DualSchool:
How Reliable are LLMs for Optimization Education?

Michael Klamkin[*†‡] **Arnaud Deza**[†‡]

Sikai Cheng[‡] **Haoruo Zhao**[‡] **Pascal Van Hentenryck**[‡]

Abstract

Consider the following task taught in introductory optimization courses which addresses challenges articulated by the community at the intersection of (generative) AI and OR: *generate the dual of a linear program.* LLMs, being trained at web-scale, have the conversion process and many instances of Primal to Dual Conversion (P2DC) at their disposal. Students may thus reasonably expect that LLMs would perform well on the P2DC task. To assess this expectation, this paper introduces DUALSCHOOL, a comprehensive framework for generating and verifying P2DC instances. The verification procedure of DUALSCHOOL uses the *Canonical Graph Edit Distance*, going well beyond existing evaluation methods for optimization models, which exhibit many false positives and negatives when applied to P2DC. Experiments performed by DUALSCHOOL reveal interesting findings. Although LLMs can recite the conversion procedure accurately, state-of-the-art open LLMs fail to consistently produce correct duals. This finding holds even for the smallest two-variable instances and for derivative tasks, such as correctness, verification, and error classification. The paper also discusses the implications for educators, students, and the development of large reasoning systems.

1 Introduction

Large Language Models (LLMs) have garnered significant interest for their potential to serve as always-available personalized education assistants, automating time-consuming tasks such as tutoring and grading in STEM education. To fully realize this potential, however, LLMs must demonstrate the ability to reliably execute detailed multi-step procedures. In particular, real-world tasks are often nuanced, making attention to detail paramount to success – a higher bar than plausible-looking text.

This paper proposes the relatively simple primal-to-dual conversion (P2DC) task as a benchmark to evaluate whether LLMs can execute detailed procedures reliably. P2DC is an interesting task for several reasons. (1) P2DC is commonly taught in introductory optimization courses. (2) LLMs, being trained on a web scale, have the conversion process and many instances of Primal to Dual Conversion (P2DC) in their training corpus. Indeed, when asked directly how to do P2DC, most LLMs respond with a correct strategy. (3) P2DC requires a clear understanding of the procedure, since there are many ways to obtain a dual. (4) P2DC captures several challenges recently articulated at the intersection of (generative) AI and OR, including the verification of optimization models, the availability of datasets, and the design of evaluation criteria and methodologies [1]. (5) P2DC is also an inherently structured task since the input and output are linear programs which can be represented in different formats (e.g., JSON, XML, MPS files). This makes the P2DC task an attractive test-bed

---

[*]Corresponding author: `klam@isye.gatech.edu`

[†]Equal contribution

[‡]NSF AI Institute for Advances in Optimization, Georgia Institute of Technology, Atlanta, GA, USA
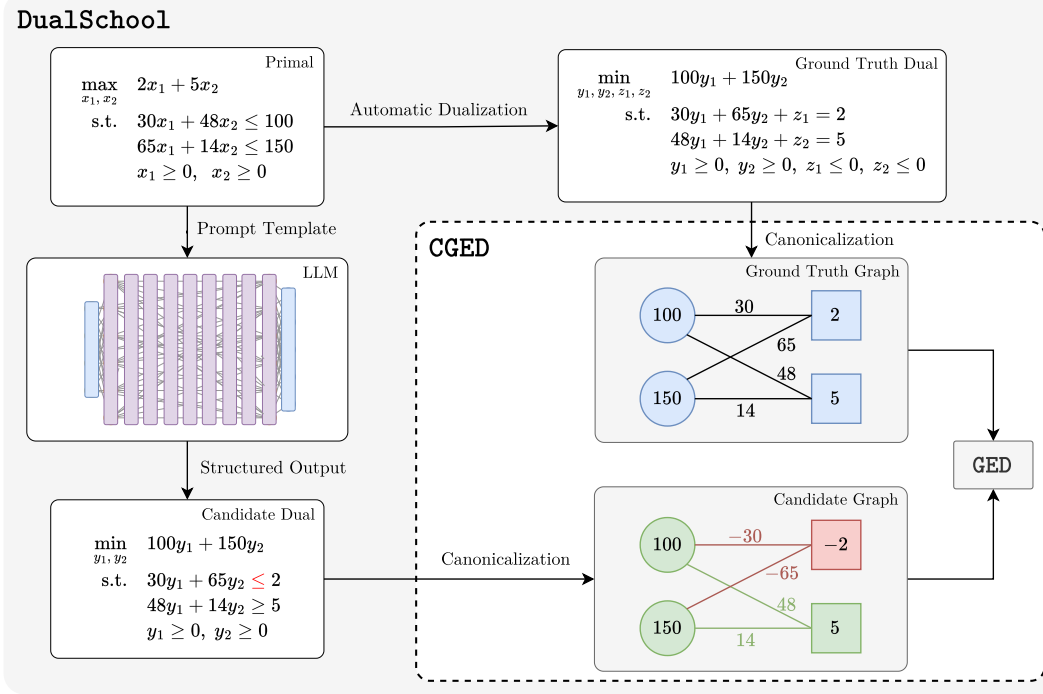
Figure 1: The P2DC task of DUALSCHOOL: it illustrates the primal-to-dual conversion, the canonical representation of linear programs and the evaluation using CGED, which is the concatenation of the canonicalization step and the Graph Edit Distance comparison.

for reasoning models specialized to structured data, a relatively under-studied but extremely valuable competency.

Because of the simplicity of P2DC and the availability of the P2DC instructions and instances in the training corpus of LLMs, students in optimization classes may reasonably expect that LLMs would perform well on the P2DC task. To assess this expectation, this paper introduces DUALSCHOOL, a comprehensive framework for generating and verifying P2DC instances. To generate P2DC instances, DUALSCHOOL leverages automatic symbolic dualization, converting new synthetic and existing primal-only datasets (e.g. NLP4OPT [2], ComplexOR [3], and EasyLP [4]), to P2DC datasets. To enable automatic evaluation, DualSchool includes a graph-based equivalence detection algorithm called *Canonical Graph Edit Distance (CGED)*. CGED is similar to the NGED algorithm of Xing et al. [5], but it adds a crucial pre-processing canonicalization step specifically designed to allow for differences in dualization procedure conventions. As such, DUALSCHOOL overcomes the limitations of existing validation techniques in the optimization setting which are either overly restrictive or overly permissive. Indeed, existing validations either force particular convention choices or forget much of the problem structure, complicating their use in post-training techniques to improve performance (e.g., [6, 7]). P2DC is illustrated in Figure 1, which exemplifies the canonical form used for comparing linear programs.

Preliminary experimental results with DUALSCHOOL reveal interesting findings: they show that the P2DC is surprisingly challenging for leading open LLMs. This discrepancy – being able to recite the procedure but not carry it out reliably – underscores a critical limitation of LLMs: they yield duals with mistakes that may be minor in terms of token count but are clearly wrong (e.g., an unbounded dual for a feasible primal). These findings hold even for the smallest two-variable instances and for derivative tasks, such as error correction, error classification, and verification. In CORRECTION, the LLM is asked to correct the error; in CLASSIFICATION, the LLM is asked what the error is; and in VERIFICATION, the LLM is asked if the primal-dual pair is valid.

The main contributions of this paper can be summarized as follows:

1. The paper proposes DUALSCHOOL, a comprehensive framework to evaluate the reliability of LLMs for a relatively simple optimization task, whose instructions are widely available. The multi-task framework includes the P2DC (primal to dual conversion) task and derivative tasks CORRECTION, CLASSIFICATION, and VERIFICATION.

