RIGHT ANSWERS, COSTLY MODELS: THE EFFICIENCY GAP IN LLM-BASED OPTIMIZATION MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimization modeling formulates real-world decision problems as mathematical programs that solvers can use to find optimal decisions. Large language models (LLMs) can automate this process, but the resulting correct formulations can require substantial time and memory to construct and solve, limiting practical scalability. Therefore, we systematically investigate whether LLMs can identify problem structure from natural-language descriptions and apply suitable optimization modeling techniques to generate mathematical models and solver code that solve the problems correctly and efficiently. To this end, we first curate **OptTips**, a knowledge base of 50 expert modeling techniques in eight families. Using this knowledge, we develop **OptDachshund**, a multi-agent framework that transforms problems from existing optimization benchmarks into new tasks for evaluating LLMs' use of modeling techniques. It constructs conventional and expert mathematical models with solver code for the same task and data, providing baselines for correctness and computational cost. The resulting **EfficientOpt** benchmark contains 561 expert-reviewed tasks with paired reference implementations. Evaluation of 11 representative LLMs reveals an efficiency gap on correctly solved tasks with comparable measurements: for every LLM, most generated programs take longer to solve than their expert counterparts. Within the comparable reference-size subset, 57% of programs with correct objective values and fewer variables and linear constraints have longer recorded solver times. Case studies show that different modeling techniques can achieve the same optimal value at similar recorded cost. Faster solving may not reduce execution time if the code takes longer to prepare data and build the model. LLM optimization modeling should therefore be evaluated for both correctness and computational efficiency.

1 INTRODUCTION

Optimization modeling connects real-world decision problems to mathematical solvers (Williams, 2013). Applications in scheduling, transportation, energy, and resource allocation depend on translating a problem into variables, an objective, and constraints (Pinedo, 2022; Toth & Vigo, 2014; Conejo et al., 2010). This translation requires expertise in choosing representations that solvers can handle effectively. LLMs offer a promising route to automating this work (Xiao et al., 2025). Agent-based approaches coordinate specialized modeling, programming, and verification roles (Xiao et al., 2024; AhmadiTeshnizi et al., 2024). Supervised fine-tuning (SFT) teaches models from problem–formulation pairs (Huang et al., 2025a; Jiang et al., 2025; Lu et al., 2025), while reinforcement learning (RL) uses solver-verified rewards (Chen et al., 2025b) or test-time policy optimization (Ding et al., 2026) to improve modeling. This progress motivates studying the computational efficiency of LLM-generated models: the time and memory needed to build and solve them. These costs limit problem sizes under practical resource budgets (Klotz & Newman, 2013; Vielma, 2015).

Executability and numerical correctness alone do not establish whether a generated optimization model is efficient to solve (Zhao et al., 2026). The same optimization problem can be addressed through different modeling and solution approaches, with sharply different computational costs (Williams, 2013; Vielma, 2015). For example, loose Big-M constants can weaken LP relaxations (Vielma, 2015); unnecessary integrality requirements can increase branching effort (van der Hulst &

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Walter, 2026). Unexploited symmetry can cause redundant search (Margot, 2010), while separable structure can enable decomposition into coordinated subproblems (Dantzig & Wolfe, 1960). Effective optimization modeling therefore requires recognizing problem structure and selecting modeling techniques that exploit it (Williams, 2013). However, simply asking an LLM to produce faster code does not guarantee an effective reformulation: efficiency-oriented prompting can introduce unreliable shortcuts or disrupt favorable structure (Wang et al., 2026). The central question is therefore: **Can LLMs recognize problem structure and apply appropriate optimization modeling techniques to obtain correct solutions at lower computational cost?**

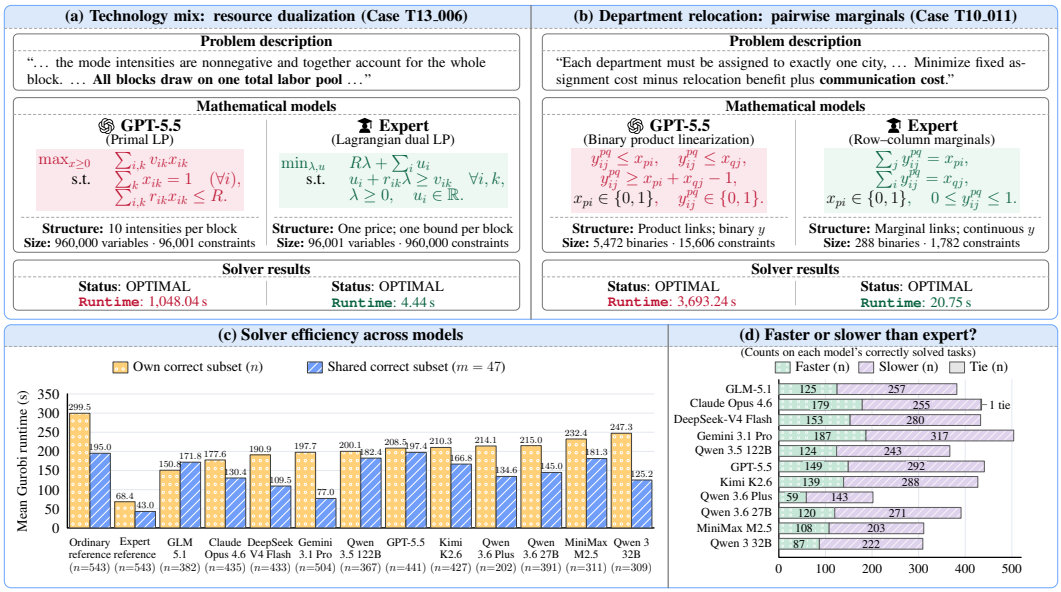


Figure 1: **Matching optimal values can conceal different computational costs.** Panels (a,b) present two case studies comparing GPT-5.5-generated and expert formulations on identical inputs; each pair reaches matching optimal objective values. **(a) Technology mixing case.** Production blocks share a labor budget. The generated primal LP models each block–mode intensity; the expert dual LP uses one resource-price variable and one bound variable per block. The generated model requires about 236× the recorded Gurobi solver time of the expert reference. **(b) Department relocation case.** Departments are assigned to cities with communication costs between them. The generated model uses binary interaction variables; the expert links continuous interaction variables to binary assignments through row and column sums. The generated model requires about 178× the recorded Gurobi solver time of the expert reference. **(c)** Mean Gurobi runtimes compare LLM-generated models with ordinary and expert reference baselines, using correctly solved tasks in the reference cost subset (gold) and its 47 tasks solved correctly by all 11 LLMs (blue). Expert references have the lowest means in both views; the gold subsets can differ across methods. **(d)** Green, purple, and gray bars count programs that are faster, slower, or tied with the expert on correctly solved tasks. For every LLM, most take longer than the expert, showing that numerical agreement alone does not establish efficient modeling.

Answering this question requires tasks that allow us to compare different modeling approaches and their computational costs. The two case studies in Figure 1(a,b) illustrate why: in technology mixing, the expert uses resource prices in a dual LP rather than explicitly modeling every production intensity; in department relocation, it retains binary assignments but represents their interactions with continuous variables linked through row and column sums. Both expert references attain the same optimal objective values as the corresponding GPT-5.5-generated models with substantially less recorded solver time (see Appendix G.1 for the mathematical checks). To study such decisions systematically, we complement recent work on solver efficiency and strategy-aware optimization modeling (Zhao et al., 2026; Kong et al., 2026a) with tasks designed around problem structures to which specific expert modeling techniques apply. Each task includes ordinary and expert references on the same data: the ordinary reference provides a conventional mathematical model and solver code, while the expert reference applies a selected technique. Two OptTips examples in Figure 3 illustrate this distinction: replacing a loose Big-M constant with a tighter valid one derived from variable bounds, and adding a valid inequality to strengthen an LP relaxation. These contrasts let us examine LLM modeling choices beyond answer correctness. Comparing the generated models and

code with the paired references allows us to assess correctness, technique use, and computational cost separately.

We first curate **OptTips**, a knowledge base of 50 expert modeling techniques in eight families, drawn from optimization textbooks, research literature, and solver documentation. Here, *optimization modeling techniques* are reusable ways to exploit mathematical structure when representing a problem for a solver or organizing it into coordinated submodels. OptTips contains 50 *technique cards*, one structured entry per technique. Each card explains when and how to apply the technique and which inefficiency it targets, with a mathematical example. To our knowledge, OptTips is the first expert-curated technique catalog designed to construct an LLM optimization-modeling benchmark that separately evaluates technique use and computational cost. Using this knowledge, we develop **OptDachshund**, a multi-agent framework for constructing tasks that test technique use. Its six agents match techniques to source problems, redesign, audit, and repair tasks, and build and validate ordinary and expert reference implementations on the same instance. Solver validation and independent expert review yield **EfficientOpt**, a benchmark of 561 problems spanning 29 target-technique groups. Each task includes a natural-language description, numerical data, and paired references for cost comparisons. Withholding references and target-technique labels tests whether LLMs independently recognize problem structure and apply suitable techniques.

Evaluation of 11 representative LLMs on this benchmark reveals five main findings: *1) Correct answers can still be costly.* Expert references have lower recorded mean solver runtimes (Figure 1(c)). For every LLM, most programs returning correct objective values take longer to solve than the corresponding expert implementation (Figure 1(d)). *2) A few slow cases dominate total solver time.* Similar median solve times can hide different mean costs; across models, the slowest tenth consumes roughly 70% of measured solver time on correctly solved tasks (Figure 10(b); Table 5). *3) Smaller formulations are not necessarily faster.* Our analysis compares variable, linear-constraint, and matrix-nonzero counts with solver runtimes (Section 4.4). *4) Technique use and observed cost are distinct.* Case studies identify different formulations attaining the same optimal value at similar recorded cost (Section 4.4). *5) Efficiency extends beyond solver runtime.* Faster solving can be offset by longer model-construction time (Section 4.5). We also report memory use (Table 10).

Our contributions are fourfold: **1) OptTips.** We curate 50 techniques in eight families into a knowledge base linking problem structure and inefficiency symptoms to expert modeling actions. **2) OptDachshund.** We develop a multi-agent framework that uses OptTips to construct tasks in three stages: match and reformulate, audit and repair, and build and validate. **3) EfficientOpt.** We construct a 561-task benchmark with ordinary and expert references and hidden technique labels to evaluate correctness, technique use, and computational efficiency separately. **4) Experiments and findings.** Our study of 11 LLMs examines correctness, technique use, model size, and generation, construction, solver, and memory costs. We find that correct objective values and compact formulations can still come at substantial cost, and compare alternative modeling choices with expert references. The EfficientOpt dataset, evaluation code, and OptTips knowledge cards are available in our anonymous repository.[1]

2 Related Work

Due to space constraints, we discuss only the most closely related work here and defer a broader review to Appendix A. *1) Expert knowledge for optimization modeling.* OptiMind (Zhang et al., 2025) uses expert hints to guide training and inference. In concurrent work, OptSkills (Yang et al., 2026b) distills modeling and solving trajectories into reusable workflows for inference. These methods use knowledge to assist generation. Our additional contribution is to turn individual techniques into specifications for constructing evaluation tasks. Our OptTips specifies applicability conditions and modeling actions, such as deriving tighter valid Big-M constants from variable bounds. Our OptDachshund turns these specifications into test problems and paired reference implementations. The target techniques and references are hidden from evaluated LLMs, so the benchmark tests independent technique selection and application. *2) Evaluating modeling quality and efficiency.* SAGE (Zhao et al., 2026) trains LLMs with correctness and solver-efficiency feedback and compares formulations generated by different models. In concurrent work, FrontierOR (Kong et al., 2026a) evaluates algorithm quality and runtime against one representative Gurobi reference per task. Their

[1] https://anonymous.4open.science/r/efficiency-EE0F/

reported evaluations do not provide a conventional/expert reference pair tied to a designated modeling technique. Our EfficientOpt adds this pair for each task on identical data, making the contrast between conventional and expert modeling explicit. We inspect technique use and separately assess numerical correctness, model-construction cost, and solving cost. This distinguishes technique adoption from efficient implementation and recognizes efficient solutions using other techniques.

