1872 problem\_statement string.

```

1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889

```

```

1890     # Add sym vars
1891     problem_statement += random.choice(["These are the foods that I need to adjust my intake of: ",
1892                                         "These are all of the foods that need to be balanced in my diet: ",
1893                                         "These are the foods from which I need to get all of my nutrients: ",
1894                                         "You need to find the right intake of ",
1895                                         "I eat ",
1896                                         "The only things I eat are ",
1897                                         "All I eat are ",
1898                                         "I only eat "
1899                                         "My diet must consist entirely of "])
1900
1901     i = 0
1902     for var in self.problem_variables:
1903         if i < self.num_vars-2:
1904             problem_statement += var + ", "
1905         elif i == self.num_vars-1:
1906             problem_statement += var + random.choice([" and ", ", and "])
1907         elif i == self.num_vars:
1908             problem_statement += var + ","
1909         i+=1
1910
1911     # Add sym resources
1912     problem_statement += random.choice(["I need to get the right balance of ",
1913                                         "I need to optimize my intake of ",
1914                                         "I need to balance my ",
1915                                         "I care about optimizing "])
1916
1917     i = 1
1918     for res in self.resources:
1919         if i < self.num_resources-2:
1920             problem_statement += res + ", "
1921         elif i == self.num_resources-1:
1922             problem_statement += res + random.choice([" and ", ", and "])
1923         elif i == self.num_resources:
1924             problem_statement += res + ","
1925         i+=1

```

Figure 24: Problem statement initial sentence list.

The fully constructed problem statement is then saved to the instantiated `Problem` object. In our example case, this would evaluate to: "I need to optimize my nutrient intake. My diet must consist entirely of apples and bananas. I need to optimize my intake of milligrams of calcium. Apples contain 2 milligrams of calcium. Bananas contain 3 milligrams of calcium. You must use no less than zero apples. You must use a positive number of bananas. You must get at most 10 milligrams of calcium from apples and bananas. You must use an integer amount of apples. You must use a whole number of bananas. Your overall goal is to maximize 2 times the number of apples plus three times the number of bananas plus apples times the number of bananas."

## E ADDITIONAL EXPERIMENTAL STATISTICS

Commenting first on Table 1, we display a model and prompt breakdown across five categories: end-to-end Accuracy, Adjusted end-to-end Accuracy, Compilation Error, Runtime Error, and Adjusted Runtime Error. The adjusted end-to-end Accuracy and run time scores apply to allow model outputs after performing a regex search to identify if there was a failure to import Gurobi properly, which required `import gurobipy`. Failures frequently resulted from attempts to run the incorrect expression: `import gurobi`. This happened overwhelmingly with LLaMa-4, and presents

Table 3: Summary of problems generated by integer, mixed integer, and continuous linear & quadratic programs

	ILP	IQP	MILP	MIQP	LP	QP
med Vars	4	3	4	4	4	4
Max Vars	8	7	9	7	9	8
med Res	2	2	2	2	2	2
Max Res	5	5	5	5	5	5
med Cons	33	27	32	32	32	36
Max Cons	1448	1021	1994	635	2009	1977
Feas	79	12	81	16	164	22
Total	184	47	198	113	372	86

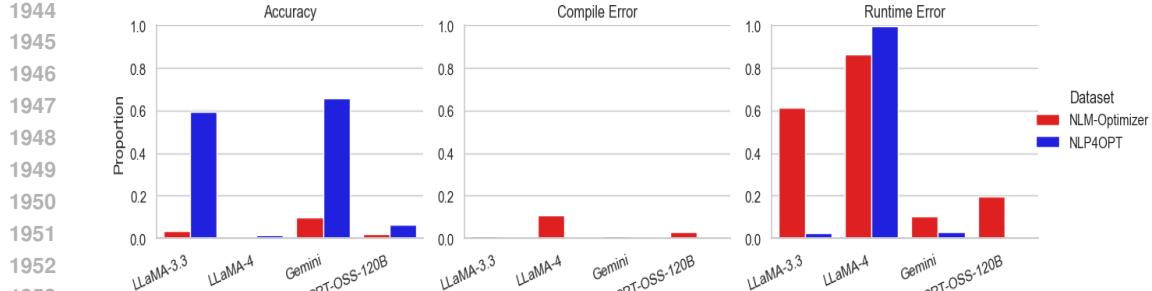


Figure 25: Summary results for each model on NL4OPT dataset vs NLMOptimizer dataset.

a curiosity more than anything, as even after this adjustment, LLaMa-4 overwhelmingly suffered from runtime errors. LLaMa-4 also overwhelmingly produced compilation errors when compared with the other models. Successful solutions include both cases where the model produced correct executable Gurobi code and cases where the model correctly identified the problem as infeasible.