2. The paper designs a *robust automatic evaluation* using a *Canonical Graph Edit Distance (CGED)* algorithm that simultaneously allows for differences in dualization convention while robustly checking the correctness of all problem data.

3. The paper is associated with a repository of *open data and code*: over 1,300 primal-dual pairs as well as error-injected variants for each are published alongside the paper, including the code to automatically generate more samples if needed.

4. Experimental results show that P2DC is a compelling challenge despite its simplicity, as state-of-the-art open LLMs struggle even for very small instances. P2DC, and its derivative tasks, also address several challenges recently articulated at the intersection of (generative) AI and OR [1].

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the P2DC task and Section 4 introduces the CGED algorithm. Then, Section 5 presents the experimental results and Section 6 concludes the paper.

2 Related Work

This section reviews related work at the intersection of large language models and optimization.

LLMs for Optimization   Recent years have witnessed a surge of interest in leveraging large language models (LLMs) for various optimization tasks (LLM4OPT). Notable examples include natural language modeling [8, 3, 2], where LLMs are given a natural language description of an optimization problem and are asked to formulate it, conversational interfaces to configure and customize existing models [9, 10], algorithmic configuration for cutting plane subroutines in MILP solvers [11], explaining infeasibilities [12], and even solving optimization problems directly [13]. These efforts highlight the growing recognition of LLMs as a versatile tool that can be applied beyond traditional natural language processing tasks and into the realm of mathematical optimization.

Evaluation Methods   Despite the increasing attention, existing work in this area primarily rely on the optimal objective value as the sole criterion for evaluating correctness of LLM-generated optimization formulations. This approach has inherent limitations, as it can silently ignore major errors in the formulation such as missing or incorrect constraints/variables if those errors happen to not affect the optimal value. The problem of equivalence detection between different optimization formulations remains largely under-explored. [5] shows that their normalized graph edit distance (NGED) more closely aligns with human assessments of correctness than token and optimal-value based evaluation. However, in the context of P2DC, the direct use of NGED results in many false negatives due to benign conventional differences, while optimal value yields many false positives.

LLM for Education and Tutoring   The potential of LLMs in education and tutoring has attracted significant interest with several recent works exploring their use in personalized learning and automated assessment. [14] investigates the use of LLMs in personalized tutoring systems. Benchmarks such as MathTutorBench [15] have been developed to evaluate the capabilities of LLMs in educational settings, and case studies such as [16] and [17] explore their potential in physics and power engineering education respectively.

3 Primal-to-Dual Conversion

This section introduces P2DC, the main task of DUALSCHOOL. A linear program (LP) is a constrained optimization problem with linear objective and affine constraints, i.e., a problem that can be stated as

$$\min_{x \in \mathbb{R}^n} \quad c^\top x \tag{1a}$$

$$\text{s.t.} \quad a_j^\top x \le b_j \qquad \forall j \in \mathcal{I}_\le \tag{1b}$$

$$a_j^\top x \ge b_j \qquad \forall j \in \mathcal{I}_\ge \tag{1c}$$

$$a_j^\top x = b_j \qquad \forall j \in \mathcal{I}_= \tag{1d}$$

where $x \in \mathbb{R}^n$ is the vector of decision variables, $c \in \mathbb{R}^n$ is the objective vector, $a_j \in \mathbb{R}^n$ are the constraint coefficient vectors, and $b_j \in \mathbb{R}$ are the constraint right-hand-sides. Any x which satisfies (1b), (1c) and (1d) (i.e., a *feasible x*) provides an upper bound on the optimal value of the LP. Any feasible solution to the *dual* of an LP, which itself is an LP[4], provides a lower bound on the optimal LP value. Moreover, in many practical applications, dual programs often have useful interpretations, as demonstrated in the example below. Readers are referred to Nemirovski [18, Section 1.3] for a detailed introduction to linear programming duality.

Example: Production Planning Consider the production planning problem where a factory manager is tasked with finding the most profitable production plan given a fixed amount of resources; wood (W) and steel (S). In this example, the factory can produce a number of doors (d) and tables (t), each using varying amounts of resources. All products are sold for a profit, p_d for doors and p_t for tables. The amount of wood (resp. steel) needed to produce a door is denoted by a_{wd} (resp. a_{sd}) and likewise the amount of wood (resp. steel) needed to produce a table is denoted by a_{wt} (resp. a_{st}). The full formulation is stated below as Model 2. The dual of this program, given below as Model 3, is known as the resource-valuation problem [19], with variables y_w and y_s denoting the so-called "shadow-price" of wood and steel respectively.

$$
\begin{array}{ll}
\max_{d,t} & p_d d + p_t t \\
\text{s.t.} & a_{wd}d + a_{wt}t \le W \\
& a_{sd}d + a_{st}t \le S \\
& d \ge 0, \ t \ge 0
\end{array}
\quad (2) \quad \Longrightarrow \quad
\begin{array}{ll}
\min_{y_w,y_s} & Wy_w + Sy_s \\
\text{s.t.} & a_{wd}y_w + a_{sd}y_s \ge p_d \\
& a_{wt}y_w + a_{st}y_s \ge p_t \\
& y_w \ge 0, \ y_s \ge 0
\end{array}
\quad (3)
$$

Performing this transformation – converting Model 2 to Model 3 – is the main task in DUALSCHOOL, called Primal-to-Dual Conversion (P2DC). Note that there exists several different procedures for deriving the dual of an LP; the most common methods are summarized in Appendix A.1. The dual program is not unique in that there exist different procedures and convention choices that can yield different, but valid, dual programs. For instance, consider Model 4, that differs from Model 3 only in its use of slack variables z_d and z_t to convert the inequality constraints to equalities. In the context of P2DC, one may arrive at Model 4 if dualizing by first converting the primal to the standard form $\min c^\top x$ s.t. $Ax \le b$ then writing the dual as $\max b^\top y$ s.t. $A^\top y = c, \ y \ge 0$.

$$
\begin{array}{ll}
\min_{y_w,y_s,z_d,z_t} & Wy_w + Sy_s \\
\text{s.t.} & a_{wd}y_w + a_{sd}y_s - z_d = p_d \\
& a_{wt}y_w + a_{st}y_s - z_t = p_t \\
& y_w \ge 0, \ y_s \ge 0, \ z_d \ge 0, \ z_t \ge 0
\end{array}
\tag{4}
$$

Models 3 and 4 are both considered "correct" duals of Model 2. This complicates equivalence detection, since directly applying NGED on the graphs corresponding to Models 3 and 4 would result in a non-zero edit-distance due to the missing slack variable nodes, missing edges denoting the slack constraint coefficients, and changed constraint senses. Thus, in the context of P2DC, it is important to use a "convention-invariant" matching procedure that explicitly treats such differences. Section 4 expands on the shortcomings of existing approaches and proposes a new metric, *Canonical Graph Edit Distance (CGED)*, that meets the requirements of the P2DC setting.

---

[4]The dual program corresponding to Model 1 is stated in Appendix A.1 as Model 7.

4 Automatic Evaluation for P2DC

This section begins by explaining why existing evaluation methods are insufficient to determine the correctness of a candidate dual in the P2DC setting. It then proposes a correctness detection algorithm that extends that of Xing et al. [5] in order to enable a "convention-invariant" matching of the candidate dual program to a known correct dual formulation obtained by automatic dualization.