3 Benchmark Construction

Definition 1 (Optimization modeling). An optimization task is $p = (q, I, \mathcal{O})$, where q specifies the problem in natural language, I is the fixed numerical input, and $\mathcal{O}$ specifies the required outputs and accuracy. *Optimization modeling* constructs a solution procedure $\widehat{\mathcal{A}} = (\widehat{\mathcal{M}}, \widehat{C})$, where $\widehat{\mathcal{M}}$ is a mathematical representation and $\widehat{C}$ is its executable implementation. The representation specifies variables, domains, objectives, and constraints, in a single formulation or coordinated submodels. The implementation handles any submodel coordination, stopping rules, and required answer recovery. A valid procedure must respect the problem semantics and satisfy $\mathcal{O}$: an optimal value alone is insufficient when original decisions are required. Different valid procedures may use different variables and feasible sets. Modeling techniques, such as tighter bounds, dualization, or decomposition, exploit problem structure in choosing and implementing these representations.

Definition 2 (Modeling efficiency). For the same task p and required answer quality, *modeling efficiency* concerns the time and memory needed to produce and execute a valid solution procedure. Lower cost indicates greater efficiency only for the stated resource, accounting scope, and execution conditions; time and memory are assessed separately. We distinguish *generation*, which produces $\widehat{\mathcal{A}}$, from *execution*, which runs $\widehat{C}$ on I. Execution includes data preparation, model construction, all solver calls, computation between calls, and answer recovery. Model construction specifically means executing code to instantiate variables, objectives, and constraints. Thus, faster solving can be offset by longer construction time, and neither smaller models nor technique adoption guarantees lower cost. Complete request costs also include retries, repairs, and checking. Appendix I details the definitions and the stages our measurements cover.

Overview. Figure 2 summarizes the construction workflow. We first curate our knowledge base *OptTips* (detailed in Sec. 3.1), whose cards specify when and how to apply modeling techniques. Guided by these cards, our multi-agent framework *OptDachshund* (detailed in Sec. 3.2) matches techniques to source problems, redesigns tasks, and builds paired ordinary/expert models with solver code on identical data. The resulting candidates then undergo solver verification and *independent expert review* (detailed in Appendix E) to check task and reference validity, technique use, leakage, and diversity. Failed checks trigger revision and re-verification; unresolved cases are rejected. Validated candidates form our benchmark *EfficientOpt* (detailed in Sec. 3.3).

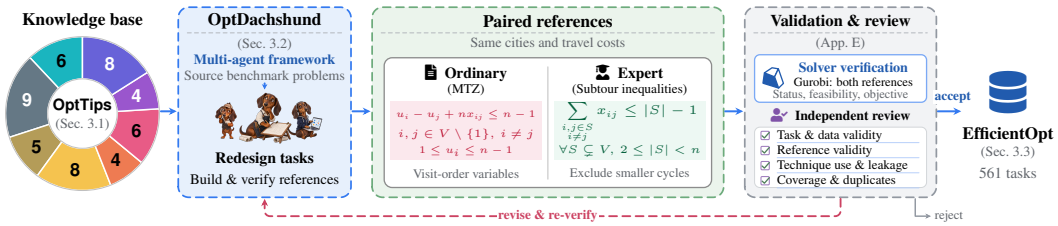


Figure 2: **Benchmark construction overview.** Our knowledge base OptTips (Sec. 3.1) guides our multi-agent framework OptDachshund (Sec. 3.2) to build tasks and paired references. Gurobi verification and independent expert review (Appendix E) lead to revision, rejection, or acceptance into our benchmark EfficientOpt (Sec. 3.3). The example seeks the cheapest tour visiting every city once. MTZ uses continuous visit-order variables u_i (excluding reference city 1); the expert model forbids disconnected cycles within each smaller city subset S. Here, V contains n cities and binary x_{ij} selects travel from city i to a different city j. Both models share travel costs and one-arrival/one-departure constraints, omitted for clarity.

3.1 OPTTIPS: EXPERT MODELING KNOWLEDGE

OptTips makes expert modeling knowledge explicit for task construction and evaluation. Drawing on optimization textbooks (Williams, 2013; Bertsimas & Tsitsiklis, 1997; Nemhauser & Wolsey, 1988; Boyd & Vandenberghe, 2004; Birge & Louveaux, 2011), research literature (Dantzig & Wolfe, 1960; Benders, 1962; McCormick, 1976; Margot, 2010; Vielma, 2015), and solver documentation (Gurobi Optimization, 2026a;c;d; IBM, n.d.; MOSEK ApS, 2025), we curate 50 modeling techniques, each represented by a structured *technique card*. The collection spans foundational formulation choices and advanced methods across eight primary methodological families (Figure 3). Its breadth concerns reusable modeling mechanisms, from bounds and reformulations to decomposition and structured optimization; 50 is the current collection size, not an exhaustive taxonomy. Appendix J provides the complete technique inventory, coverage rationale, and card schema. The full cards are available in our anonymous repository.

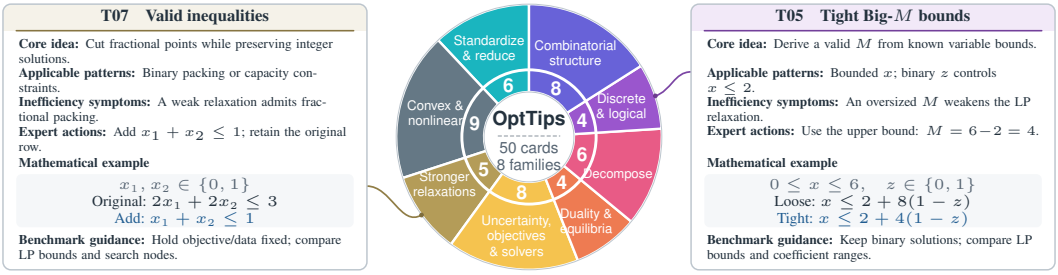


Figure 3: OptTips: 50 cards in eight families, with two abbreviated examples showing the seven-field structure. Full card examples appear in Appendix J (Figures 12 and 13).

Each OptTips card has seven fields. Taking the tight Big-M bounds card in Figure 3 as an example, we explain what each field records. The *(1) identifier* provides a reference label, here T05. The *(2) core idea* states the mathematical principle: derive a tight, valid M from known variable bounds. The *(3) applicable patterns* specify when the technique applies, such as encoding $z = 1 \Rightarrow x \leq 2$ with binary z and $0 \leq x \leq 6$. The *(4) inefficiency symptoms* describe what needs improvement; here, an oversized M leaves the LP relaxation unnecessarily loose. To address this, the *(5) expert actions* give the concrete modeling step: use the bound $x \leq 6$ to replace $M = 8$ with $M = 6 - 2 = 4$. The *(6) mathematical example* expresses the change in equations, comparing $x \leq 2 + 8(1 - z)$ with $x \leq 2 + 4(1 - z)$. Both have the same binary feasible solutions, but at relaxed $z = \frac{1}{2}$, the upper bound on x falls from 6 to 4. Finally, the *(7) benchmark guidance* explains how to construct and compare test models. For this technique, it calls for paired models with the same task, data, and objective: check that binary feasible solutions are unchanged, then compare LP objective bounds, coefficient ranges, and solve times. Appendix J provides eight full card examples, one per family.

We use OptTips to guide benchmark construction and to examine LLM modeling choices. During construction, a card specifies the problem structure needed for a technique and the change to implement in the expert reference. OptDachshund uses this information to redesign source problems and build ordinary and expert references for the same final task and data (Section 3.2). We assess technique use with an LLM judge across all generated programs and inspect selected cases manually (Appendices F.1 and F.3); Appendix G.1 gives case-level mathematical checks and their limitations. We assess efficiency from recorded runtime and resource use, independently of technique adoption.

3.2 OPTDACHSHUND: MULTI-AGENT BENCHMARK CONSTRUCTION

OptDachshund is a multi-agent framework that uses OptTips to transform source problems into candidate tasks with ordinary and expert reference implementations. Its six agents work in three stages: *(1) match and reformulate*, *(2) audit and repair*, and *(3) build and validate* (Figure 4).

Source problems. We draw problems from MAMO EasyLP and ComplexLP (Huang et al., 2025b), and OptMATH (Lu et al., 2025). We select self-contained problems that can be adapted to test an OptTips technique, with solutions we can check using the benchmark's solvers. Unclear or incomplete statements and duplicate or near-duplicate templates are flagged for revision or removal.

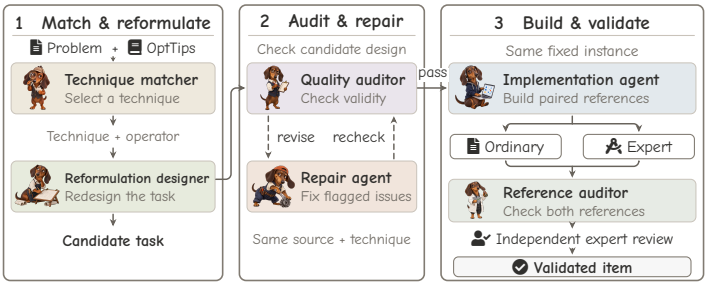


Figure 4: OptDachshund workflow: from source problems to validated tasks with ordinary and expert references.

1) Match and reformulate. The *Technique Matcher* selects an OptTips technique and proposes how to modify a source problem to test its use. The *Reformulation Designer* then writes the revised problem statement and data schema and prepares a formulation and solution for internal validation. For example, `T35_001` tests variable elimination in an allocation problem where each group's allocations must sum to a fixed total. Once all but one allocation are chosen, the remaining allocation is determined by subtraction. Its variable can therefore be replaced by the total minus the other allocations, with its lower and upper bounds imposed on that expression. The public statement gives the task requirements without revealing the intended technique, so the evaluated LLM must choose its own modeling approach.

2) Audit and repair. The *Quality Auditor* checks that the statement and data define a clear problem. It then checks whether the model correctly represents the stated decisions, constraints, and objective, and whether the selected technique applies. It also looks for duplicate tasks and hints that reveal the intended technique. The *Repair Agent* fixes the reported problems and returns the task for another check. For minor revisions, we keep the source problem and selected technique unchanged. If a revision changes the task requirements or the technique, we repeat the full audit.

3) Build and validate. After the design audit, the *Reference Implementation Agent* builds ordinary and expert references for the same redesigned task, data, and required outputs. The ordinary reference uses a conventional approach; the expert applies the selected technique. The *Reference Audit Agent* checks their formulations, code, and solver results. Appendix H gives implementation details and an example prompt. At least two experts review each candidate independently: one checks the public specification and another examines both references. A third expert resolves substantive disagreements. Reference runs and optimality evidence verify the answers. We require correct solutions and valid technique use. Appendix E explains the review criteria and revision process.

3.3 EFFICIENTOPT: BENCHMARK COMPOSITION AND INTERFACE

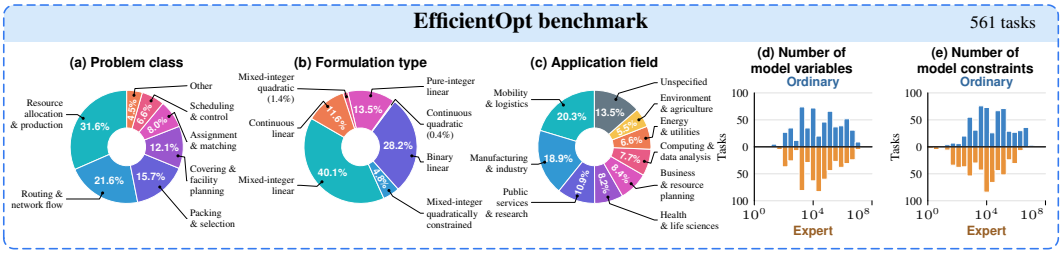


Figure 5: Task composition and reference model sizes for the 561 tasks in EfficientOpt. (a–c) Problem classes, formulation types of ordinary references, and application fields. (d,e) Variable and constraint counts for ordinary (blue, above) and expert (orange, below) references, shown on logarithmic horizontal axes. Appendix B gives the classifications and counting rules.

The resulting benchmark EfficientOpt contains 561 test tasks. These tasks span different problem classes and application fields, with reference models that vary in formulation type and size (Figure 5). *1) Problem classes.* The tasks cover eight classes, including resource allocation and production (31.6%), routing and network flow (21.6%), and scheduling and control (6.6%). Two small classes are grouped as *Other* in Figure 5(a). *2) Formulation types.* The ordinary reference models fall into seven types, including mixed-integer linear (40.1%), binary linear (28.2%), and continuous linear (11.6%); quadratic formulations are also represented (Figure 5(b)). *3) Application fields.* The

tasks cover eight named fields, including mobility and logistics, manufacturing and industry, and health and life sciences. Tasks without a clear application context are labeled *Unspecified* (Figure 5(c)). *4) Reference-model sizes.* Across all 561 tasks, reference model sizes span several orders of magnitude (Figure 5(d,e)). We take the maximum variable and total constraint counts over each reference's models before presolve. Expert references have fewer variables in 43.1% of pairs and fewer constraints in 51.0%. Appendix B details the classifications and counting rules.