In all Figures where models are end-labeled by '`_op_result`', we mean to indicate model results for the original problems produced for these experiments, with the additional end-label of '`_wsym`' indicating the expanded prompting with additional prompting for symbolic representation.

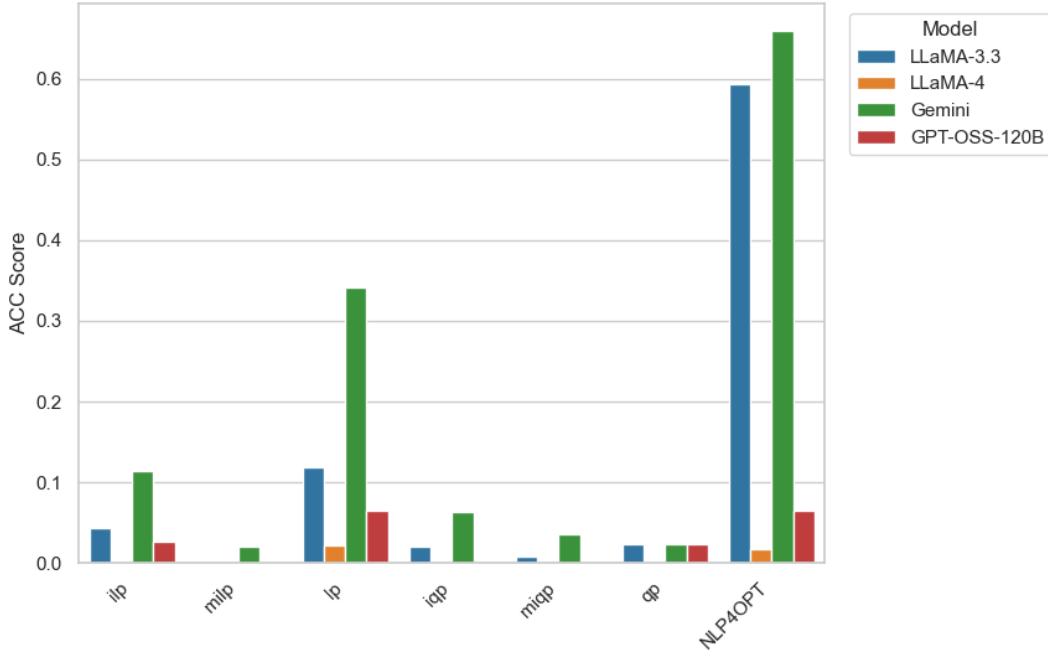


Figure 26: End-to-End accuracy scores by model and problem type.

Overall, the poor performance of LLaMa-4 applied to both datasets and both prompts. More interesting for our purposes is the collapse in performance for the other models when tasked with answering the NLMOptimizer problems. In Tables 4 to 10, we display the end-to-end Accuracy, Compilation Error, and Runtime Error by each model, prompt, and problem type. We collectively visualize the end-to-end Accuracy, Compilation Error, and Runtime Errors across model, prompt, and model type in Figures 26 – 31.

We consistently observe that the model performance collapses for any problem that is not a standard linear program. Most especially, we observe that even Gemini-1.5 suffers on the linear programs