NER-based matching   Several prior works [2, 20, 21], including the NL4OPT "generation" sub-task, use a declaration-level accuracy [2, Equation 2] that matches tokens within declarations. Although, compared to a naive token-matching, this metric allows to handle permutations of declarations, it cannot handle basic symmetries that are either *within* declarations, such as the order of terms in a constraint, or span across multiple declarations, such as variable sign convention.

Optimal Value   Most prior work uses an optimal value check, often referred to as execution accuracy, to establish the correctness of a given formulation [8, 22]. Although this is a necessary condition for correctness, it often yields false positives. For example, if the formulation omits a constraint that is not tight at the optimal solution, the optimal value check will still mark the formulation correct. Furthermore, in the P2DC setting, simply echoing back the primal model always gives the same objective value, even for problems which are not self-dual (due to strong duality). This makes the optimal value check easy to "reward-hack" [23], limiting its applicability as a reward signal.

Polyhedral congruence and isomorphism   Since the feasible set of an LP is a polyhedron, polyhedral computation libraries, e.g., `polymake` [24], can be used to evaluate whether the polyhedra corresponding to the candidate and ground truth feasible sets are congruent or isomorphic. However, checking congruency does not fit the P2DC setting since, although it does treat permutations and variable sign convention differences and, in some cases it may be useful to forget constraint scaling and redundant constraints, polyhedral congruence also allows for arbitrary transformations such as rotation and translation, breaking the primal-dual correspondence. Similarly, polyhedral isomorphism is not a good fit since it verifies only the incidence structure of the polyhedron, effectively forgetting most of the structure imposed by the problem data.

Graph Edit Distance   Recent work in ML for optimization use graph representations of optimization problems [25, 26]. Xing et al. [5] proposes to use graph edit distance (GED) algorithms on these graph representations to evaluate equivalence between a candidate and a ground truth program. Their method, called NGED, is attractive due to its ability to handle variable and constraint permutations. Furthermore, GED is a rich reward signal in that it outputs not only a boolean value but the optimal edit path between the two formulations. However, NGED is still overly restrictive in the P2DC setting since dualization procedures and conventions can result in dual programs that have different graph representations, i.e. additional variable nodes and edges when slack variables appear. Thus, directly applying NGED is too restrictive for P2DC.

Finally, note that each of these metrics rely on successful parsing of the LLM output into the required representation for comparison, e.g. XML for NER or the bipartite graph for GED. In cases where the parser fails due to incorrect formatting of the LLM response, the paper deems the formulations not equivalent. However, as shown in Section 5, the vast majority of responses are correctly parsed.

4.1 Canonical Graph Edit Distance

This paper proposes the *Canonical Graph Edit Distance (CGED)* for correctness detection: it modifies the NGED method to include a canonicalization step in order to control for variations in dualization procedure. In particular, the paper notes that these convention differences result in (combinations of) two kinds of "symmetries" in the dual programs: variable sign and slack variables. Although NGED itself includes some canonicalization such as converting the objective sense to minimization and single-sided inequalities to less-than sense, it fails to treat these convention differences, leading to many false negatives as demonstrated in Section 5. The following paragraphs describe the proposed modifications in detail.

Slack variables   As demonstrated using Models 3 and 4 in Section 3, slack variables appear in the dual when the dualization procedure treats primal variables as free, either by explicitly including their

bounds in the main constraints or due to the particular standard form or ruleset used. The correction detection algorithm treats these convention differences by eliminating the resulting slack variables. Equation (5) demonstrates such an elimination.

$$
\begin{array}{ll}
\min & x_1 + x_2 \\
\text{s.t.} & x_1 + x_2 + s = 1 \\
& s \leq 0
\end{array}
\quad \Longrightarrow \quad
\begin{array}{ll}
\min & x_1 + x_2 \\
\text{s.t.} & x_1 + x_2 \geq 1
\end{array}
\tag{5}
$$

Variable Sign The second common symmetry that arises due to variations in dualization convention is variable sign, since as pointed out in Step 2 of the Lagrangian dualization method in Appendix A.1, the sign constraint given to a dual variable only has to match how the corresponding constraint's residual is formed. In other words, as long as the memorized procedure/standard form/ruleset is consistent, the dual variables can be given any sign, including free variables (corresponding to equality constraints in the primal). For instance, the programs in Equation 6 are deemed equivalent for the purposes of P2DC.

$$
\begin{array}{ll}
\min & x_1 + x_2 \\
\text{s.t.} & x_1 + x_2 \geq 1 \\
& x_1 \geq 0, \; x_2 \geq 0
\end{array}
\quad \Longleftrightarrow \quad
\begin{array}{ll}
\min & -x_1' + x_2 \\
\text{s.t.} & -x_1 + x_2 \geq 1 \\
& x_1' \leq 0, \; x_2 \geq 0
\end{array}
\tag{6}
$$

To establish equivalence up to variable sign, both the candidate and ground truth are reformulated to convert variables whose bound constraint reads $x \leq u$ to $x \geq -u$ by flipping the sign of its constraint and objective coefficients. In order to allow for sign convention differences in free variables and those with double-sided finite bounds, CGED exploits the variable permutation property of GED by using the difference-of-positive variables transformation $x \in \mathbb{R} \implies x = x^+ - x^-$, $x^+ \geq 0$, $x^- \geq 0$. An example for the double-sided finite bounded variable case is given in Appendix A.2.1.

When these modifications are used as pre-processing steps for GED, the overall procedure is able to recognize the correctness of a dual even if it differs from the ground truth in the following ways:

1. **Objective sense** – flipping the objective sense is allowed as long as all objective coefficient signs are also flipped. In P2DC, this corresponds to a common post-processing that is applied if, for instance, all the objective coefficients are negative.

2. **Inequality sense** – flipping the sense of an inequality is allowed as long as the signs of the coefficients and RHS are also flipped. In P2DC this corresponds to a common post-processing that is applied if, for instance, all the problem data in a constraint is negative.

3. **Variable and constraint permutation** – reordering constraints and variables is allowed.

4. **Slack variables** – using slack variables to turn inequalities into equalities is allowed. In P2DC, slack variables appear in the dual when treating primal variables as free.

5. **Variable sign** – flipping the sign of a variable is allowed as long as the sign of its constraint and objective coefficients are also flipped. In P2DC, this corresponds to a common convention choice when defining dual variables associated with primal inequality constraints.

Note that although CGED is designed specifically for the P2DC setting, the canonicalization procedures can be used more broadly to detect equivalence between formulations or as a normalization procedure for systems that take a linear program as input. For other applications, it is important to consider exactly what should be preserved and what should be forgotten. In particular, due to its specialization to P2DC, CGED does not treat several symmetries which may be natural to forget in other areas such as scaling of variables or constraints and variable substitutions.

5 Experiments

This section describes the experiments conducted to evaluate the performance of leading open LLMs on the DUALSCHOOL dataset.

Benchmark Instances DUALSCHOOL comprises over 1300 LP instances drawn from three main sources: (1) two dimensional LPs from bounded toy poltyopes, (2) continuous relaxations of small-scale combinatorial optimization instances and (3) LP instances from prior work on natural language

Table 1: Mean (max) number of variables and constraints for each dataset.

	ComplexOR[3]	Easy LP [4]	NL4OPT[2]	NLP4LP[8]	2D	CO-Small
Variables	4.1 (9)	2.8 (5)	2.0 (3)	2.2 (6)	2.0 (2)	3.9 (5)
Constraints	4.5 (12)	4.3 (14)	2.9 (5)	3.1 (6)	5.7 (12)	3.5 (6)
Instances	15	585	205	266	108	140

modeling benchmarks[2–4, 8]. For each instance, DUALSCHOOL includes ground-truth duals generated using symbolic dualization [27]. For the CORRECTION, VERIFICATION, and CLASSIFICATION tasks, DUALSCHOOL also includes duals with (labeled) errors; the error types are described in Appendix C. Table 1 summarizes the data sources.