Task interface. The task interface specifies what an LLM receives and what it must produce. Each task provides a natural-language problem statement, specifying the required outputs, and a schema for the input data. The LLM writes executable solver code for its chosen formulation. We run the code on the task's fixed numerical instance using Gurobi. The technique label, both reference implementations, and validation records are hidden from the LLM.

Paired references. The ordinary reference uses a conventional approach, while the expert reference applies the selected OptTips technique to the same task and data. We use verified task objectives to check the LLM's numerical answer and reference execution records to compare construction and solver costs separately. An LLM judge assesses technique use from code, with manual inspections of selected cases (Appendices F.1 and F.3). A different approach can also produce a correct, efficient solution. Section 4.1 defines the numerical checks and cost measures.

4 EXPERIMENTS

We evaluate LLM-generated programs in terms of numerical correctness, computational cost, and modeling choices. Four questions guide our analysis: **RQ1:** How does numerical accuracy vary across tasks? **RQ2:** For correctly solved tasks, how do computational costs compare with the ordinary and expert references? **RQ3:** Do smaller models solve faster, and which modeling techniques do LLMs use? **RQ4:** What time and memory costs are missed when we measure only solver time?

4.1 EXPERIMENTAL SETUP

Tasks and evaluation. We evaluate the 11 general-purpose LLMs in Table 1 on all 561 EfficientOpt tasks. We executed each task three times on the same machine with the same settings. Recorded results varied slightly, so the main tables report a typical run. Each LLM generates solver code from the problem description. We count a run as numerically correct if Gurobi returns `OPTIMAL` and its finite objective matches the verified reference value (absolute and relative tolerances: 10^{-6}). Appendix B details task statistics, code generation, and answer verification; Appendix C adds accuracy breakdowns and detailed cost analyses. We also evaluate five fine-tuned models. Their low accuracy limits efficiency comparisons, so we report their accuracy and failure analysis separately in Appendix C.6.

Evaluation metrics. We report accuracy, technique use, and code generation and execution costs:

- Accuracy (%): percentage of all 561 tasks solved correctly under the numerical check above.
- `Runtime` (s): elapsed optimization time reported by Gurobi.
- `Work` (units): Gurobi's measure of the computation performed by the solver.
- `Build` (s): recorded preparation time, including preprocessing and any solves within that stage.
- `Build+opt` (s): `Build` plus elapsed time in the separately timed solver calls.
- `API latency` (s): time for the final generation call, including service and network delays.
- `Tokens`: provider-reported input, output, and, where available, reasoning token counts.
- `RSS`: RAM occupied by the process, sampled during model preparation and solving.
- `Gurobi peak`: peak memory allocated within the Gurobi environment.

Appendix D gives the measurement boundaries and comparison rules.

4.2 NUMERICAL OUTCOMES VARY ACROSS TASKS (RQ1)

An optimal solver status is not sufficient. Gurobi solves the supplied model, which may omit required constraints. Of 6,171 generated programs, 4,602 report `OPTIMAL`, but 338 of these return incorrect objective values. We therefore verify numerical answers before comparing costs: solving an incorrect model quickly does not establish an efficiency gain.

Overall accuracy hides large differences across tasks. The models are much more successful on some task types than others. Across the evaluated models, accuracy is 93.9% on tasks that split into independent subproblems, but only 27.3% on the five tasks targeting column generation (adding variables as needed during optimization). Success on one kind of problem therefore does not ensure reliable answers on another. Results by task group show where errors recur and closer inspection is needed (see Appendix C.2 for accuracy breakdowns by task group).

4.3 Correct Answers Can Still Be Costly (RQ2)

Correct answers still leave room for faster solving. Table 1 compares costs with both references on each LLM's eligible tasks. All 11 LLMs have lower aggregate `Runtime` than the ordinary reference, but higher `Runtime` than the expert (ratios 1.49–2.03). Higher `Work` ratios show that this gap also appears in measured solver effort. Including model preparation does not close it: every `Build+opt` ratio against the expert remains above one. Thus, outperforming a conventional implementation leaves room to reduce computational cost. Using both references makes this visible; an ordinary-reference comparison alone would miss the remaining gap to expert performance.

Table 1: Numerical accuracy and computational cost. Accuracy uses all 561 tasks. Cost ratios use complete measurements on correctly solved tasks within the 543-task reference cost subset (LLM/reference; below 1 means lower cost). See Appendix D for metric definitions and cost-ratio calculations.

	Correct answers		Cost ratio (LLM/reference)			
			`Runtime`		`Work`	`Build+opt`
Model	Count ↑	Rate (%) ↑	Ordinary ↓	Expert ↓	Expert ↓	Expert ↓
Gemini 3.1 Pro	512	91.3	0.52	1.68	1.98	1.43
GPT-5.5	447	79.7	0.58	1.79	2.08	1.64
Claude Opus 4.6	442	78.8	0.46	1.49	1.77	1.42
DeepSeek-V4 Flash	441	78.6	0.49	1.62	1.78	1.47
Kimi K2.6	433	77.2	0.55	1.79	2.02	1.59
Qwen 3.6 27B	394	70.2	0.61	1.94	1.97	1.99
GLM-5.1	390	69.5	0.50	1.68	1.92	1.44
Qwen 3.5 122B	369	65.8	0.55	1.82	1.99	1.73
MiniMax M2.5	315	56.1	0.59	1.80	2.01	1.77
Qwen 3 32B	313	55.8	0.66	2.03	2.08	1.83
Qwen 3.6 Plus	208	37.1	0.52	1.83	2.22	1.63

A few slow solutions account for most solving time. For each LLM, the slowest ∼10% of runs with correct objective values and valid timings consume 68.4–76.2% of total `Runtime` (Figure 6). Median runtimes are 23.1–36.0 seconds, but summed `Runtime` over each model's eligible tasks is 12.0–27.7 hours. Appendix C.7 identifies the slow tasks for all 11 models and compares them with the expert references. As a sensitivity calculation, halving only the slowest tenth would reduce total solver time by 34.2–38.1% without changing any model's median.

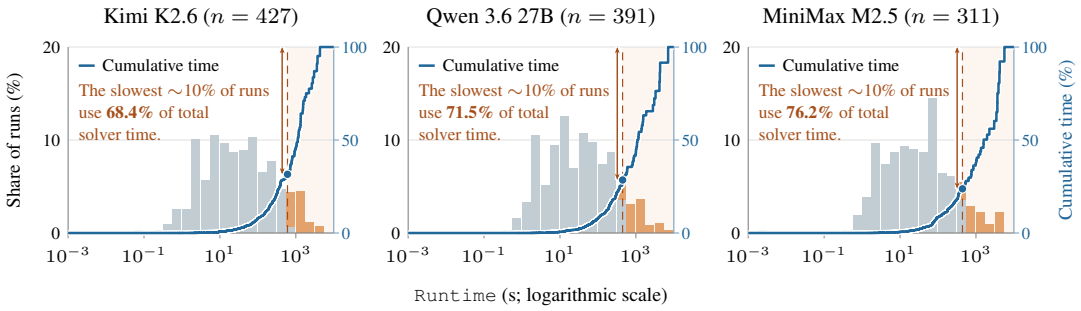


Figure 6: Runtime distributions and cumulative time. Bars show the percentage of runs in each time interval (left axis). Blue curves show the share of total `Runtime` used by runs at or below each runtime (right axis). Orange highlights the slowest ∼10%; arrows show their contribution to total time. Only correctly solved tasks with valid timings are included. See Appendix C.3 for per-model runtime statistics.

4.4 Modeling Choices and Solver Efficiency (RQ3)

Smaller models do not necessarily solve faster. Of the 735 outputs that passed our numerical checks and had smaller reported models than their expert references, 449 (61.1%) still took longer to solve (Figure 7(a)). Variable and constraint counts describe model size, but they do not fully capture how difficult the problem is for the solver. We therefore measure `Runtime` directly to assess whether a smaller model is faster to solve. Appendix C.4 reports per-model results under two size criteria and examines the magnitude of the slowdowns.

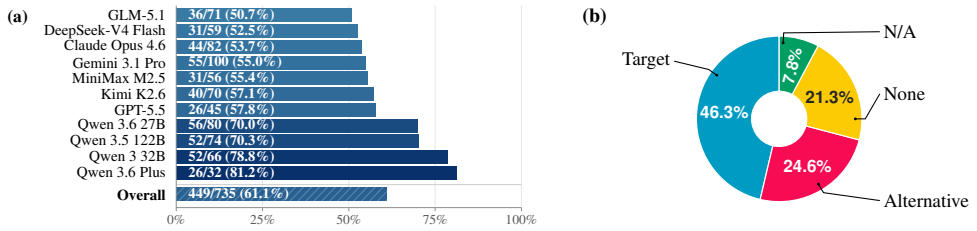


Figure 7: Size and technique use. (a) Among correct programs with smaller formulations, how many are slower than the expert? Labels give slower/total; the striped bar combines all 11 LLMs. (b) Technique-use labels for 6,171 programs. Appendix C.4 analyzes size and runtime; Appendix F.1 details technique use.

The same answer can come from different modeling choices. In department relocation (T10_011), GPT-5.5 adds binary variables to represent pairs of department assignments. The expert represents these interactions with continuous variables linked to the original assignments, reducing the binary count from 5,472 to 288. These formulations differ in variable types and linking constraints, distinctions that objective values and runtimes alone do not reveal. Appendix G.1 explains their integer equivalence and different LP relaxations.

LLMs use different modeling approaches. Each task has a designated OptTips technique. Our LLM-based assessment identifies it or an equivalent implementation in 46.3% of programs, and another technique in 24.6% (Figure 7(b)). These labels do not establish correctness or speed. For T13_006, Gemini models technology allocations directly, while the expert reformulates the problem using resource prices. Both return the same objective, with recorded preparation and solving times (`Build+opt`) of 16.18 and 16.27 seconds. Appendix F.1 gives technique-use breakdowns by model and task group; Appendix G examines the costs and limits of this comparison.

4.5 Solving Is Only Part of the Cost (RQ4)

Model preparation can change which implementation is faster. Among 4,202 correct runs in the cost subset, 17.3% spend longer on `Build` than on optimization. Adding `Build` to the externally timed optimization stage reverses the LLM–expert speed comparison in 8.6% of 4,148 pairs. These reversals occur for all 11 LLMs. Appendix C.5 gives per-model results and sensitivity checks; Appendix G.2 shows how construction offsets faster solving on T20_011.

Waiting for generated code can take longer than running it. Across all available responses, per-model median `API latency` ranges from 14.0 to 231.2 seconds. Among 4,202 correct runs with both measurements, generation takes longer than recorded construction and solving (`Build+opt`) in 55.6% of cases, rising to 76.2% for Qwen 3.6 27B and 74.1% for Qwen 3 32B. Appendix C.5 compares generation latency and token profiles across models.

Typical memory use can hide demanding cases. Across all 4,202 correct runs in the cost analysis, median `Gurobi peak` is 0.135 GB, while the 90th percentile is 2.314 GB and 224 runs (5.3%) reach at least 4 GB. Appendix C.5 compares solver and process memory across models.

5 Discussion and Conclusion

We introduced OptTips, which codifies 50 expert techniques in knowledge cards linking problem structure and inefficiency symptoms to concrete modeling actions. OptDachshund turns this knowledge into a six-agent workflow for task reformulation, auditing, repair, and paired reference construction and validation. EfficientOpt provides 561 tasks with ordinary and expert implementations on the same instances, allowing numerical correctness, technique use, and computational cost to be evaluated separately. Across the 11 evaluated LLMs, programs returning correct objective values use less aggregate Gurobi solver time than ordinary references but more than expert references on comparable task subsets. Smaller formulations can still solve more slowly, and model preparation can offset faster solving. Future work can extend this evaluation across solvers, instance scales, and iterative methods. Progress should be assessed by both solution correctness and computational cost.