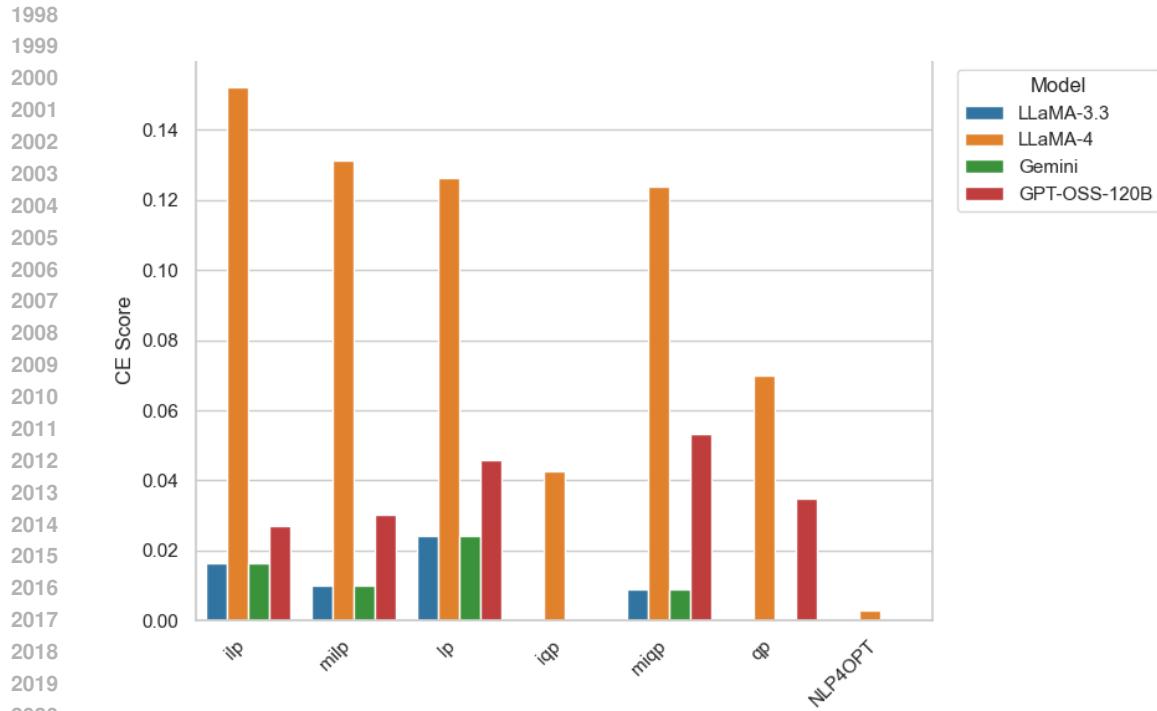


Figure 27: Compilation error scores by model and problem type.

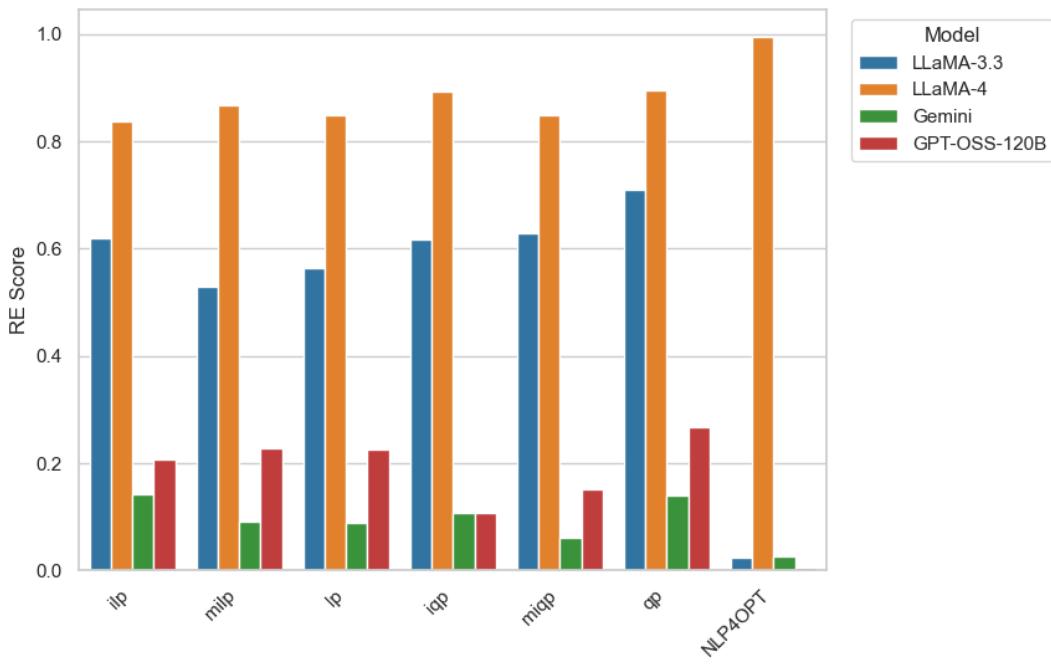


Figure 28: Runtime error scores by model and problem type.