Language Models   Due to resource limitations, the experiments consider only small and medium-sized open-weight LLMs. The models are evaluated in both the zero-shot and one-shot in-context learning settings. Readers are referred to Appendix A.3 for more details about model configurations and compute resources.

Evaluation Pipeline   For GENERATION and CORRECTION, the LLM is prompted to produce `gurobipy` code that formulates the dual. The code is executed in a sandbox environment to create the `gurobipy.Model` object which is then written to MPS for evaluation. That MPS file is then compared to that of the ground truth dual using NGED [5], OBJ [8, 22], and the proposed CGED (Section 4). Instances that crash or cannot be parsed are counted as incorrect. For the VERIFICATION and CLASSIFICATION tasks, two methods for extracting an answer from the LLM response are tested: 1) XML from free-flow output and 2) enforced JSON schema.[5] Problems are rendered to the model in LaTeX using the prompt template described in Appendix D.[6]

The DUALSCHOOL tasks have two kinds of outputs: models (GENERATION and CORRECTION) and choices (VERIFICATION and CORRECTION). For tasks with model outputs, three metrics are reported: the canonical graph edit distance (CGED) from Section 4, the normalized graph edit distance (NGED) from Xing et al. [5], and the objective-match (OBJ). Each of these is reported as an accuracy, i.e. how often the edit distance to the ground truth dual is equal to zero. For tasks with choice outputs, the classification accuracy is reported, i.e. how often the LLM chose the correct choice.

5.1   Results for P2DC GENERATION

Table 2 reports the CGED, NGED, and OBJ accuracies across four benchmark datasets under both 0-shot and 1-shot prompting conditions. The Execution accuracy (Exec%) column reports the percentage of instances for which the LLM code successfully produced an MPS file. Overall, even though Exec% is high for most models, no model reliably produces correct duals – even when prompted with small, synthetic instances. The best-performing model, Phi 4 - 14B, reaches 47.8% CGED and 53.7% objective accuracy on the NL4OPT samples in the 0-shot setting. Due to space limitations and relatively poor performance, the results for the instances coming from CO small and Easy LP are presented in Appendix A.3 in Table 4.

In all cases, OBJ consistently exceeds CGED. This highlights a key pitfall of objective-based evaluation: LLMs often produce duals with the correct objective value but incorrect or malformed structure. Based on an informal analysis of samples in this category – CGED is nonzero while OBJ is true – the most common mistake is omitting a (dual) variable bound that happened to be redundant. Conversely, NGED is too restrictive in this setting, giving consistently lower scores than CGED. These results underscore the need for convention-invariant evaluation like CGED in the P2DC setting. Surprisingly, one-shot prompting provides no consistent benefit and occasionally degrades performance (e.g. Phi-4 drops from 35.7% to 28.6% on the COMPLEXOR samples), suggesting limited applicability of in-context learning in the P2DC setting.

---

[5] using the Structured Outputs feature in Ollama `https://ollama.com/blog/structured-outputs`

[6] LaTeX is generated using `https://jump.dev/JuMP.jl/stable/manual/models/#Print-the-model`

Table 2: Aggregated accuracy results for the GENERATION task

Model	Prompt	Exec%	ComplexOR			NL4OPT			NLP4LP			2D		
			NGED	OBJ	CGED	NGED	OBJ	CGED	NGED	OBJ	CGED	NGED	OBJ	CGED
Mistral-7B	0-shot	24.8	0	0	0	0	0	0	0	3.9	0	0	11.5	0
	1-shot	22.5	0	50.0	0	0	0	0	0	3.3	0	0	7.7	0
Phi 4-14B	0-shot	99.1	7.1	50.0	**35.7**	24.4	53.7	**47.8**	10.5	46.8	**30.8**	1.0	5.7	**1.0**
	1-shot	99.7	7.1	42.9	**28.6**	2.0	34.5	**27.6**	2.9	34.0	**18.9**	14.0	18.7	**14.0**
Gemma 3-12B	0-shot	69.9	0	0	0	0.6	3.7	3.0	0	4.6	0	0	8.5	0
	1-shot	92.5	0	0	0	0	0	0	0	3.0	0	0	5.8	0
Qwen 2.5–7B	0-shot	93.7	0	0	0	0	3.1	0.5	1.3	3.5	1.8	0	1.9	0
	1-shot	94.2	0	0	0	0	0	0	0	1.8	0	0	2.9	0
Qwen 2.5–14B	0-shot	92.2	7.1	14.3	7.1	4.3	20.7	13.6	2.7	10.9	3.6	0	2.1	0
	1-shot	93.6	0	0	0	0	0	0	0	3.1	0.4	10.4	23.6	10.4
Llama 3.1-8B	0-shot	94.8	0	0	0	0	1.7	1.1	0	1.4	0	0	6.6	0
	1-shot	95.3	0	0	0	0	0	0	0	4.6	1.4	0	9.7	0
Llama 3.3-70B	0-shot	73.8	16.7	41.7	25.0	17.0	45.8	39.9	8.2	30.6	18.9	0	11.1	0
	1-shot	100.0	0	21.4	14.3	0	19.0	17.6	0.4	24.8	16.0	1.9	6.5	1.9

5.2 Results for P2DC CORRECTION

Figure 2 reports the performance of LLMs on the CORRECTION task. All models struggle to reliably repair the incorrect duals, with accuracies below 60% across all models and error types. Similarly to the GENERATION task, the Phi 4 and Llama 3.3 models outperform the others. These uniformly low accuracies – even on error types that are relatively easy to detect as shown in the next section – reveal that CORRECTION is essentially just as challenging as GENERATION for the open LLMs considered.

Figure 2: Accuracy for the CORRECTION task by model and error type.

5.3 Results for P2DC VERIFICATION and CLASSIFICATION

Figure 3 reports the accuracies for the primal–dual pair VERIFICATION task (left) and the CLASSIFICATION task (right), with a red dashed line denoting the random-guess baseline (50% for VERIFICATION, 25% for CLASSIFICATION). Overall, results are slightly negative: most models cluster at or below the random-guess baseline, reflecting limited reliability of predictions. Importantly, note that in the VERIFICATION task, across all models there is a clear bias towards predicting "no", resulting in high accuracy on all but the "Correct" category (black); the only one where the right answer is "yes". This is even more prevalent when using structured outputs, as shown in Appendix B in Figure 4.

In the error classification task, detecting the flipped objective sense stands out as the easiest error type, with the best-performing models achieving accuracies between 70% and 90%. However, the more nuanced error types remain challenging with most models' accuracies hovering near or below the random-guess line.

Figure 3: Accuracy for the VERIFICATION and CLASSIFICATION tasks by model and error type.