AI USE STATEMENT

LLM-based agents are components of the benchmark-construction method described in Section 3, and an LLM-as-judge protocol assesses technique use as described in Appendix F.1. AI assistance was also used for manuscript organization and language editing. The authors take full responsibility for the content and accuracy of the final manuscript.

REPRODUCIBILITY STATEMENT

Reproducibility is supported by task specifications, numerical instances, reference implementations, execution records, and analysis scripts. Appendix B defines evaluation coverage; Appendix E describes validation; Appendix D defines measurement scopes; Appendix G provides mathematical checks and case-level cost comparisons; Appendix F documents the judge snapshot and its limitations. Source and input identifiers, execution conditions, and reasons for missing measurements are retained. Generation and execution costs are reported within their measured scopes; the available records do not reconstruct the full cost of all attempts. The code and EfficientOpt dataset are available through our anonymized GitHub repository at `https://anonymous.4open.science/r/efficiency-EE0F/`.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 2024.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela Van Der Schaar. Autoformulation of mathematical optimization models using LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 1864–1886. PMLR, 2025.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. doi: 10.1007/BF01386316.

Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997. ISBN 9781886529199.

John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer, 2nd edition, 2011. ISBN 9781461402367. doi: 10.1007/978-1-4614-0237-4.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN 9780521833783. doi: 10.1017/CBO9780511804441.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for 2+3=? on the overthinking of long reasoning models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 9487–9499. PMLR, 2025a.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 106027–106069, 2025b.

Antonio J. Conejo, Miguel Carrión, and Juan M. Morales. *Decision Making Under Uncertainty in Electricity Markets*, volume 153 of *International Series in Operations Research & Management Science*. Springer, 2010. ISBN 9781441974204. doi: 10.1007/978-1-4419-7421-1.

George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960. doi: 10.1287/opre.8.1.101.

Zezhen Ding, Zhen Tan, Jiheng Zhang, and Tianlong Chen. OR-R1: Automating modeling and solving of operations research optimization problem via test-time reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 228–236, 2026. doi: 10.1609/aaai.v40i1.36983.

Google. The job shop problem. OR-Tools documentation, 2024. CP-SAT interval variables and NoOverlap constraints; accessed 2026-09-21.

Gurobi Optimization. Gurobi Optimizer Reference Manual: Constraints, 2026a. SOS, indicator, piecewise-linear, and general constraints; accessed 2026-09-21.

Gurobi Optimization. Gurobi Optimizer Reference Manual: Model attributes, 2026b. Runtime, Work, MemUsed, and MaxMemUsed; accessed 2026-09-17.

Gurobi Optimization. Gurobi numerical guidelines: Tolerances and user-scaling, 2026c. Accessed 2026-09-17.

Gurobi Optimization. Gurobi Optimizer Reference Manual: Variable attributes, 2026d. Starts, hints, priorities, and basis attributes; accessed 2026-09-17.

Gurobi Optimization. What is performance variability?, 2026e. Official support documentation; accessed 2026-09-17.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025a. doi: 10.1287/opre.2024.1233.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710, 2025b.

IBM. Best practices with indicator constraints, n.d. CPLEX 12.9.0 documentation; accessed 2026-09-21.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ed Klotz and Alexandra M. Newman. Practical guidelines for solving difficult linear programs. *Surveys in Operations Research and Management Science*, 18(1–2):1–17, 2013. doi: 10.1016/j.sorms.2012.11.001.

Minwei Kong, Chonghe Jiang, Ao Qu, Wenbin Ouyang, Zhaoming Zeng, Xiaotong Guo, Zhekai Li, Junyi Li, Yi Fan, Xinshou Zheng, et al. Frontieror: Benchmarking llms' capacity for efficient algorithm design in large-scale optimization. *arXiv preprint arXiv:2605.25246*, 2026a.

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, et al. Alphaopt: Formulating optimization programs with self-improving llm experience library. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2389–2400, 2026b.

Zhong Li, Zihan Guo, Xiaohan Lu, Juntao Wang, Jie Song, Chao Shen, Jiageng Wu, and Mingyang Sun. OptArgus: A multi-agent system to detect hallucinations in LLM-based optimization modeling. *arXiv preprint arXiv:2605.11738*, 2026a. doi: 10.48550/arXiv.2605.11738.

Zhong Li, Qi Huang, Yuxuan Zhu, Mohammad Mohammadi Amiri, Niki van Stein, Thomas Bäck, Matthijs van Leeuwen, Zaiwen Wen, and Lincen Yang. MM-OptBench: A solver-grounded benchmark for multimodal optimization modeling. *arXiv preprint arXiv:2605.12154*, 2026b. doi: 10.48550/arXiv.2605.12154.

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026c.

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye Hao. OptiTree: Hierarchical thoughts generation with tree search for LLM optimization modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 120713–120781, 2025.

Haoyang Liu, Yuyang Cai, Jie Wang, Xiongwei Han, Minyang Hu, Shuqi LIU, Mingxuan Yuan, Jianye HAO, and Feng Wu. Opt-miner: Empowering information-seeking agent with tree-guided data synthesis for optimization modeling. In *Forty-third International Conference on Machine Learning*, 2026a.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, et al. Opt-verifier: Unleashing the power of llms for optimization modeling via dual-side verification. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b.

Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory Driven by Influential Applications*, INFORMS TutORials in Operations Research, pp. 1–12. INFORMS, 2013. doi: 10.1287/educ.2013.0112.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40769–40802. PMLR, 2025.

Hongliang Lu, Zhong Li, Yuxuan Chen, Yuan Lan, Fan Zhang, and Zaiwen Wen. Pearl: Solver-in-the-loop interactive optimization modeling from natural language. *arXiv preprint arXiv:2607.18256*, 2026.

François Margot. Symmetry in integer linear programming. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey (eds.), *50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art*, pp. 647–686. Springer, Berlin, Heidelberg, 2010. doi: 10.1007/978-3-540-68279-0_17.

Garth P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976. doi: 10.1007/BF01580665.

MOSEK ApS. *MOSEK Modeling Cookbook*, 2025. Release 3.4.0, November 5, 2025; accessed 2026-09-21.

George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988. ISBN 9780471828198. doi: 10.1002/9781118627372.

Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 6th edition, 2022. ISBN 9783031059209. doi: 10.1007/978-3-031-05921-6.

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 *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2022.

Dania Refai and Moataz Ahmed. A component-level evaluation framework for diagnosing llm errors in optimization modelling. *Neurocomputing*, pp. 133981, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Paolo Toth and Daniele Vigo (eds.). *Vehicle Routing: Problems, Methods, and Applications*, volume 18 of *MOS-SIAM Series on Optimization*. Society for Industrial and Applied Mathematics, 2nd edition, 2014. ISBN 9781611973587. doi: 10.1137/1.9781611973594.

Rolf van der Hulst and Matthias Walter. Implied integrality in mixed-integer optimization. *Mathematical Programming*, 2026. doi: 10.1007/s10107-026-02389-3.

Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57 (1):3–57, 2015. doi: 10.1137/130915303.

Haoyu Wang, Yuliang Song, Tao Li, Zhiwei Deng, Yaqing Wang, Deepak Ramachandran, Eldan Cohen, and Dan Roth. Formalize, don't optimize: The heuristic trap in llm-generated combinatorial solvers. *arXiv preprint arXiv:2605.12421*, 2026.

Zhuohan Wang, Ziwei Zhu, Yizhou Han, Yufeng Lin, Zhihang Lin, Ruoyu Sun, and Tian Ding. Optibench: Benchmarking large language models in optimization modeling with equivalence-detection evaluation. OpenReview manuscript, 2024.

Zhuohan Wang, Ziwei Zhu, Ziniu Li, Congliang Chen, Yizhou Han, Yufeng Lin, Zhihang Lin, Angyang Gu, Xinglin Hu, Ruoyu Sun, et al. Orgeval: Graph-theoretic evaluation of llms in optimization modeling. *arXiv preprint arXiv:2510.27610*, 2025.

H. Paul Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 5th edition, 2013. ISBN 9781118443330.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2024.

Ziyang Xiao, Jingrong Xie, Lilin Xu, Shisi Guan, Jingyan Zhu, Xiongwei Han, Xiaojin Fu, WingYin Yu, Han Wu, Wei Shi, Qingcan Kang, Jiahui Duan, Tao Zhong, Mingxuan Yuan, Jia Zeng, Yuan Wang, Gang Chen, and Dongxiang Zhang. A survey of optimization modeling meets LLMs: Progress and future directions. *arXiv preprint arXiv:2508.10047*, 2025. doi: 10.48550/arXiv.2508.10047.

Beinuo Yang, Qishen Zhou, Junyi Li, Chenxing Su, Panagiotis Angeloudis, and Simon Hu. Orthought: Benchmarking and automating logistics optimization modeling via structured llm reasoning. *Artificial Intelligence for Transportation*, 6:100059, 2026a.

Haochen Yang, Ke Zhao, Mengyuan Ma, Xingyu Lu, Xiangfeng Wang, and Hong Qian. OptSkills: Learning generalizable optimization skills from problem archetypes via cluster-based distillation. *arXiv preprint arXiv:2605.29829*, 2026b.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *International Conference on Learning Representations*, volume 2025, pp. 24726–24759, 2025.

Jingyang Yi, Jiazheng Wang, and Sida Li. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1302.

Xinzhi Zhang, Zeyi Chen, Humishka Zope, Hugo Barbalho, Konstantina Mellou, Marco Molinaro, Janardhan Kulkarni, Ishai Menache, and Sirui Li. Optimind: Teaching llms to think like optimization experts. *arXiv preprint arXiv:2509.22979*, 2025.

Ruiqing Zhao, Fengzhi Li, Yuan Zuo, Rui Liu, YanSong Liu, Yunfei Ma, Fanyu Meng, and Junlan Feng. Strategy-aware optimization modeling with reasoning LLMs. In *Forty-third International Conference on Machine Learning*, 2026.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. StepORLM: A self-evolving framework with generative process supervision for operations research language models. In *International Conference on Learning Representations*, volume 2026, pp. 6914–6940, 2026.

Appendix

Table of Contents

A Extended Related Work . . . 14
A.1 Generating and Refining Optimization Models . . . 15
A.2 Expert Knowledge and Reusable Modeling Skills . . . 15
A.3 Benchmark Coverage and Semantic Verification . . . 15
A.4 Evaluating Computational Efficiency . . . 16
B Benchmark Composition and Evaluation Coverage . . . 16
B.1 Task Composition and Classification . . . 16
B.2 Evaluation Coverage and Reference Verification . . . 17
B.3 Code Generation Protocol . . . 17
C Detailed Experimental Results . . . 17
C.1 Numerical Outcomes (RQ1) . . . 17
C.2 Accuracy Across Task Groups (RQ1) . . . 18
C.3 Solver Costs Across Tasks and Models (RQ2) . . . 18
C.4 Model Size and Solver Time (RQ3) . . . 22
C.5 Costs Beyond Solver Time (RQ4) . . . 22
C.6 Results for Fine-Tuned Models . . . 24
C.7 Slow Runs and Total Solver Time (RQ2) . . . 25
D Evaluation Metrics and Comparison Protocol . . . 26
D.1 Metrics and Measurement Scope . . . 26
D.2 Task Selection and Result Aggregation . . . 27
E Dataset Quality Control and Expert Verification . . . 28
E.1 Independent Review and Information Access . . . 28
E.2 Acceptance Criteria . . . 28
E.3 Decisions, Repair, and Corpus Review . . . 30
F Assessing Modeling Technique Use . . . 30
F.1 Automated Assessment of Technique Use . . . 30
F.2 Combining Technique Labels with Execution Outcomes . . . 30
F.3 Manual Inspection of Selected Modeling Choices . . . 32
G Case Studies on Modeling Choices and Efficiency . . . 32
G.1 What the Compared Formulations Preserve . . . 32
G.2 How Construction Changes the Cost Comparison . . . 33
H OptDachshund: Agent Roles and Information Flow . . . 33
I Task Validity and Computational Cost . . . 34
I.1 Valid Solution Procedures . . . 34
I.2 Resource Accounting and Comparison Scope . . . 35
J OptTips: Technique Catalog, Coverage, and Extension . . . 35
J.1 Rationale and Coverage . . . 35
J.2 Card Structure and Extension . . . 36
J.3 Complete Technique Inventory . . . 36

A Extended Related Work

We situate EfficientOpt within four connected research directions: generating optimization models, using reusable modeling knowledge, designing benchmarks and verification methods, and evaluating computational efficiency. These directions overlap: trained models can also use search, retrieval, and solver feedback during inference.