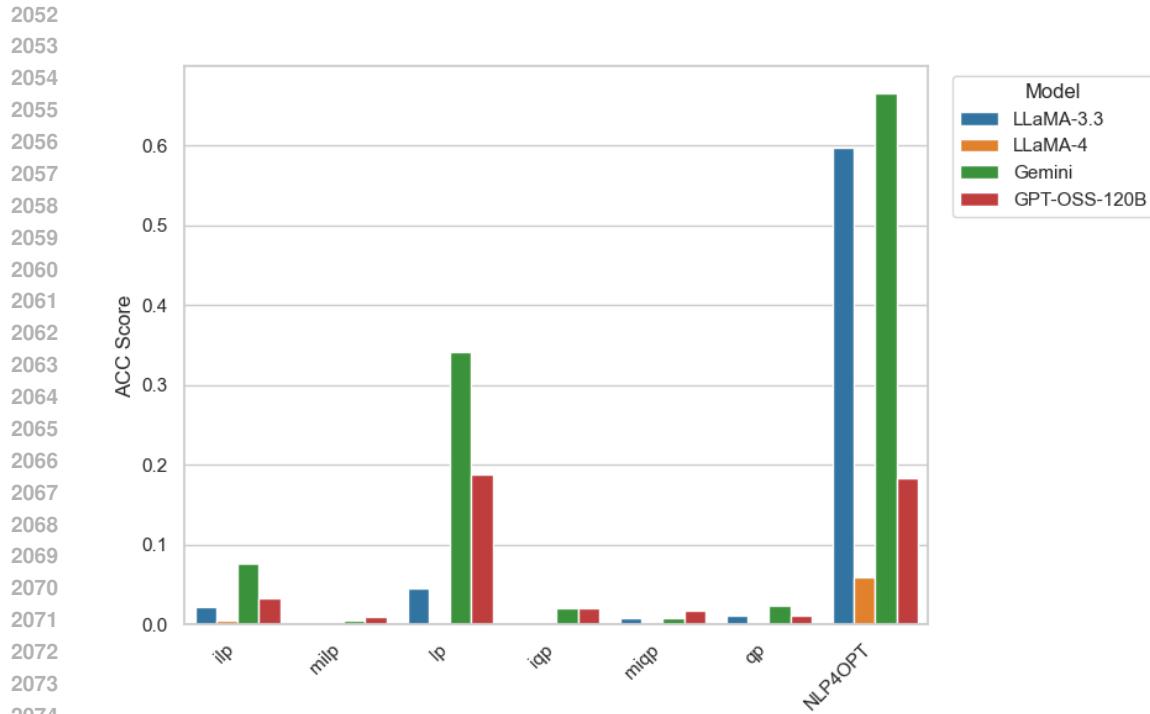


Figure 29: End-to-end accuracy scores by model with additional prompt and problem type.

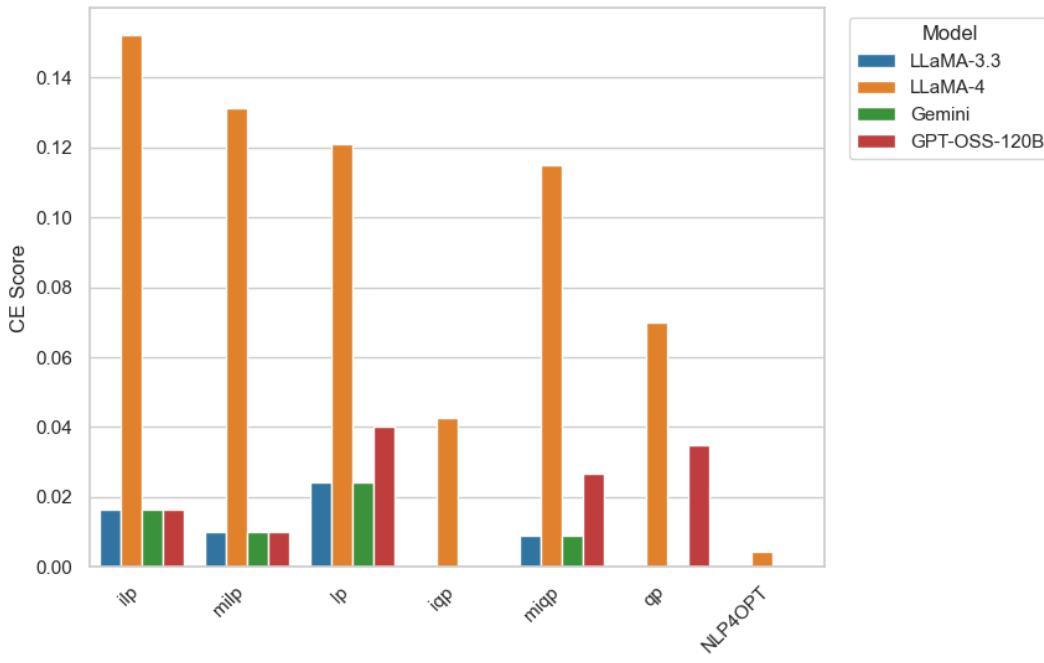


Figure 30: Compilation error scores by model with additional prompt and problem type.