6 Conclusion

This paper introduced DUALSCHOOL, the first comprehensive benchmark for probing an LLM's ability to perform and critique primal-to-dual conversions in linear programming. DUALSCHOOL com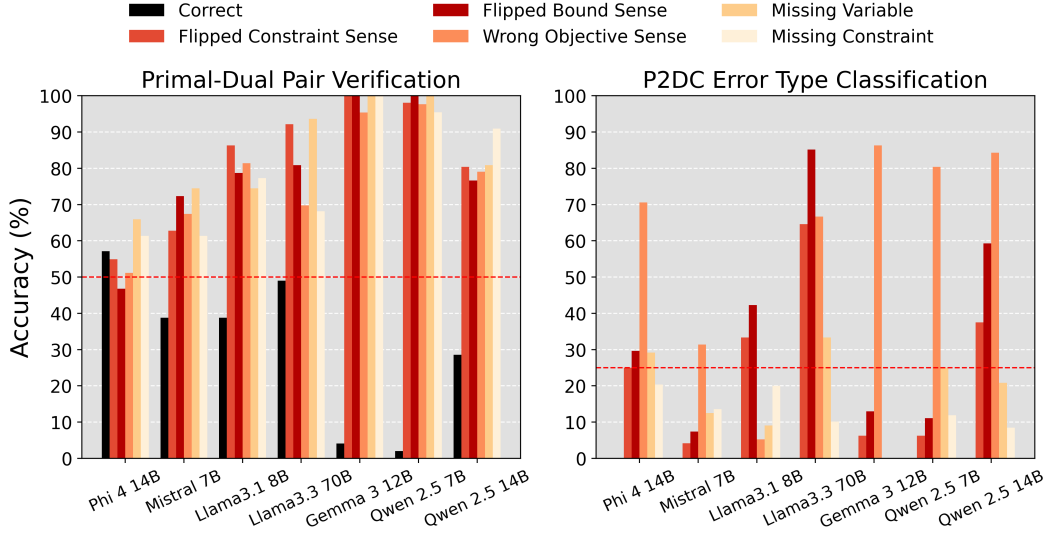bines four structured tasks (generation, verification, correction, and error classification) with a graph-based correctness detector that goes beyond simple token matching or objective-value checks, and is specifically designed to avoid false positives and negatives for the primal-to-dual task. Code is published alongside the paper as well as a dataset of P2DC samples based on both synthetic and real-world LPs, each including a set of duals with injected errors. Preliminary experiments using leading open models show that the best LLMs tested achieve a best-case dualization accuracy of only 47.8%, with similarly low performance on the derivative tasks.

From an education standpoint, it is important to communicate to students, especially those without knowledge of generative AI and its implementation, that LLMs are not in the same equivalence class as e.g. Matlab or Julia. Although LLMs can return the "recipe" for various mathematical tasks, they struggle to follow these recipes even for simple tasks that are expected from undergraduate students in introductory classes. Moreover, it is important to be aware of the fact that LLM responses often feature high-quality writing and rich formatting, making it easy to believe the response is accurate.

From a research standpoint, DUALSCHOOL provides a simple, yet meaningful benchmark to measure progress in LLMs and reasoning systems over the next years. Furthermore, thanks to the automatic labeling and evaluation algorithms, DUALSCHOOL can be directly used in fine-tuning methods such as reinforcement learning with symbolic feedback [7] that can leverage rich reward signals. Future directions include extending DUALSCHOOL to quadratic and conic formulations and evaluating its efficacy as a fine-tuning dataset.

Acknowledgements

References

[1] Segev Wasserkrug, Leonard Boussioux, Dick den Hertog, Farzaneh Mirzazadeh, Birbil Ş. Ilker, Jannis Kurtz, and Donato Maragno. Enhancing decision making through the integration of large

language models and operations research optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(27):28643–28650, Apr. 2025. doi: 10.1609/aaai.v39i27.35090. URL `https://ojs.aaai.org/index.php/AAAI/article/view/35090`.

[2] Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 189–203. PMLR, 28 Nov–09 Dec 2022. URL `https://proceedings.mlr.press/v220/ramamonjison23a.html`.

[3] Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*, 2023.

[4] Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. Mamo: a mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144*, 2024.

[5] Linzi Xing, Xinglu Wang, Yuxi Feng, Zhenan Fan, Jing Xiong, Zhijiang Guo, Xiaojin Fu, Rindra Ramamonjison, Mahdi Mostajabdaveh, Xiongwei Han, et al. Towards human-aligned evaluation for linear programming word problems. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 16550–16556, 2024.

[6] Subbarao Kambhampati, Kaya Stechly, and Karthik Valmeekam. (how) do reasoning models reason? *Annals of the New York Academy of Sciences*, 2025.

[7] Piyush Jha, Prithwish Jana, Pranavkrishna Suresh, Arnav Arora, and Vijay Ganesh. Rlsf: Reinforcement learning via symbolic feedback. *arXiv preprint arXiv:2405.16661*, 2024.

[8] Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, and Madeleine Udell. Optimus-0.3: Using large language models to model and solve optimization problems at scale. *arXiv preprint arXiv:2407.19633*, 2024.

[9] Dimos Tsouros, Hélène Verhaeghe, Serdar Kadıoğlu, and Tias Guns. Holy grail 2.0: From natural language to constraint models. *arXiv preprint arXiv:2308.01589*, 2023.

[10] Connor Lawless, Jakob Schoeffer, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahareh Sarrafzadeh. "i want it that way": Enabling interactive decision support using large language models and constraint programming. *ACM Transactions on Interactive Intelligent Systems*, 14(3):1–33, 2024.

[11] Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. Llms for cold-start cutting plane separator configuration. *arXiv preprint arXiv:2412.12038*, 2024.

[12] Hao Chen, Gonzalo E Constante-Flores, and Can Li. Diagnosing infeasible optimization problems using large language models. *INFOR: Information Systems and Operational Research*, 62(4):573–587, 2024.

[13] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

[14] Pepper Miller and Kristen DiCerbo. Llm based math tutoring: Challenges and dataset, 2024.

[15] Jakub Macina, Nico Daheim, Ido Hakimi, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. Mathtutorbench: A benchmark for measuring open-ended pedagogical capabilities of llm tutors. *arXiv preprint arXiv:2502.18940*, 2025.

[16] Ryan Mok, Faraaz Akhtar, Louis Clare, Christine Li, Jun Ida, Lewis Ross, and Mario Campanelli. Using large language models for grading in education: an applied test for physics. *Physics Education*, 60(3):035006, 2025.

[17] Alan Hickey, Cathal Ó Faoláin, and Paul Cuffe. Large language models in power engineering education: A case study on solving optimal dispatch coursework problems. In *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–5. IEEE, 2024.

[18] Arkadi Nemirovski. *Introduction to linear optimization*. World Scientific, 2024.

[19] Bryan Carsberg. On the linear programming approach to asset valuation. *Journal of Accounting Research*, pages 165–182, 1969.

[20] Rindranirina Ramamonjison, Haley Li, Timothy T Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions. *arXiv preprint arXiv:2209.15565*, 2022.

[21] Ganesh Prasath and Shirish Karande. Synthesis of mathematical programs from natural language specifications. *arXiv preprint arXiv:2304.03287*, 2023.

[22] Chenyu Huang, Zhengyang Tang, Dongdong Ge, Shixi Hu, Ruoqing Jiang, Benyou Wang, Zizhuo Wang, and Xin Zheng. Orlm: A customizable framework in training large models for automated optimization modeling. *arXiv e-prints*, pages arXiv–2405, 2024.

[23] Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471, 2022.

[24] Benjamin Assarf, Ewgenij Gawrilow, Katrin Herr, Michael Joswig, Benjamin Lorenz, Andreas Paffenholz, and Thomas Rehn. Computing convex hulls and counting integer points with polymake. *Mathematical Programming Computation*, 9:1–38, 2017.

[25] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

[26] Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33:18087–18097, 2020.

[27] Guilherme Bodin, Joaquim, Benoît Legat, Oscar Dowson, Pietro Monticone, Mathieu Besançon, LukasBarner, Miles Lubin, and martincornejo. jump-dev/dualization.jl: v0.6.0, May 2025. URL `https://doi.org/10.5281/zenodo.15338697`.