A.1 Generating and Refining Optimization Models

Inference workflows organize the steps between a problem description and executable solver code. Chain-of-Experts (Xiao et al., 2024) coordinates specialized agents, while OptiMUS (AhmadiTeshnizi et al., 2024) uses a modular workflow to formulate models, generate and debug code, and evaluate solutions. AutoFormulation (Astorga et al., 2025) and OptiTree (Liu et al., 2025) use tree search to explore candidate modeling decisions. These methods address how to construct and revise a candidate when a single generation is insufficient.

Learning-based methods improve the modeler's parameters through specialized data and feedback. ORLM (Huang et al., 2025a) trains on synthetic optimization-modeling instructions; LLMOPT (Jiang et al., 2025) uses multi-instruction tuning for mathematical formalization and code generation. OptMATH (Lu et al., 2025) and ReSocratic (Yang et al., 2025) synthesize mathematical models and corresponding natural-language problems to support training. Solver feedback provides another source of supervision: SIRL (Chen et al., 2025b) derives rewards from executable code and instance-level model verification; StepORLM (Zhou et al., 2026) combines solver outcomes with generative process supervision. OR-R1 (Ding et al., 2026) combines supervised fine-tuning with test-time policy optimization, while PEARL (Lu et al., 2026) learns interactive execution and revision using Python and solver feedback. Together, these approaches motivate evaluating both the mathematical model and its implemented solution procedure.

A.2 Expert Knowledge and Reusable Modeling Skills

Knowledge-augmented approaches differ in what they store and how it guides modeling. Opt-Miner (Liu et al., 2026a) studies information seeking for optimization modeling. AlphaOPT (Kong et al., 2026b) maintains a library of solver-verified insights with applicability conditions, explanations, and examples. OptiMind (Zhang et al., 2025) uses expert analyses of errors within problem classes to develop guidance for training and inference. In concurrent work, OptSkills (Yang et al., 2026b) distills modeling and solving trajectories into workflows and pitfalls organized by problem archetype, then retrieves these skills during inference. Its examples include activation variables, disjunctive constraints, and Big-M formulations, demonstrating that reusable workflows already contain concrete modeling knowledge.

Our OptTips organizes knowledge at the level of individual modeling techniques for task design and formulation inspection. A card connects applicability conditions and inefficiency symptoms to modeling actions and a mathematical example. Our OptDachshund uses these specifications to redesign tasks and construct ordinary and expert reference implementations on the same numerical instance. This use of technique-level knowledge for benchmark construction complements its use as assistance during generation; the evaluated LLMs receive neither the target-technique cards nor the reference implementations.

A.3 Benchmark Coverage and Semantic Verification

Benchmarks differ in both the problems they cover and the evidence used to assess a generated model. NL4Opt (Ramamonjison et al., 2022) studies the translation of natural-language descriptions into linear-programming formulations. ComplexOR (Xiao et al., 2024) and NLP4LP (AhmadiTeshnizi et al., 2024) extend evaluation to more complex modeling requirements and descriptions. IndustryOR (Huang et al., 2025a) and MAMO (Huang et al., 2025b) broaden application and mathematical-modeling coverage. OptiBench (Yang et al., 2025) includes linear and nonlinear problems with or without tabular data, while OptMATH (Lu et al., 2025) contributes challenging instances through its synthesis and filtering process.

Recent benchmarks extend scale and input modality. MIPLIB-NL (Li et al., 2026c) reconstructs 223 industrial MILP instances from MIPLIB 2017 as natural-language specifications with separate numerical data, using independent reconstruction and expert review to check fidelity. MM-OptBench (Li et al., 2026b) combines text with structured visuals in 780 solver-verified instances across six problem families. Its diagnostics separate input-extraction errors from formulation and implementation failures. These resources broaden the settings in which modeling capabilities can be tested.

Verification also extends beyond matching an objective value. The equivalence-oriented OptiBench (Wang et al., 2024) and the graph-based ORGEval framework and Bench4Opt dataset (Wang et al.,

2025) examine structural relationships between formulations. Component-level evaluation (Refai & Ahmed, 2026) diagnoses variables, objectives, and constraints separately. OptArgus (Li et al., 2026a) audits consistency across descriptions, symbolic formulations, and code, measuring false alarms, error localization, and detection. Opt-Verifier (Liu et al., 2026b) combines structure-side and solution-side verification. These distinctions inform our separation of numerical agreement from mathematical inspection: a matching objective alone does not establish that a modeling transformation preserves the task's required outputs.

A.4 Evaluating Computational Efficiency

Efficiency can concern generating an answer, checking it, or executing the resulting solver program. Test-time scaling and overthinking studies examine how reasoning effort translates into answer quality (Snell et al., 2025; Wu et al., 2025; Chen et al., 2025a; Yi et al., 2025). In optimization modeling, ORThought (Yang et al., 2026a) reports token efficiency, and component-level evaluation (Refai & Ahmed, 2026) includes latency and token usage. OptArgus (Li et al., 2026a) measures the cost of running its auditor and discusses efficiency-related modeling issues in its error taxonomy. These resource scopes differ from the construction and solution of the generated optimization model.

Several works directly examine downstream computation. SAGE (Zhao et al., 2026) trains with correctness and solver-efficiency feedback from alternative modeling strategies and evaluates solver time, iterations, and formulation size. Experiments on CP-SynC-XL (Wang et al., 2026) show that efficiency-oriented prompting can yield small or inconsistent speedups and can compromise correctness through unreliable search modifications. In concurrent work, FrontierOR (Kong et al., 2026a) evaluates solution quality and runtime for LLM-designed optimization algorithms against a representative formulation and Gurobi implementation for each task. Its scope includes reformulation and decomposition, and its references are also hidden from evaluated models.

Our EfficientOpt focuses this evaluation on designated modeling techniques. Each task is constructed around an applicable technique and paired with ordinary and expert references for the same task and data. The pair makes the conventional and technique-based implementations explicit comparison points. We inspect technique adoption separately from numerical correctness and recorded generation, model-construction, solving, and memory costs. This design supports distinguishing non-adoption of the designated technique, costly implementations that adopt it, and effective alternatives. The contribution is this combination of technique-guided task construction, paired references, and separate assessments, building on existing work on modeling knowledge and computational efficiency.

B Benchmark Composition and Evaluation Coverage

EfficientOpt contains 561 tasks covering 29 of the 50 OptTips techniques. The remaining 21 techniques have 63 supplementary tasks, three per technique, documented in Appendix J and excluded from the reported evaluation. This appendix describes task classification, reference verification, measurement coverage, and code generation.

B.1 Task Composition and Classification

Problem categories and application fields. Each task receives one primary problem category and one application label based on its statement and numerical instance (Table 2). Tasks without a clear application context are labeled *Unspecified*. These categories describe the decision problem and its context; the target modeling technique is recorded separately.

Table 2: Task counts by problem category (left) and application field (right), each totaling 561. Figure 5(a) combines fitting and estimation with layout planning as *Other* (25 tasks).

Primary problem category	n	Application field	n
Resource allocation & production	177	Mobility & logistics	114
Routing & network flow	121	Manufacturing & industry	106
Packing & selection	88	Unspecified	76
Covering & facility planning	68	Public services & research	61
Assignment & matching	45	Business & resource planning	47
Scheduling & control	37	Health & life sciences	46
Fitting & estimation	21	Computing & data analysis	43
Layout planning	4	Energy & utilities	37
		Environment & agriculture	31
Total	**561**	**Total**	**561**

Formulation types. We classify each task's ordinary reference by its objective, constraints, and variable domains before presolve. The 561

references fall into seven mutually exclusive classes: MILP (225), binary IP (158), nonbinary pure IP (76), LP (65), MIQCP (27), MIQP (8), and QP (2), as shown in Figure 5(b).

The classification follows the implemented ordinary formulation. Integer variables restricted to $\{0, 1\}$ count as binary. The MILP class includes 13 ordinary reference models with linearly representable general constraints. Explicit products of decision variables retain a quadratic classification, even when a linear reformulation is possible. Stochastic and robust models can belong to these same algebraic classes.

These classifications complement the 29 target-technique groups, which contain 5–46 tasks each. Appendix C.2 provides the group descriptions and corresponding accuracy results.

B.2 Evaluation Coverage and Reference Verification

We evaluate 11 general-purpose LLMs on all 561 tasks, retaining one outcome per model–task pair, including generation and execution failures. Each model's numerical accuracy uses all 561 tasks, yielding 6,171 retained outcomes in total. Cost comparisons require a correct numerical answer and the measurements needed for the relevant metric, as detailed in Appendix D.

Reference answers and cost coverage. All 561 tasks have verified reference objectives. For 543 tasks, the selected ordinary and expert executions reach optimality and agree on the objective under the cost-measurement protocol. The remaining 18 tasks are verified through supplementary reference executions, analytic or exact solutions, or optimality certificates on the same numerical inputs. These checks supply reference answers for accuracy; their timings are not added to the cost analysis, which retains the 543-task subset. Appendix E describes the validation criteria.

Reference model sizes. Figure 5(d,e) covers all 561 reference pairs. For each implementation, variable and total constraint counts are separate maxima over models submitted to the solver before presolve; the two maxima need not come from the same model or solver call. Repeated solves are not summed. Total constraints include native linear, quadratic, general, and SOS constraints, counting each once; variable bounds, integrality declarations, and solver-generated cuts are excluded. Counts are measured during execution or derived exactly from the implementation and fixed input where needed. Appendix C.4 instead relates runtime to the variable, linear-constraint, and nonzero counts recorded in the corresponding execution.

B.3 Code Generation Protocol

Each prompt provides the problem statement and input-field descriptions. It requests a function that constructs and returns a Gurobi model; the evaluation runner supplies the complete numerical instance at execution time and invokes the solver. Some prompts also request a symbolic formulation summary. Code-only prompts include a general efficiency instruction without specifying the target technique.

C Detailed Experimental Results

This appendix expands the main findings with accuracy breakdowns, cost distributions, and analyses of model size, preparation, generation, and memory use. It also reports fine-tuned-model failures (Section C.6) and identifies the slow tasks driving cumulative solver costs for all 11 general-purpose models (Section C.7). Measurement definitions and comparison rules are in Appendix D; technique-use assessments are in Appendix F.1.

C.1 Numerical Outcomes (RQ1)

All 11 general-purpose models are evaluated on the same 561 tasks using the verified reference objectives and the correctness criterion in Section 4.1. Table 3 separates correct answers, incorrect objectives returned with `OPTIMAL` status, and other generation or execution outcomes.

Solver status can give a different ranking from accuracy. Across 6,171 evaluations, 4,264 return correct objective values (69.1%). Of the 4,602 programs reporting `OPTIMAL`, 338 (7.3%) return incorrect values. For example, Claude Opus 4.6 produces more `OPTIMAL` results than GPT-5.5

Table 3: Numerical outcomes on all 561 tasks per model. Correct, Objective mismatch, and Other outcome sum to 561. The `OPTIMAL` column is the sum of Correct and Objective mismatch.

	Mutually exclusive outcomes			Solver status
Model	Correct	Objective mismatch	Other outcome	OPTIMAL
Gemini 3.1 Pro	512	10	39	522
GPT-5.5	447	23	91	470
Claude Opus 4.6	442	36	83	478
DeepSeek-V4 Flash	441	25	95	466
Kimi K2.6	433	23	105	456
Qwen 3.6 27B	394	27	140	421
GLM-5.1	390	40	131	430
Qwen 3.5 122B	369	28	164	397
MiniMax M2.5	315	33	213	348
Qwen 3 32B	313	56	192	369
Qwen 3.6 Plus	208	37	316	245
Total	4,264	338	1,569	4,602

(478 versus 470), but fewer correct answers (442 versus 447). Checking the objective therefore matters both for identifying errors and for comparing models.

Most unsuccessful evaluations end without an `OPTIMAL` result. The 1,569 records in Other outcome account for 82.3% of the 1,907 unsuccessful evaluations. They include generation failures, execution errors, and nonoptimal solver termination. Table 3 distinguishes these outcomes from objective errors after an optimal solve. Section C.2 examines how accuracy and objective errors vary across task groups.