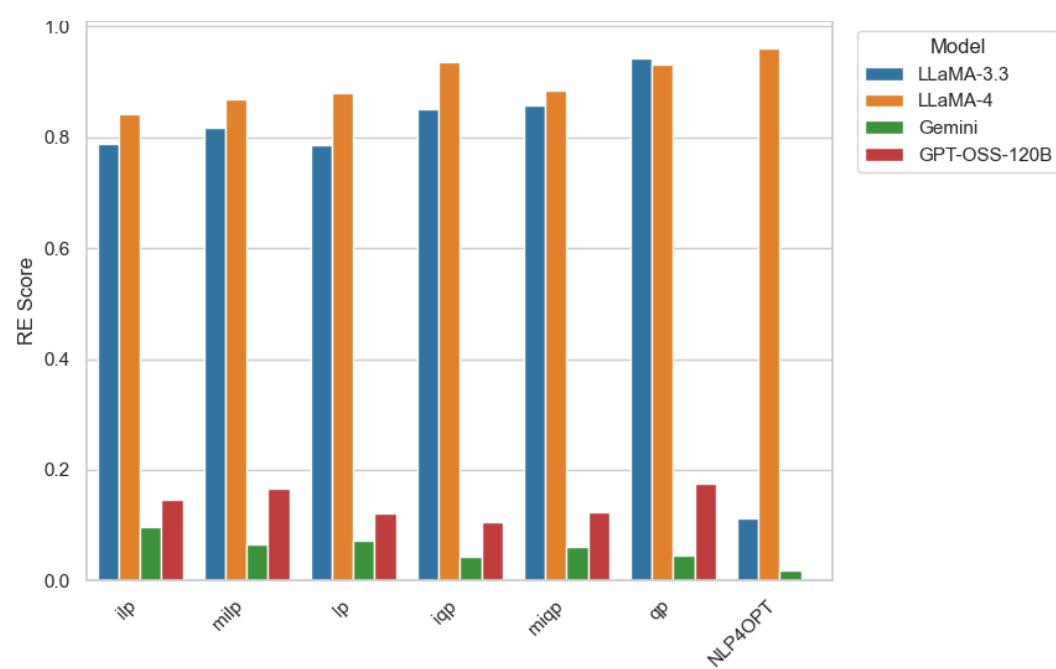


Figure 31: Runtime error scores by model with additional prompt and problem type.

Table 4: Problem type statistics for LLaMa-3.3 and NLMOptimizer problems

	ACC	CE	RE
ilp	0.043	0.016	0.620
milp	0.000	0.010	0.530
lp	0.118	0.024	0.565
iqp	0.021	0.000	0.617
miqp	0.009	0.009	0.628
qp	0.023	0.000	0.709

Table 5: Problem type statistics for LLaMa-4 and NLMOptimizer problems

	ACC	CE	RE
ilp	0.000	0.152	0.837
milp	0.000	0.131	0.869
lp	0.022	0.126	0.849
iqp	0.000	0.043	0.894
miqp	0.000	0.124	0.850
qp	0.000	0.070	0.895

Table 6: Problem type statistics for Gemini and NLMOptimizer problems

	ACC	CE	RE
ilp	0.114	0.016	0.141
milp	0.020	0.010	0.091
lp	0.341	0.024	0.089
iqp	0.064	0.000	0.106
miqp	0.035	0.009	0.062
qp	0.023	0.000	0.140

2160

2161

2162

Table 7: Problem type statistics for GPT-OSS and NLMOptimizer problems

2163

2164

2165

2166

2167

2168

2169

	ACC	CE	RE
ilp	0.027	0.027	0.207
milp	0.000	0.030	0.227
lp	0.065	0.046	0.226
iqp	0.000	0.000	0.106
miqp	0.000	0.053	0.150
qp	0.023	0.035	0.267

2170

2171

2172

2173

Table 8: Problem type statistics for LLaMa-3.3 and NLMOptimizer problems and expanded prompt

2174

2175

2176

2177

2178

2179

2180

	ACC	CE	RE
ilp	0.022	0.016	0.788
milp	0.000	0.010	0.818
lp	0.046	0.024	0.785
iqp	0.000	0.000	0.851
miqp	0.009	0.009	0.858
qp	0.012	0.000	0.942

2181

2182

2183

2184

Table 9: Problem type statistics for LLaMa-4 and NLMOptimizer problems and expanded prompt