[28] PACE. *Partnership for an Advanced Computing Environment (PACE)*, 2017. URL `http://www.pace.gatech.edu`.

[29] Arthur T Benjamin. Sensible rules for remembering duals—the sob method. *SIAM review*, 37 (1):85–87, 1995.

[30] J. Löfberg. Dualize it: software for automatic primal and dual conversions of conic programs. *Optimization Methods and Software*, 24:313 – 325, 2009. ISSN 1055-6788. doi: 10.1080/10556780802553325.

[31] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[32] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 15:581–589, 2023. doi: 10.1007/s12532-023-00239-3.

[33] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671. URL `https://epubs.siam.org/doi/10.1137/141000671`.

[34] Yansen Zhang, Qingcan Kang, Wing Yin Yu, Hailei Gong, Xiaojin Fu, Xiongwei Han, Tao Zhong, and Chen Ma. Decision information meets large language models: The future of explainable operations research. *arXiv preprint arXiv:2502.09994*, 2025.

[35] Jonas Charfreitag and Mohammed Ghannam. GeCO, 2023. URL `https://github.com/CharJon/GeCO`.

A Appendix

A.1 Primal to dual conversion methods

Standard Form A common method for forming the dual of a primal program is to first memorize a standard-form primal-dual pair, i.e. $\min\ c^\top x$ s.t. $Ax \leq b \implies \max\ b^\top y$ s.t. $A^\top y = c,\ y \geq 0$, then convert the given primal to that standard form and apply the memorized map. If the standard form primal memorized is Model 1, this method is coincides with SOB (modulo objective sense). The dual of Model 1 is included below as Model 7.

$$\max_{x \in \mathbb{R}^n}\quad \sum_{j \in \mathcal{I}_\leq} b_j y_j + \sum_{j \in \mathcal{I}_{lgeq}} b_j y_j + \sum_{j \in \mathcal{I}_=} b_j y_j \tag{7a}$$

$$\text{s.t.}\quad A^\top y = c \tag{7b}$$

$$y_j \geq 0 \qquad\qquad\qquad\qquad \forall j \in \mathcal{I}_\leq \tag{7c}$$

$$y_j \leq 0 \qquad\qquad\qquad\qquad \forall j \in \mathcal{I}_\geq \tag{7d}$$

$$y_j \in \mathbb{R} \qquad\qquad\qquad\qquad \forall j \in \mathcal{I}_= \tag{7e}$$

Sensible-Odd-Bizarre Benjamin [29] describes the Sensible-Odd-Bizarre (SOB) method for re-membering how to write the dual of a linear program. Variants of the method, i.e. table or rule-based approaches, are widely used as a practical approach to write the dual without having to go through a standard form. The method is summarized in Table A.1

Table 3: The Sensible-Odd-Bizarre method for mnemonic dualization [29].

	Primal (Dual)	**Dual (Primal)**
Objective	Maximize $c^\top x$ (Minimize $b^\top y$)	Maximize $b^\top y$ (Minimize $c^\top x$)
	Constraint j:	Variable y_j (or x_i):
Sensible	$a_j^\top x_i \geq b_j$	$y_j \geq 0$
Odd	$a_j^\top x_i = b_j$	$y_j \in \mathbb{R}$
Bizarre	$a_j^\top x_i \leq b_j$	$y_j \leq 0$
	Variable x_i (or y_j):	Constraint j:
Sensible	$x_i \geq 0$	$a_i y_j \leq b_i$
Odd	$x_i \in \mathbb{R}$	$a_i y_j = b_i$
Bizarre	$x_i \leq 0$	$a_i y_j \geq b_i$

Lagrangian Duality The Lagrangian route to deriving the dual program starts by forming the Lagrangian function by introducing Lagrangian multipliers. The dual program is then the problem of maximizing the infimum of the Lagrangian over the primal variables subject to the Lagrangian multiplier sign constraints (for minimization primals).

1. Take as input the primal program Model 1.
2. Form the Lagrangian by introducing multipliers y_j. The example below will use the sign convention $y_j \geq 0\ \forall j \in \mathcal{I}_\leq,\ \ y_j \leq 0\ \forall j \in \mathcal{I}_\geq,\ \ y_j \in \mathbb{R}\ \forall j \in \mathcal{I}_=$ which corresponds to residual convention $b_j - a_j^\top x$. [7]

$$L(x, y) = c^\top x + \sum_{j \in \mathcal{I}_\leq} y_j^\top (b_j - a_j^\top x) + \sum_{j \in \mathcal{I}_\geq} y_j^\top (b_j - a_j^\top x) + \sum_{j \in \mathcal{I}_=} y_j^\top (b_j - a_j^\top x)$$

$$= b^\top y + x^\top (c - A^\top y)$$

3. Form the dual function by taking the infimum of the Lagrangian over x:

$$d(y) = \inf_{x \in \mathbb{R}^n} L(x, y) = \begin{cases} b^\top y & \text{if } c - A^\top y = 0 \\ -\infty & \text{otherwise} \end{cases}$$

---

[7]Note that the opposite sign convention can be used if using $a_j^\top x - b_j$ for that residual.

4. Maximize the dual function subject to the Lagrangian multiplier constraints:

$$
\begin{aligned}
\max_{y} \quad & d(y) \\
\text{s.t.} \quad & y_j \geq 0 \quad \forall j \in \mathcal{I}_{\leq} \\
& y_j \leq 0 \quad \forall j \in \mathcal{I}_{\geq} \\
& y_j \in \mathbb{R} \quad \forall j \in \mathcal{I}_{=}
\end{aligned}
\quad \implies \quad
\begin{aligned}
\max_{y} \quad & b^\top y \\
\text{s.t.} \quad & A^\top y = c \\
& y_j \geq 0 \quad \forall j \in \mathcal{I}_{\leq} \\
& y_j \leq 0 \quad \forall j \in \mathcal{I}_{\geq} \\
& y_j \in \mathbb{R} \quad \forall j \in \mathcal{I}_{=}
\end{aligned}
$$

Automatic Dualization   Several software systems allow for the automatic dualization of convex programs including YALMIP in MATLAB [30, 31] and JuMP in Julia [32, 33]. DualSchool uses `Dualization.jl` [27] from the JuMP ecosystem which implements a standard form-based approach.

A.2   CGED Implementation Details

Besides the canonicalization steps described in Section 4, there are several differences in how CGED is implemented compared to the EOR [34] implementation of NGED:

1. Variable nodes have only one feature c_i compared to the c_i, l_i, and u_i in NGED. This is due to the fact that variable bounds are included in the constraint nodes.

2. Constraint nodes have only one feature b_j compared to the l_i, u_i in NGED since in CGED, constraints are reformulated to $a_j^\top x \geq b_j$ rather than $l_j \leq a_j^\top x \leq u_j$. This allows to consider equivalent $l_j \leq a_j^\top x \leq u_j \iff -u_j \leq -a_j^\top x \leq -l_j$ and $a_j^\top x = b_j \iff -a_j^\top x = -b_j$. Note that the EOR [34] implementation differs from NGED as described in Xing et al. [5] in that the EOR version does not normalize constraints with less-than sense to greater-than.