Accuracy and cost comparisons use different task sets. Of the 4,264 correct evaluations, 4,202 belong to the 543-task reference cost subset; the other 62 contribute only to accuracy (Appendix B.2). All 4,202 have paired `Runtime`, `Work`, and size measurements, while 4,148 have complete `Build+opt` measurements. Table 1 uses the same 4,148 pairs for all four cost ratios; per-model counts appear in Table 8. Missing values are excluded. The model-size analysis additionally requires compatible counting scopes and positive measurements (Section C.4).

C.2 Accuracy Across Task Groups (RQ1)

Table 4 reports accuracy across all 11 models, grouped by target OptTips technique; Figure 8 shows each model separately. The 29 groups contain 5–46 tasks each and include the modeling variants listed in the table.

Accuracy varies substantially across task groups. Across the 11 models, accuracy ranges from 27.3% (15/55) on T17 (column generation) to 93.9% (248/264) on T15 (independent block decomposition). Each task contributes one outcome per model.

Objective errors are concentrated in a few groups. T08 (substitution and product linearization) and T17 contain 51 of the 561 tasks (9.1%) but account for 109 of the 338 programs that report `OPTIMAL` with incorrect objective values (32.2%). On T17's five tasks, 41 of the 55 model–task evaluations produce an `OPTIMAL` result, but only 15 of these match the verified reference objectives. Appendix F.1 reports automated technique-use assessments; Appendix F.3 summarizes selected manual checks.

C.3 Solver Costs Across Tasks and Models (RQ2)

We first describe solver costs on the tasks each model solves correctly, then compare models on shared tasks. All analyses use the 543-task reference cost subset defined in Section C.1; Appendix D.2 gives the comparison and aggregation rules. Table 5 summarizes measured times and solver work; Table 1 reports costs relative to the two references.

A small number of long runs raises average solver time. Median `Runtime` is 23.1–36.0 seconds across models, whereas mean runtime is 150.8–247.3 seconds (Table 5). The slowest $\sim$10% of correctly solved runs with valid timings account for 68.4–76.2% of total solver time (Figure 10(b)).

Table 4: Task groups and numerical accuracy. n is the number of tasks. Correct gives the number of correctly solved cases out of all $11n$ model–task pairs; the last column shows the corresponding percentage. Descriptions list the modeling variants represented in each group.

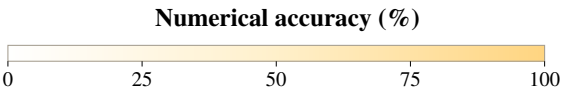
ID	Techniques and formulation variants	n	Correct	Accuracy (%)
T01	Equality-based elimination; balance reformulations	32	249/352	70.7
T02	Bounds; epigraphs; persistence variants	15	93/165	56.4
T03	Network-flow capacities as variable bounds	19	192/209	91.9
T04	Tight bounds; exact relaxations; network persistence	25	177/275	64.4
T05	Big-M formulations; persistent project activation	19	143/209	68.4
T06	Indicators, semicontinuous variables, and SOS	21	177/231	76.6
T07	Valid inequalities; arc-persistence variants	16	146/176	83.0
T08	Substitution and product linearization	46	246/506	48.6
T09	Piecewise-linear and native compact representations	8	56/88	63.6
T10	Convex hulls, envelopes, and convexification	30	236/330	71.5
T12	Duality; persistence and epigraph variants	6	52/66	78.8
T13	Lagrangian relaxation and dual formulations	15	145/165	87.9
T15	Independent block decomposition	24	248/264	93.9
T16	Benders decomposition	13	133/143	93.0
T17	Dantzig–Wolfe decomposition and column generation	5	15/55	27.3
T18	Total unimodularity; flow-based TSP variants	21	179/231	77.5
T19	Compact coverage without explicit assignments	14	135/154	87.7
T20	Compact temporal recurrences and direct linking	43	326/473	68.9
T21	Route enumeration; flow connectivity; temporal variants	21	101/231	43.7
T22	Robust counterparts for budgeted uncertainty	27	172/297	57.9
T24	Dominance pruning, fixing, and bound propagation	18	136/198	68.7
T25	Symmetry breaking through ordering and indexing	19	125/209	59.8
T30	Heuristic MIP starts	16	136/176	77.3
T34	Elimination of nonnegative slack variables	16	123/176	69.9
T35	Variable elimination and dimensionality reduction	13	93/143	65.0
T38	Convex conjugacy and Fenchel duality	7	51/77	66.2
T44	Transportation and network-simplex structure	14	103/154	66.9
T48	Cover, clique, subtour, and capacity cuts	15	87/165	52.7
T49	Linking constraints and activation strengthening	23	189/253	74.7
Total	29 groups; 11 models	561	4,264/6,171	69.1

Table 5: Recorded costs on correctly solved tasks within the reference cost subset. Each metric uses that model's available measurements; medians are calculated separately. The final column uses the same 47 tasks for all 11 models and the expert reference. Times are in seconds; `Work` is in solver work units.

	`Runtime` (s)			`Work`	`Build`	`Build+opt`	Shared `Runtime`
Model	Median	Mean	P90	Median	Median	Median	Mean ($n = 47$)
Gemini 3.1 Pro	32.51	197.66	414.53	22.66	0.63	37.80	77.04
GPT-5.5	28.84	208.50	671.57	21.57	0.97	37.77	197.44
Claude Opus 4.6	23.52	177.59	369.99	18.84	0.87	37.38	130.45
DeepSeek-V4 Flash	23.11	190.92	430.92	18.19	0.82	32.71	109.45
Kimi K2.6	29.70	210.28	599.96	20.36	0.76	36.10	166.83
Qwen 3.6 27B	32.23	215.00	453.29	20.77	1.65	49.24	144.98
GLM-5.1	27.59	150.78	352.58	18.87	1.01	33.77	171.77
Qwen 3.5 122B	26.00	200.10	408.90	20.36	0.84	32.58	182.40
MiniMax M2.5	30.60	232.36	474.85	23.98	0.91	38.86	181.27
Qwen 3 32B	35.95	247.32	698.13	22.53	1.25	53.98	125.19
Qwen 3.6 Plus	34.20	214.13	426.31	26.45	0.71	38.19	134.57
Expert reference	—	—	—	—	—	—	43.05

Figure 9 illustrates the lowest, middle, and highest shares using Kimi K2.6, Qwen 3.6 27B, and MiniMax M2.5. Section C.7 identifies the slow tasks for every model and compares their runtimes with the expert references.

Direct model comparisons use the same tasks on both sides. Each model's runtime distribution in Table 5 includes a different set of correctly solved tasks. Figure 10(a) compares each pair on the tasks both models solve correctly, yielding 126–420 tasks per pair. A smaller set of 47 tasks is solved correctly by all 11 models. On this fixed set, mean `Runtime` is 43.05 seconds for the expert reference and 77.04–197.44 seconds for the LLMs (Table 5). This comparison holds the task set fixed across every model, but covers only 8.4% of the benchmark.



Group (n)	Gemini 3.1 Pro	GPT 5.5	Claude Opus 4.6	DeepSeek V4 Flash	Kimi K2.6	Qwen 3.6 27B	GLM 5.1	Qwen 3.5 122B	MiniMax M2.5	Qwen 3 32B	Qwen 3.6 Plus
T01 ($n = 32$)	75%	91%	88%	78%	91%	78%	53%	75%	66%	66%	19%
T02 ($n = 15$)	60%	80%	53%	60%	67%	27%	67%	40%	53%	47%	67%
T03 ($n = 19$)	100%	100%	100%	100%	100%	89%	95%	89%	95%	74%	68%
T04 ($n = 25$)	92%	84%	60%	68%	72%	60%	60%	60%	56%	52%	44%
T05 ($n = 19$)	89%	95%	84%	79%	89%	89%	79%	42%	79%	21%	5%
T06 ($n = 21$)	100%	90%	100%	90%	90%	86%	57%	38%	90%	52%	48%
T07 ($n = 16$)	100%	100%	94%	88%	88%	100%	75%	88%	81%	81%	19%
T08 ($n = 46$)	91%	37%	54%	65%	57%	59%	41%	46%	57%	24%	4%
T09 ($n = 8$)	100%	75%	100%	75%	100%	62%	75%	25%	38%	38%	12%
T10 ($n = 30$)	100%	97%	50%	80%	97%	90%	50%	47%	60%	63%	53%
T12 ($n = 6$)	83%	100%	100%	100%	83%	67%	67%	50%	67%	83%	67%
T13 ($n = 15$)	100%	100%	100%	93%	93%	93%	80%	87%	93%	100%	27%
T15 ($n = 24$)	100%	100%	88%	100%	96%	100%	100%	96%	92%	92%	71%
T16 ($n = 13$)	100%	100%	100%	85%	100%	100%	85%	92%	85%	92%	85%
T17 ($n = 5$)	0%	40%	60%	0%	0%	0%	60%	60%	40%	40%	0%
T18 ($n = 21$)	95%	81%	90%	100%	90%	67%	81%	90%	90%	48%	19%
T19 ($n = 14$)	100%	93%	71%	100%	100%	79%	86%	93%	93%	79%	71%
T20 ($n = 43$)	95%	81%	81%	72%	65%	67%	84%	58%	56%	65%	33%
T21 ($n = 21$)	71%	48%	67%	71%	52%	19%	52%	29%	33%	10%	29%
T22 ($n = 27$)	85%	48%	78%	89%	93%	63%	78%	59%	19%	7%	19%
T24 ($n = 18$)	100%	78%	78%	94%	78%	67%	72%	67%	17%	72%	33%
T25 ($n = 19$)	84%	74%	95%	42%	26%	84%	84%	84%	21%	47%	16%
T30 ($n = 16$)	100%	100%	100%	81%	88%	62%	75%	81%	31%	75%	56%
T34 ($n = 16$)	88%	100%	62%	50%	38%	75%	44%	62%	75%	81%	94%
T35 ($n = 13$)	100%	92%	62%	85%	69%	85%	77%	69%	15%	31%	31%
T38 ($n = 7$)	100%	86%	43%	100%	71%	71%	86%	43%	29%	71%	29%
T44 ($n = 14$)	86%	64%	100%	79%	71%	57%	71%	79%	21%	50%	57%
T48 ($n = 15$)	100%	67%	73%	67%	60%	20%	53%	67%	7%	53%	13%
T49 ($n = 23$)	96%	70%	91%	78%	87%	70%	78%	100%	30%	74%	48%

Cells: percentage of the n tasks solved correctly by each model.

Figure 8: **Numerical accuracy by model and task group.** Each cell shows the percentage of tasks solved correctly, with all tasks in the group included in the denominator. Model order follows Table 1.

Table 6: **Solver costs by the number of models answering each task correctly.** The four groups contain 541 tasks in the reference cost subset, each solved correctly by at least one model. /Expert is the shifted geometric `Runtime` ratio (Appendix D.2). Slower (%) is the percentage of valid LLM–expert pairs with longer LLM runtime.

Models with correct answers	Tasks	`Runtime` pairs	/Expert ↓	Slower (%)
1–3	26	67	1.69	58.2
4–7	192	1,169	1.58	57.5
8–10	276	2,449	1.67	66.8
11 (all models)	47	517	2.82	82.2

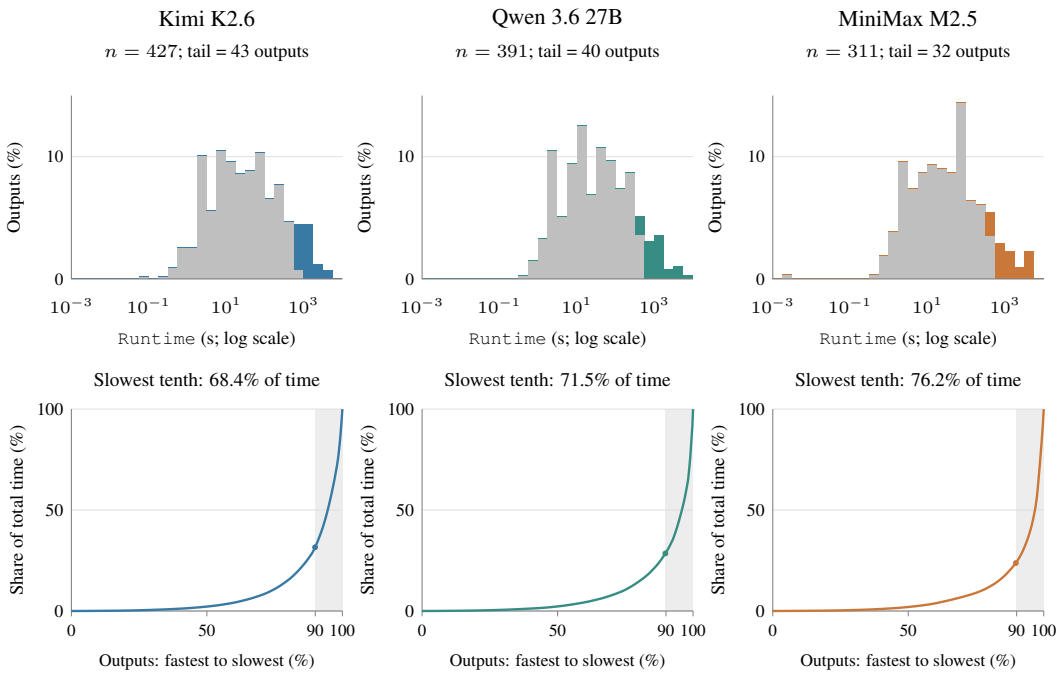


Figure 9: How long runs contribute to total solver time. The three models have the lowest, middle, and highest shares of time spent in their slowest tenth. Top: runtime histograms on a logarithmic time axis, with the slowest $\lceil n/10 \rceil$ runs highlighted. Bottom: cumulative share of solver time as runs are ordered from fastest to slowest; shading marks the same slowest runs. Each model uses its correctly solved tasks with valid timings.