2185

2186

2187

2188

2189

2190

2191

	ACC	CE	RE
ilp	0.005	0.152	0.842
milp	0.000	0.131	0.869
lp	0.000	0.121	0.879
iqp	0.000	0.043	0.936
miqp	0.000	0.115	0.885
qp	0.000	0.070	0.930

2192

2193

2194

2195

Table 10: Problem type statistics for Gemini and NLMOptimizer problems and expanded prompt

2196

2197

2198

2199

2200

2201

2202

2203

2204

2205

2206

	ACC	CE	RE
ilp	0.076	0.016	0.098
milp	0.005	0.010	0.066
lp	0.341	0.024	0.073
iqp	0.021	0.000	0.043
miqp	0.009	0.009	0.062
qp	0.023	0.000	0.047

2207

2208

2209

2210

2211

2212

2213

Table 11: Problem type statistics for GPT-OSS and NLMOptimizer problems and expanded prompt

	ACC	CE	RE
ilp	0.033	0.016	0.147
milp	0.010	0.010	0.167
lp	0.188	0.040	0.121
iqp	0.021	0.000	0.106
miqp	0.018	0.027	0.124
qp	0.012	0.035	0.174

generated under the NLMOptimizer framework, with end-to-end Accuracy below .35. Gemini-1.5 also managed to be inaccurate, but not due to code failure, whereas the LLaMa models both had high incidence rates of run time errors, with LLaMa-4 also suffering from high compilation errors above 10% for every problem type but quadratic programs and integer quadratic programs. Conversely, the LLaMa models consistently had trouble with all forms of quadratic programs, with Runtime Errors above 80% as seen in Figure 28.

We then consider a granular breakdown across our three categories by the number of problem variables and problem types, with each model and prompt pairing being visualized in its own respective figure (across Figures 4-11). Specifically, in Figure 4, we observe that LLaMa-3.3 performs poorly for all problem types, with scores generally worsening as the number of variables increases across problem types, although curiously end-to-end Accuracy scores are parabolic for classical linear programs, and bottom out at 5 variables. In contrast, Figure 8 shows that the end-to-end Accuracy does not exhibit this parabola when expanding the prompt to include symbolic reasoning. The general trend for both prompts is that compile and runtime errors increase as the number of variables increases across all problem types. Figures 5 and 9 show that LLaMa-4 remained a poor performer across all categories and all problem types, under-performing even in comparison to its own predecessor model.

Token limitation for LLaMa-3.3 ruled out all but the simplest linear and quadratic programs whenever the full conversation history was provided as context. We also found that the GPT-OSS-120B model performed poorly on both datasets, but demonstrated significant improvements when given additional prompting to follow the symbolic interchange format. Removing either GPT-OSS-120B or the Gemini model eliminated the significance of the E2a results, which were otherwise insignificant with respect to the two LLaMa models.

We breakdown Gemini-1.5’s performance with respect to the number of variables in Figures 6 and 11. Again, general trends indicate that performance declines as problem complexity increases across all problem types. Linear programs remain the best performers. Curiously, Gemini-1.5’s performance on integer linear programs shot up from 0% to almost 50% when the number of variables increased from 7 to 8, despite having bottomed out at 6 variables. Crucially, we witnessed that runtime errors increased in proportion to the number of variables across all models.

Finally, we also broke down the performance of each model and prompt pair according to whether the problem was feasible by problem type in Figures 32 - 38. We generally observed that models performed more ‘accurately’ when the problem was unsolvable than when it was solvable across all problem types. We suspect this can be attributed to our measurement method, which simply examined that the output of the problem matched the corresponding Problem output; it is infinitely easier to make a problem infeasible. Consider an LLM Service incorrectly recording one constraint as another- both the correct constraint and the incorrect constraint happen to render a problem description infeasible. In such a case, the LLM Service might get the correct answer unintentionally, despite incorrectly formalizing the problem.

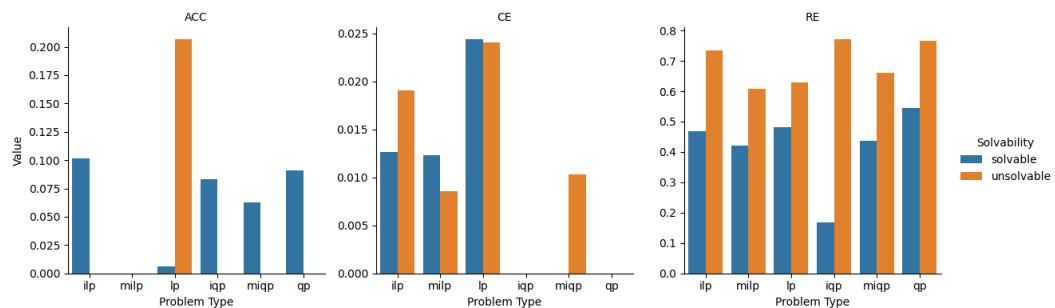


Figure 32: Distribution of scores for LLaMa-3.3 according to problem feasibility.