A.2.1   Variable sign canonicalization example

The treatment of free and double-sided bounded variables in CGED relies on combining the difference-of-positives trick with the variable permutation invariance of GED. In the case of double-sided bounds, it also relies on the inequality sense normalization. Consider the example ground-truth and candidate pair in Equation 8.

$$
\begin{aligned}
\min \quad & x_1 + x_2 \\
\text{s.t.} \quad & x_1 + x_2 \geq 1 \\
& 1 \leq x_1 \leq 2, \ x_2 \geq 0
\end{aligned}
\quad \iff \quad
\begin{aligned}
\min \quad & -x_1 + x_2 \\
\text{s.t.} \quad & -x_1 + x_2 \geq 1 \\
& -2 \leq x_1 \leq -1, \ x_2 \geq 0
\end{aligned}
\tag{8}
$$

The algorithm begins by moving the double-sided bounds to the constraints, normalizing to $\geq$ sense.

$$
\begin{aligned}
\min \quad & x_1 + x_2 \\
\text{s.t.} \quad & x_1 + x_2 \geq 1 \\
& x_1 \geq 1 \\
& -x_1 \geq -2 \\
& x_1 \in \mathbb{R}, \ x_2 \geq 0
\end{aligned}
\quad \iff \quad
\begin{aligned}
\min \quad & -x_1 + x_2 \\
\text{s.t.} \quad & -x_1 + x_2 \geq 1 \\
& x_1 \geq -2 \\
& -x_1 \geq 1 \\
& x_1 \in \mathbb{R}, \ x_2 \geq 0
\end{aligned}
\tag{9}
$$

Then the difference of positives transformation is applied to free variables.

$$
\begin{aligned}
\min \quad & x_1^+ - x_1^- + x_2 \\
\text{s.t.} \quad & x_1^+ - x_1^- + x_2 \geq 1 \\
& -x_1^+ + x_1^- \geq -2 \\
& x_1^+ - x_1^- \geq 1 \\
& x_1^+ \geq 0, \ x_1^- \geq 0, \ x_2 \geq 0
\end{aligned}
\quad \iff \quad
\begin{aligned}
\min \quad & -x_1^+ + x_1^- + x_2 \\
\text{s.t.} \quad & -x_1^+ + x_1^- + x_2 \geq 1 \\
& x_1^+ - x_1^- \geq -2 \\
& -x_1^+ + x_1^- \geq 1 \\
& x_1^+ \geq 0, \ x_1^- \geq 0, \ x_2 \geq 0
\end{aligned}
\tag{10}
$$

Observe that the formulations are identical when swapping x_1^+ for x_1^- and vice-versa, which corresponds to a node order permutation that GED handles naturally.

Indeed, the difference-of-positives transformation can in principle be applied to all variables. Then, all bounds can be treated as constraints, removing the need for the explicit $x \leq u \implies x \geq -u$

canonicalization step. However, applying the transformation to all variables would introduce many extra nodes and edges which can needlessly slow down the graph edit distance computation and extend the optimal edit path length. Thus, CGED uses explicit canonicalization for one-side finite bounded variables (the most common case) to only introduce additional variables when handling free and double-bounded variables.

A.3 Additional details on experimental setup

Dataset Generation   The DUALSCHOOL samples come from three sources:

1. **2D LPs:** 36 canonical polytopes, each with three distinct objective vectors, ranging from simple shapes (e.g., unit square, triangle) to more complex ones (e.g., hexagon, irregular pentagon).

2. **CO Relaxations:** Seven families of combinatorial optimization instances are generated using GECO [35]: maximum independent set, multidimensional knapsack, maximum cut, maximum clique, minimum vertex cover, packing, and production planning.

3. **LLM4OPT-Derived LPs:**
   - NLP4LP ([8]): use the provided `gurobipy` code directly.
   - NL4OPT[2], Easy LP[4], ComplexOR[3]: these benchmarks only supply an objective value and prompt. Thus, Llama 3.3 is used, following [8], to generate `gurobipy` formulations for each sample. These formulations are checked using the objective value and re-tried until there is a match. Any instance for which there is no match within five retries is excluded from DUALSCHOOL.

LLM Configuration   All models run with temperature 0.0, context window size 8192, repeat penalty 1.1, top k 40, top p 0.9 and min p 0.05. Inference is performed via Ollama[8].

Compute Resources   The paper used in total about 1000 GPU-hours to run LLM inference for both the experiments presented and those run during development. The evaluations are run on CPU and are relatively fast, adding up to only about 1 CPU-hour. All experiments were run on a node with two Intel Xeon 6426Y (2.5GHz) CPUs and 8 NVIDIA L40S 48GB GPUs.

B Additional Experimental Results

B.1 LLMs know the procedure

To verify that the evaluated models "know" how to dualize an LP, the authors prompt and manually evaluate each model's response to "How do you convert a primal linear program to its dual?" Since this prompt does not request a highly detailed response nor an example, LLMs are free to remain at a high level where it is easier to describe a valid procedure; details such as converting correctly to its chosen standard form are a single sentence rather than a careful subroutine implementation. In this setting, all models evaluated produced valid procedures except for Gemma 3 - 12B which has an incorrect sign in the objective coefficients and Mistral 7B which forgot to treat primal equality constraints. Notably, all models used a standard form-type method, though with varying standard forms.

B.2 Symbolic and Synthetic Verification of LLM P2DC

Although the main task of DUALSCHOOL is dualization of specific linear programs, i.e. single instances, one may consider instead prompting the LLMs to generate code that carries out the dualization more generically, i.e. to write Python code implementing symbolic dualization. This section contains experimental results for such a setting, where the LLM is asked to create a function that takes the primal problem data tuple (A, b, c, l, u, objective sense, constraint senses) and returns its dual as a `gurobipy` model. Table 5 reports the aggregated accuracy results for this task, for both open and closed models. Similarly to the main GENERATION task, the execution accuracy is high

---

[8]`https://github.com/ollama/ollama`

Table 4: Aggregated accuracy results for the GENERATION task (continued)

Model	Prompt	Exec%	CO small			Easy LP		
			NGED	OBJ	CGED	NGED	OBJ	CGED
Mistral - 7B	0-shot	23.0	0	0	0	0	0	0
	1-shot	21.3	0	0	0	0	0	0
Phi 4 - 14B	0-shot	96.2	0	0	0	8.1	24.8	**7.9**
	1-shot	94.0	0	0	0	5.5	12.9	**6.0**
Gemma 3 - 12B	0-shot	64.8	0	0	0	0	0.8	0
	1-shot	91.4	0	0	0	0	0	0
Qwen 2.5 – 7B	0-shot	80.9	0	0	0	0	1.5	0
	1-shot	89.3	0	0	0	0	0.5	0
Qwen 2.5 – 14B	0-shot	84.1	0	0	0	0	5.6	0
	1-shot	96.2	0	0	0	0	0.4	0
Llama 3.1 - 8B	0-shot	92.1	0	0	0	0	1.0	0
	1-shot	89.0	0	0	0	0	0.6	0
Llama 3.3 - 70B	0-shot	71.0	0	0	0	3.5	10.4	3.5
	1-shot	99.8	0	0	0	0	1.0	0

Figure 4: Accuracy for VERIFICATION and CLASSIFICATION DUALSCHOOL tasks when using structured outputs.

– in fact 100% – indicating the LLMs reliably produce executable code. However, the dualization accuracy of the produced routines is very low, with only chatGPT 4o achieving a non-zero CGED.

B.3 Common mistakes

An informal analysis reveals the following common mistakes made by LLMs in the GENERATION task (when the response is almost correct):

- `gurobipy` defaults – When declaring a new variable in `gurobipy`, the default lower bound is zero. Sometimes, when attempting to model $x \leq 0$, the LLM forgets to set the lower bound to $-\infty$, effectively adding $x = 0$ instead.

16

Table 5: Aggregated accuracy results for the symbolic P2DC task.