The expert advantage also appears outside the 47 shared tasks. On the other 494 tasks solved correctly by at least one LLM, 2,346 of 3,685 valid LLM–expert pairs (63.7%) have longer LLM runtimes. Their pooled shifted geometric `Runtime` ratio is 1.64. Table 6 groups these tasks by how many models solve them correctly: the ratio remains above one in each group. Thus, the observed efficiency gap is not confined to the small set solved correctly by every model.

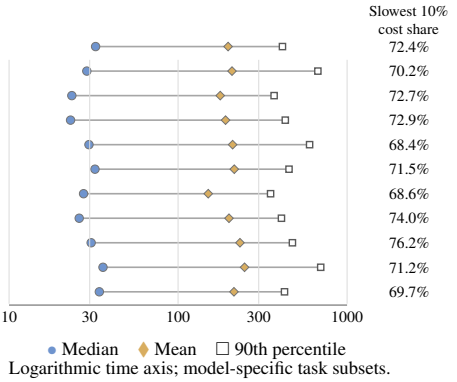
(a) How often is the row model faster?

	1	2	3	4	5	6	7	8	9	10	11
1. Gemini	–	53	49	50	51	65	58	56	58	61	54
2. GPT-5.5	47	–	42	47	49	53	55	53	52	58	47
3. Claude	51	58	–	53	58	65	62	60	57	59	54
4. DeepSeek	50	53	47	–	50	59	59	54	54	54	50
5. Kimi	49	51	42	50	–	60	51	54	54	53	51
6. Qwen 27B	35	47	35	41	40	–	45	41	40	47	37
7. GLM	42	45	38	41	49	55	–	52	48	56	53
8. Qwen 122B	44	47	40	46	46	59	48	–	50	58	42
9. MiniMax	42	48	43	45	46	60	52	50	–	56	45
10. Qwen 32B	39	42	41	46	47	53	44	42	44	–	40
11. Qwen Plus	46	53	46	50	49	63	47	58	55	60	–

Row faster (%): 0 – 50 – 100

Tasks solved correctly by both models ($n = 126$–420).

(b) Own-subset `Runtime` (s)



Logarithmic time axis; model-specific task subsets.

Figure 10: **Model comparisons on shared tasks and runtime distributions.** (a) Percentage of tasks solved correctly by both models with valid timings on which the row model is faster than the column model (10^{-6} s tie tolerance; 126–420 tasks per pair). Opposite cells need not sum to 100% because ties remain in the denominator. (b) Median, mean, and 90th-percentile `Runtime` on tasks solved correctly by each model, and the slowest tenth's share of measured `Runtime`. Whiskers summarize the distribution; they are not confidence intervals. Model order follows Table 1.

C.4 Model Size and Solver Time (RQ3)

We compare each generated model with the expert reference on the same task to test whether a smaller model also solves faster. Within the reference cost subset, we retain tasks whose archived ordinary and expert size summaries describe a single model. Requiring correct objective values and positive size counts and `Runtime` yields 3,788 LLM–expert pairs on 497 tasks. Dimensions and runtime come from the same recorded execution.

Two ways to identify smaller models. The first criterion compares the pre-presolve counts of variables, linear constraints, and nonzero coefficients in the linear constraint matrix: none of these three counts may increase, and at least one must decrease. The second requires fewer variables and fewer linear constraints, without restricting nonzeros. Both comparisons are relative to the expert. Table 7 reports the number of pairs satisfying each criterion and how often the LLM takes longer to solve the task.

Table 7: Model size and solver time relative to the expert reference. Eligible counts pairs before applying the size criteria; n counts pairs meeting each criterion. Slower counts pairs where LLM `Runtime` exceeds the expert's runtime by more than 10^{-6} seconds; percentages use n as the denominator. Rows are linear constraints; nonzeros are nonzero coefficients in their constraint matrix.

		No larger on three counts			Fewer variables and rows		
Model	Eligible	n	Slower	%	n	Slower	%
Gemini 3.1 Pro	465	100	55	55.0	45	24	53.3
GPT-5.5	401	45	26	57.8	19	11	57.9
Claude Opus 4.6	395	82	44	53.7	41	19	46.3
DeepSeek-V4 Flash	395	59	31	52.5	29	10	34.5
Kimi K2.6	390	70	40	57.1	28	13	46.4
Qwen 3.6 27B	352	80	56	70.0	31	23	74.2
GLM-5.1	342	71	36	50.7	27	12	44.4
Qwen 3.5 122B	329	74	52	70.3	28	21	75.0
MiniMax M2.5	277	56	31	55.4	29	14	48.3
Qwen 3 32B	270	66	52	78.8	31	25	80.6
Qwen 3.6 Plus	172	32	26	81.2	9	9	100.0
Total	3,788	735	449	61.1	317	181	57.1

Smaller reported models often solve more slowly. Under the first criterion, 449 of 735 pairs (61.1%) have longer LLM runtimes. The per-model proportion ranges from 50.7% to 81.2%, so the pattern appears in every model (Figure 7(a)). Under the second criterion, 181 of 317 pairs (57.1%) are slower. Reducing both variables and linear constraints therefore does not by itself establish a runtime improvement.

The differences are not limited to nearly tied runtimes. Of the 449 slower cases under the first criterion, 346 take at least twice as long as the expert, with at least one additional second of solver time. These cases span all 11 models. Including zero-valued counts and runtimes changes the slower proportions to 60.6% (449/741) and 56.0% (181/323) under the first and second criteria, respectively. The finding is therefore similar with or without zero-valued observations.

C.5 Costs Beyond Solver Time (RQ4)

We examine model preparation, code generation, and memory use. Preparation and execution comparisons use correctly solved tasks in the 543-task reference cost subset. Generation summaries include all available responses. Appendix D defines the measurements.

Preparing a model can take longer than solving it. Among 4,202 LLM runs with valid preparation and solver-call timings, `Build` exceeds solver-call time in 729 (17.3%). The proportion ranges from 11.5% to 22.5% across models (Table 8), even though the median preparation share of `Build+opt` is only 2.8%.

Preparation can reverse the comparison with the expert. On the 4,148 pairs with both LLM and expert timings, we first compare the externally measured solver-call times, then compare `Build+opt`. Adding preparation reverses which implementation is faster in 357 pairs (8.6%): 122 LLM programs change from faster to slower than the expert, and 235 change from slower to

faster. These reversals occur in all 11 models. Thus, a shorter solver call does not necessarily mean a faster implementation once preparation is included.

Table 8: Preparation time and comparisons with the expert. LLM runs have recorded preparation and solver-call times; LLM–expert pairs also have the corresponding expert measurements. `Build` longer counts runs where preparation exceeds solver-call time, with percentages over LLM runs. The last two columns count changes in the LLM's faster/slower relation to the expert when preparation is included.

Model	LLM runs	`Build` longer n (%)	LLM–expert pairs	Faster to slower	Slower to faster
Gemini 3.1 Pro	504	58 (11.5)	499	0	49
GPT-5.5	441	73 (16.6)	435	8	33
Claude Opus 4.6	435	76 (17.5)	430	21	29
DeepSeek-V4 Flash	433	88 (20.3)	428	14	18
Kimi K2.6	427	71 (16.6)	423	14	22
Qwen 3.6 27B	391	88 (22.5)	386	24	15
GLM-5.1	382	60 (15.7)	375	5	22
Qwen 3.5 122B	367	70 (19.1)	361	14	14
MiniMax M2.5	311	61 (19.6)	308	13	11
Qwen 3 32B	309	57 (18.4)	303	6	14
Qwen 3.6 Plus	202	27 (13.4)	200	3	8
Total	4,202	729 (17.3)	4,148	122	235

The reversals include substantial time differences. In 160 of the 357 reversals, the final `Build+opt` difference is at least 10 seconds. Our default comparison treats differences within 10^{-6} seconds as ties. Increasing this tolerance to the larger of 0.01 seconds and 1% of the longer compared time still gives 317 reversals (7.6%).

Table 9: Generation latency and token counts for all available responses, including incorrect programs. Values are medians. n_A, n_T, and n_R count records with latency, input/output-token, and reasoning-token measurements, respectively. Prompt and Completion denote input and output tokens. Token definitions vary by provider; reasoning tokens may already be included in Completion, and a reported zero does not establish that no internal reasoning occurred.

Model	n_A/n_T	`API latency` (s)	`Prompt`	`Completion`	`Reasoning`	n_R
Gemini 3.1 Pro	559/558	55.4	743.5	6,714.5	5,827	558
GPT-5.5	535/534	61.2	715.5	1,431	516	522
Claude Opus 4.6	561/561	17.2	937	748	0	561
DeepSeek-V4 Flash	561/561	81.9	714	11,327	10,562	561
Kimi K2.6	561/561	82.2	714	11,680	10,714	561
Qwen 3.6 27B	558/558	194.9	724	8,319	8,193	190
GLM-5.1	561/561	27.9	690	1,671	1,245	531
Qwen 3.5 122B	561/561	62.2	724	6,727	2,401	12
MiniMax M2.5	560/560	55.9	689	4,837.5	0	13
Qwen 3 32B	560/560	231.2	702	8,247.5	7,686.5	560
Qwen 3.6 Plus	561/561	14.0	726	1,135	–	0

Generating code often takes longer than running it. Table 9 summarizes the final recorded API call for each response, including those that do not lead to a correct solution. Median `API latency` ranges from 14.0 to 231.2 seconds across models. Among the 4,202 correct runs with both `API latency` and `Build+opt` recorded, generation takes longer in 2,336 (55.6%). The proportion reaches 76.2% for Qwen 3.6 27B and 74.1% for Qwen 3 32B, using each model's own correctly solved tasks in the cost subset. Reducing solver time alone therefore leaves a substantial part of the recorded waiting time unchanged. These final-call waiting times exclude earlier generation or repair calls; Appendix D.1 details their measurement scope.

More output tokens do not necessarily mean longer waits. Reported median prompt lengths vary from 689 to 937 tokens, while median completion lengths span 748–11,680 (Table 9). Kimi has a larger median completion count than Qwen 3 32B (11,680 versus 8,247.5), yet a much shorter median wait (82.2 versus 231.2 seconds). Output volume and elapsed waiting time thus capture different aspects of generation cost. These summaries do not isolate the causes: latency includes service and network waiting, and token definitions vary by provider. Reasoning-token coverage also differs sharply; Qwen 3.5 122B's median uses only 12 records, so that column does not support a uniform comparison of reasoning effort.

The upper tail matters more than small differences in median memory. Table 10 covers all 4,202 correct cost-analysis runs after unit harmonization. Per-model median `Gurobi peak` values cluster between 107.5 and 139.1 MiB, but each model's P90 is 12.6–21.4 times its own median. Across all runs, the median is 128.5 MiB and P90 is 2,206.7 MiB; 224 runs (5.3%) reach at least 4 GB. The common pattern is therefore a long upper tail, which a median-based memory budget would miss. These summaries use each model's own correctly solved tasks, so they do not isolate differences between models on identical workloads.