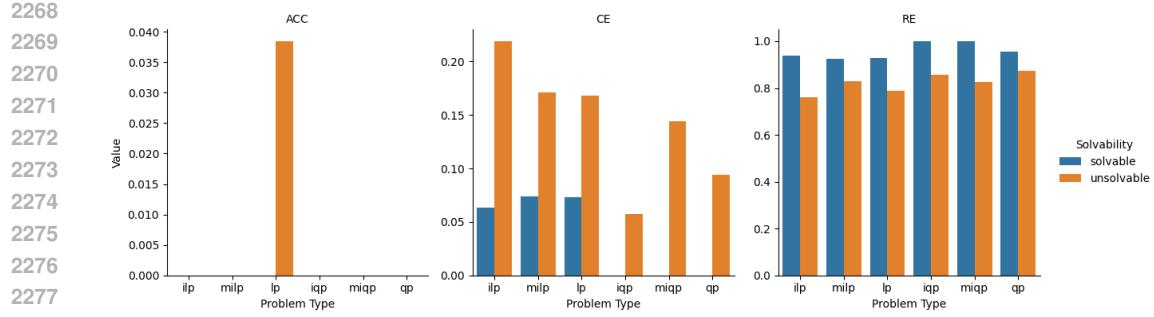


Figure 33: Distribution of scores for LLaMa-4 according to problem feasibility.

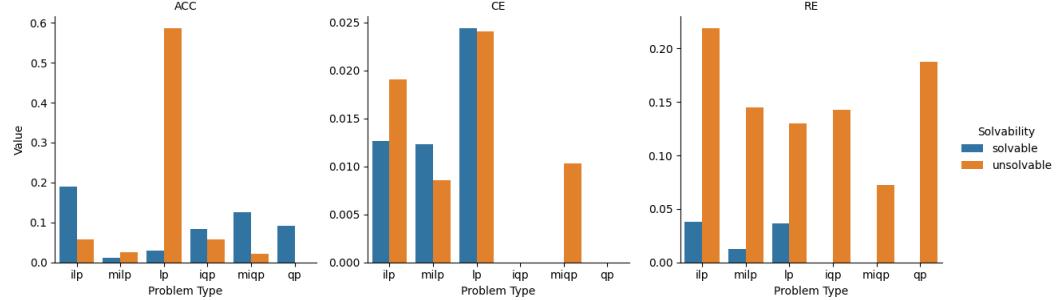


Figure 34: Distribution of scores for Gemini according to problem feasibility.

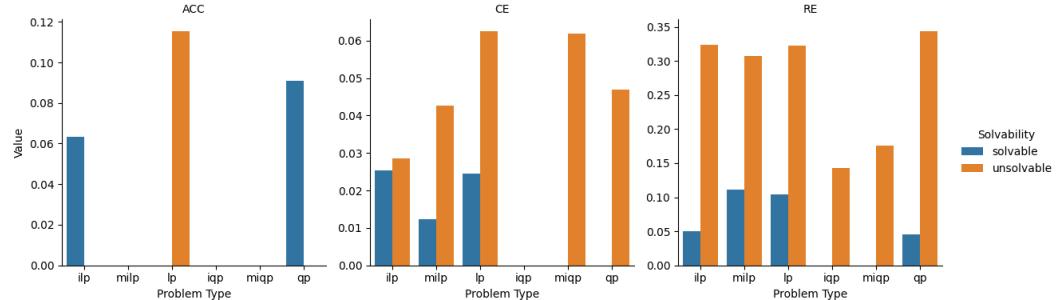


Figure 35: Distribution of scores for GPT-OSS according to problem feasibility.