Model	ComplexOR			NL4OPT			NLP4LP			Easy LP		
	NGED	OBJ	CGED	NGED	OBJ	CGED	NGED	OBJ	CGED	NGED	OBJ	CGED
Claude 3.5	0	0	0	0	0	0	0	0	0	0	0	0
Claude 3.7	0	14.3	0	0	0.5	0	0	3.3	0	0	0	0
Gemini 1.5 Flash	0	14.3	0	0	0	0	0	0.4	0	0	0	0
Gemini 2.0 Flash	0	14.3	0	0	33.2	0	0	27.4	0	0	0	0
Gemini 2.5 Flash	0	0	0	0	0	0	0	0	0	0	0	0
Gemma3 12B	0	0	0	0	0	0	0	0	0	0	0	0
Gemma3 27B	0	0	0	0	0	0	0	4.1	0	0	0	0
Gemma3 4B	0	0	0	0	0	0	0	0	0	0	0	0
chatGPT 4o	7.1	7.1	**7.1**	0	0.5	0	0	2.9	0	0	0	0
chatGPT 4o mini	0	0	0	0	0	0	0	0	0	0	0	0
chatGPT 4o mini-high	0	0	0	0	0	0	0	0	0	0	0	0
chatGPT O3	0	0	0	0	0	0	0	0	0	0	0	0
Llama3.1 8B	0	7.1	0	0	0	0	0	7.9	0	0	0	0
Llama3.2 3B	0	0	0	0	0	0	0	0	0	0	0	0
Phi 4	0	0	0	0	0	0	0	0	0	0	0	0
Qwen2.5 14B	0	0	0	0	0	0	0	0	0	0	0	0
Qwen2.5 7B	0	0	0	0	0	0	0	0	0	0	0	0

- Incorrect bound senses – often, the variable bound senses are consistently wrong, i.e. all bounds are flipped compared to ground truth, but the constraint and objective coefficients are not flipped accordingly.
- Missing dual variables – LLMs tend to forget to include dual variables corresponding to primal variable upper bounds when there is a finite lower bound on the same primal variable.

C Error Types

To systematically inject errors into ground-truth dual programs, the paper considers the following error types.

Wrong Objective Sense Flip the sense of the dual objective (minimize $\leftrightarrow$ maximize), without adjusting the coefficients.

$$
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \geq 1 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{array}
\implies
\begin{array}{ll}
\max & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \geq 1 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{array}
$$

Missing Variable Randomly select a variable and delete it from the model.

$$
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \geq 1 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{array}
\implies
\begin{array}{ll}
\min & 5x_1 \\
\text{s.t.} & 2x_1 \geq 1 \\
& x_1 \geq 0
\end{array}
$$

Missing Constraint Randomly select a constraint and delete it from the model.

$$
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \geq 1 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{array}
\implies
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & x_1 \geq 0,\ x_2 \geq 0
\end{array}
$$

Flipped Constraint Sense Randomly select a constraint and change its sense.

$$
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \geq 1 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{array}
\implies
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \leq 1 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{array}
$$

Flipped Bound Sense Randomly select a variable and change the sense of its bound constraint.

$$
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \geq 1 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{array}
\implies
\begin{array}{ll}
\min & 5x_1 + 4x_2 \\
\text{s.t.} & 2x_1 + 3x_2 \geq 1 \\
& x_1 \geq 0,\ x_2 \leq 0
\end{array}
$$

D Prompt Formats

Figure 5 includes a visualization of the prompt format used in the experiments.

```
.................................................        .................................................
:                              prompt_template :        :                               primal_problem :
:                                               :        :                                               :
: You are an expert optimization                :        : Max 100 y[0] + 200 y[1] + 150 y[2]            :
: practitioner and mathematician.               :        : Subject to                                    :
: Your task is to read and understand the       :        :   x[0]_lb : x[0] ≥ 0                           :
: primal linear program below and produce       :        :   x[1]_lb : x[1] ≥ 0                           :
: python code for the dual linear program in    :        :   y[0]_lb : y[0] ≥ 0                           :
: python with gurobipy.                         :        :   y[1]_lb : y[1] ≥ 0                           :
:                                               :        :   y[2]_lb : y[2] ≥ 0                           :
: Here is the given primal problem:             :        :   Budget : 3 x[0] + 4 x[1] ≤ 4                 :
: {primal_problem}                              :        :   Coverage_0 : -x[0] + y[0] ≤ 0               :
:                                               :        :   Coverage_1 : -x[1] + y[1] ≤ 0               :
: Put all your code between two '=====' lines,  :        :   Coverage_2 : -x[0] + y[2] ≤ 0               :
: like this, and use the boiler plate code:     :        .................................................
: =====                                         :        .................................................
: {code_format}                                 :        :                                   code_format :
: =====                                         :        : import gurobipy as gp                         :
:                                               :        : from gurobipy import GRB, quicksum            :
: Generate the complete code, including the     :        :                                               :
: dual model definition, variables,             :        : def create_dual_model():                      :
: constraints, objective function.              :        :     model = gp.Model("dual_model")            :
: It must be runnable.                          :        :     # TODO: create the dual model             :
: Take a deep breath and think step by step     :        :     ...                                        :
: about the primal to dual conversion process.  :        :     model.optimize()                          :
:                                               :        :     return model                             :
.................................................        .................................................
```
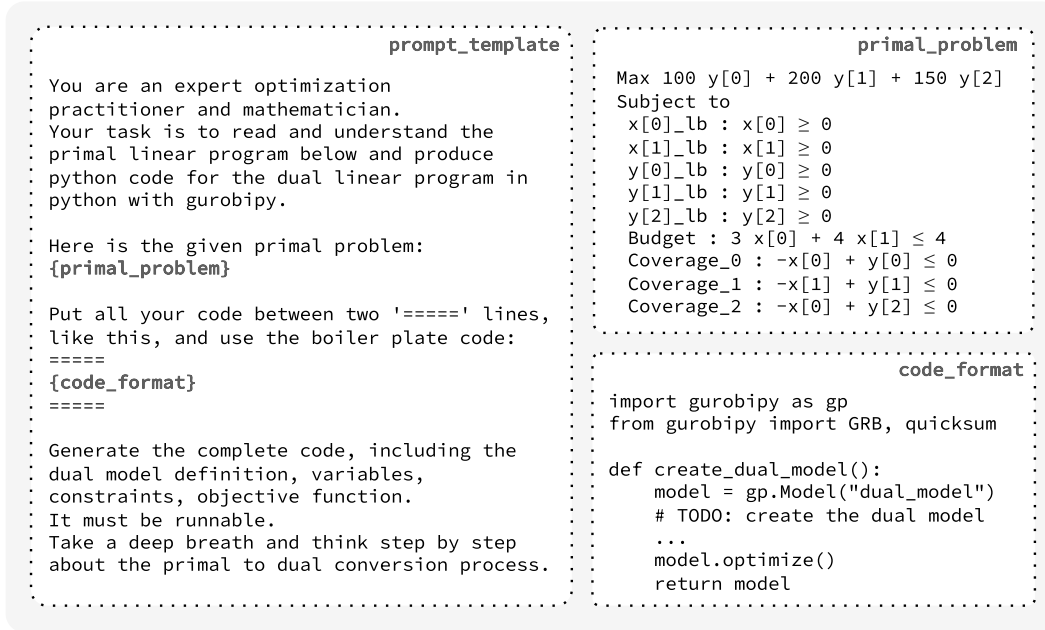
Figure 5: The prompt template used in the Section 5 experiments.