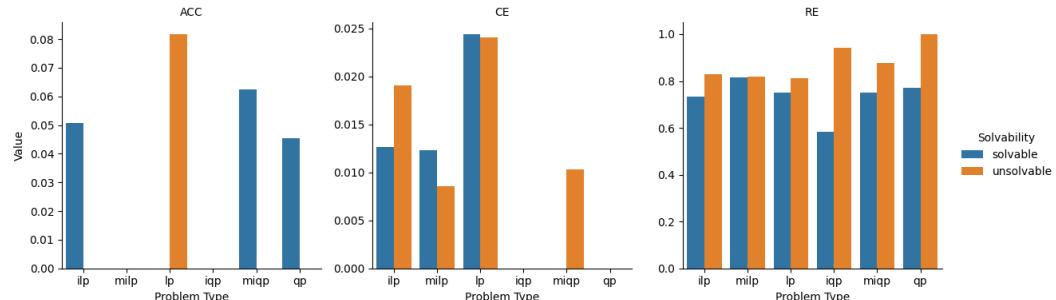


Figure 36: Distribution of scores for LLaMa-3.3 with additional prompting according to problem feasibility.

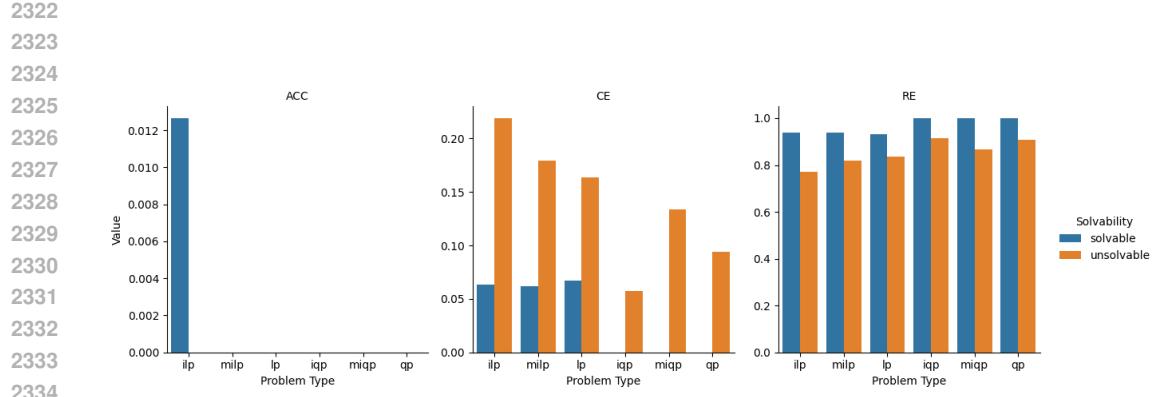


Figure 37: Distribution of scores for LLaMa-4 according to problem feasibility.

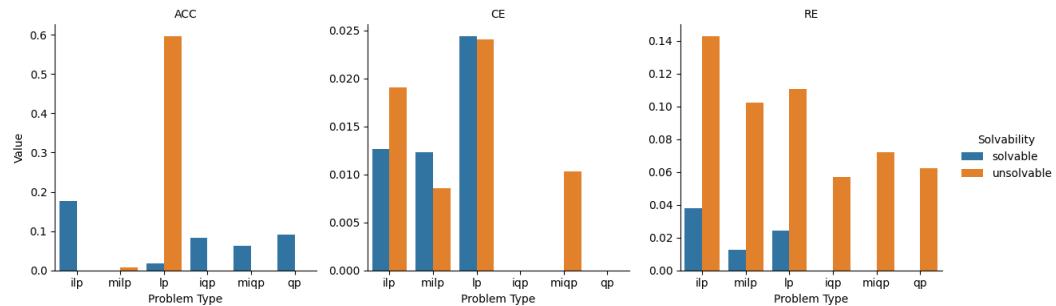


Figure 38: Distribution of scores for Gemini according to problem feasibility.

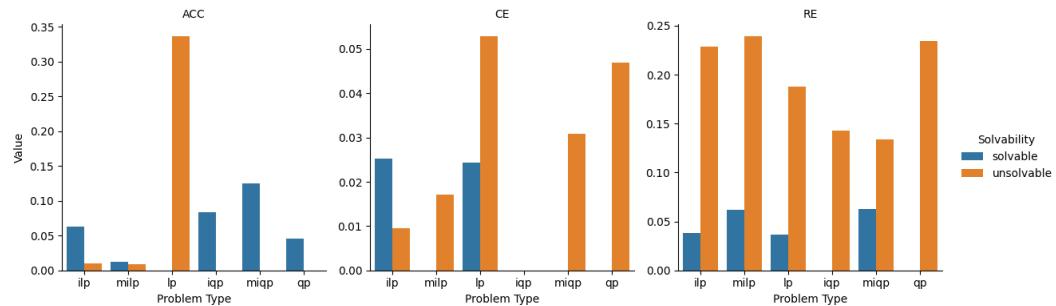


Figure 39: Distribution of scores for GPT-OSS according to problem feasibility.