The process-memory subset has a different distribution. The 1,769 runs with all three `RSS` snapshots have a median solver allocation of 61.7 MiB, less than half the full-set median of 128.5 MiB. Their P90 remains similar, however: 2,238.2 versus 2,206.7 MiB. Thus, the matched subset retains demanding cases but does not represent the full distribution's center. We use the full set to describe solver-memory allocations and the matched subset to compare process and solver measurements on the same runs.

Process snapshots and solver peaks answer different questions. In the matched subset, median `RSS` snapshot maximum is 74.8 MiB and P90 is 1,328.6 MiB. Post-solve `RSS` is at least twice its pre-solve value in 412 runs (23.3%), showing that memory held at the end of solving can substantially exceed the pre-solve level. These boundary snapshots can miss temporary peaks during optimization; `Gurobi peak` instead tracks solver-environment allocations. Neither their difference nor the difference between their quantiles measures Python memory overhead (Appendix D.1).

Table 10: Solver and process memory in MiB. The full set contains all 4,202 correct runs used for cost analysis. The matched subset contains 1,769 runs with both solver memory and process `RSS` sampled before preparation, before solving, and after solving; `RSS` is the largest of these three snapshots. `Gurobi peak` measures peak allocation within the solver environment. P90 is the 90th percentile. Overall pools runs across models; each model uses its own correctly solved tasks.

	All correct cost-analysis runs			Matched subset with process `RSS`				
		`Gurobi peak`			`RSS` snapshot max		`Gurobi peak`	
Model	n	Median	P90	n	Median	P90	Median	P90
Gemini 3.1 Pro	504	124.7	2,397.9	210	79.2	2,011.6	72.0	2,403.3
GPT-5.5	441	136.3	2,198.6	186	70.8	2,069.4	58.7	2,440.1
Claude Opus 4.6	435	118.4	2,510.0	176	78.8	2,082.5	57.8	2,507.5
DeepSeek-V4 Flash	433	130.4	1,938.5	184	75.9	1,234.1	69.6	1,660.2
Kimi K2.6	427	117.2	2,085.0	183	73.0	1,204.1	57.6	1,723.6
Qwen 3.6 27B	391	110.7	1,389.6	166	74.0	890.9	41.8	1,262.2
GLM-5.1	382	136.1	2,402.8	157	80.0	1,937.9	67.1	2,475.4
Qwen 3.5 122B	367	132.6	1,727.3	134	73.7	1,147.5	56.1	1,918.8
MiniMax M2.5	311	137.8	1,807.7	160	77.6	1,225.3	53.9	1,874.7
Qwen 3 32B	309	139.1	2,331.5	124	84.9	1,343.5	60.4	2,502.1
Qwen 3.6 Plus	202	107.5	2,305.8	89	73.4	1,051.0	53.0	2,037.1
Overall	4,202	128.5	2,206.7	1,769	74.8	1,328.6	61.7	2,238.2

C.6 Results for Fine-Tuned Models

We evaluate five fine-tuned models on all 561 EfficientOpt tasks: LLMOPT (14B) (Jiang et al., 2025), OptMATH (7B and 32B) (Lu et al., 2025), and SIRL (7B and 32B) (Chen et al., 2025b). This section examines their accuracy and where their programs fail.

Evaluation setup. We execute the saved programs on the fixed task instances, using the correctness criterion and verified reference objectives from Section 4.1. Programs call Gurobi directly or through Pyomo's `gurobi_direct` interface. Each run uses one solver thread, `Seed=0`, `MIPGap=0`, a 2 GiB process-memory limit, and Gurobi `SoftMemLimit=1.5` (GB). Up to ten programs run concurrently. These runs contribute only to the accuracy and failure analysis; their timings are excluded from efficiency comparisons.

Correct solutions remain limited. OptMATH (32B) and SIRL (32B) achieve the highest accuracy in this group, at 24.42% and 21.03%, respectively; the smaller models range from 0 to 6.42%

Table 11: Accuracy and execution outcomes for five fine-tuned models, with 561 tasks per model. Correct counts programs that finish with `OPTIMAL` status and the verified objective value. It is a subset of `OPTIMAL`, not an additional outcome category. The six columns from `OPTIMAL` through Other sum to 561 in each row.

Model	Correct n (%)	`OPTIMAL`	Static failure	Code error	Time limit	Memory limit	Other
OptMATH (32B)	137 (24.42)	189	11	212	51	57	41
SIRL (32B)	118 (21.03)	188	34	164	77	72	26
LLMOPT (14B)	36 (6.42)	77	20	377	28	35	24
OptMATH (7B)	1 (0.18)	4	401	146	3	6	1
SIRL (7B)	0 (0.00)	0	504	57	0	0	0

(Table 11). These small sets of correct solutions limit efficiency comparisons, motivating the separate accuracy and failure analysis.

Optimal solver status does not ensure a correct answer. Across the five models, 458 programs finish with `OPTIMAL` status, but 166 (36.2%) return an objective that differs from the verified answer. As in Section C.1, a solver can optimize the generated model successfully even when that model gives the wrong answer to the task.

Failure patterns differ across models. OptMATH (7B) and SIRL (7B) have 401 and 504 static failures, respectively: these programs cannot be compiled or lack the required modeling function. LLMOPT instead has 377 code errors, its most common outcome. Code errors include invalid data access and incorrect use of the modeling or solver API. For the two 32B models, time and memory limits also account for many outcomes: 108 runs for OptMATH and 149 for SIRL. These categories distinguish programs that cannot execute, programs that return incorrect objectives, and runs stopped by resource limits.

Time and memory limits include termination by either the execution process or Gurobi. Other covers the remaining solver and execution failures, including policy and logging interruptions.

C.7 Slow Runs and Total Solver Time (RQ2)

We examine which runs account for most solver time and how the expert implementations perform on those same tasks. As in Appendix C.3, we use 4,202 runs that pass the numerical checks and have valid `Runtime` measurements within the 543-task reference cost subset. For each LLM, we rank its n eligible runs by `Runtime` and select the slowest $k = \lceil n/10 \rceil$, approximately 10%. Table 12 summarizes their contribution to the model's total. Here, total time is the sum of recorded solver runtimes. It does not represent total program time or the elapsed time of a parallel batch.

Table 12: Solver-time concentration for each LLM. Time share is the selected k runs' percentage of total `Runtime`. Slower gives the number exceeding expert time by more than 10^{-6} seconds, out of k. LLM/expert divides summed LLM time by summed expert time on these same k tasks; it is not the shifted geometric ratio in Table 1.

	All eligible runs			Slowest ∼10%			
Model	n	Median (s)	Total (h)	k	Time share (%)	Slower	LLM/expert
Gemini 3.1 Pro	504	32.5	27.67	51	72.4	48/51	5.43
GPT-5.5	441	28.8	25.54	45	70.2	43/45	7.71
Claude Opus 4.6	435	23.5	21.46	44	72.7	42/44	5.31
DeepSeek-V4 Flash	433	23.1	22.96	44	72.9	43/44	5.80
Kimi K2.6	427	29.7	24.94	43	68.4	42/43	4.97
Qwen 3.6 27B	391	32.2	23.35	40	71.5	37/40	6.23
GLM-5.1	382	27.6	16.00	39	68.6	37/39	3.45
Qwen 3.5 122B	367	26.0	20.40	37	74.0	33/37	6.96
MiniMax M2.5	311	30.6	20.07	32	76.2	31/32	5.78
Qwen 3 32B	309	36.0	21.23	31	71.2	31/31	5.96
Qwen 3.6 Plus	202	34.2	12.02	21	69.7	20/21	4.84

A few runs account for most solver time. Across models, the slowest ∼10% of runs consume 68.4–76.2% of total solver time. Median runtimes are only 23.1–36.0 seconds, yet totals reach 12.0–27.7 hours, with just 11–24 runs per model accounting for at least half. These totals describe different task sets; Appendix C.3 compares models on shared tasks. Together, the selected sets contain 427 model–task runs covering 101 distinct tasks. T10 (convex hulls, envelopes, and convexification) contributes the largest share of selected-run time for every model: 24.6–50.5%.

Expert implementations are often faster on these slow tasks. On each LLM's selected tasks, its summed `Runtime` is 3.45–7.71 times the expert total. Of the 427 selected runs, 407 are slower than the expert; 359 take at least twice as long, with a gap of at least one second. Table 13 shows the slowest run for each LLM. Qwen 3.6 27B takes 7,065.35 seconds on `T07_003`, versus 15.21 seconds for the expert. Other tasks are slow for both: Kimi K2.6 takes 4,324.49 seconds on `T06_001`, versus 2,905.46 seconds for the expert. These comparisons describe tasks selected for high LLM runtime; they neither estimate the overall LLM–expert gap nor isolate the effect of a particular technique.

Table 13: The slowest eligible run for each LLM and the expert's runtime on the same task. Times are Gurobi `Runtime`. Share is the selected run's percentage of that LLM's total eligible solver time. Runs are selected by LLM time, irrespective of the expert time.

Model	Task	LLM (s)	Expert (s)	Share (%)
Gemini 3.1 Pro	`T24_002`	4,677.80	303.66	4.7
GPT-5.5	`T25_019`	4,071.28	16.38	4.4
Claude Opus 4.6	`T48_009`	4,718.84	22.63	6.1
DeepSeek-V4 Flash	`T15_012`	5,431.07	18.95	6.6
Kimi K2.6	`T06_001`	4,324.49	2,905.46	4.8
Qwen 3.6 27B	`T07_003`	7,065.35	15.21	8.4
GLM-5.1	`T05_020`	4,736.70	285.27	8.2
Qwen 3.5 122B	`T07_003`	6,501.74	15.21	8.9
MiniMax M2.5	`T21_022`	5,617.01	71.17	7.8
Qwen 3 32B	`T48_004`	5,077.62	164.01	6.6
Qwen 3.6 Plus	`T10_010`	3,079.45	236.71	7.1

The slowest runs dominate total solver time without affecting the median. For each model, halving the recorded runtimes of its slowest 10% of runs would reduce total solver time by 34.2–38.1%, while leaving the median unchanged. Halving the runtimes of all other runs instead would reduce total solver time by only 11.9–15.8%. Thus, the median alone obscures how a small number of slow runs drive total solver time.

D Evaluation Metrics and Comparison Protocol

This appendix defines the measurement and comparison rules used in Section 4 and Appendix C. Appendix D.1 explains what each metric captures; Appendix D.2 specifies eligible comparisons, aggregation, and the limits of their interpretation.

D.1 Metrics and Measurement Scope

We measure numerical accuracy, technique use, and computational cost separately. Table 14 summarizes the metrics; the following paragraphs explain their measurement boundaries.

Preparation and solving. `Build` measures preparation, including preprocessing, enumeration, and any solves or between-solve computation within the recorded intervals. `Build+opt` adds the elapsed time around separately timed solver calls, without adding Gurobi's `Runtime` again. This subtotal can omit input loading, imports, result extraction, and other work outside the timed intervals. Appendix I.2 distinguishes it from complete execution and request time.

Solver time and effort. Gurobi's `Runtime` and `Work` attributes describe the most recent optimization call (Gurobi Optimization, 2026b). We report the returned model's solve for single-model runs and sum the tracked calls for multi-solve procedures. Untracked solves inside model-building code may be absent from these totals even when their time is included in `Build`. `Work` measures solver effort in arbitrary units, not seconds or Python operations; it depends on the model, hardware, and solver settings.

Table 14: Evaluation metrics and what they measure.

Quantity	Recorded measurement	What it captures
Numerical check	`OPTIMAL` status and a finite objective matching the verified reference value	Numerical agreement on the tested instance; full task semantics and required decisions need separate verification.
Technique use	Automated labels from code and task context	Identifies target, alternative, none, or N/A; does not establish correctness or efficiency (Appendix F.1).
Solver cost	Gurobi `Runtime` (s), `Work` (units)	Time and solver effort for the recorded optimization calls.
`Build+opt`	Preparation time + separately timed solver calls (s)	An execution subtotal; excludes work outside the recorded intervals.
Generation	`API latency` (s); provider-reported `Tokens`	Waiting time for the final recorded call; token usage reported for its response. Earlier calls are excluded.
Model size	Variables, constraints, linear-matrix nonzeros	Dimensions before presolve. Figure 5 and Appendix C.4 use different counting scopes, explained below.
Memory	Process `RSS` snapshots; `Gurobi peak`	Sampled process-resident memory and peak solver-environment allocation, respectively.

Code generation. `API latency` measures the final recorded client call, including service and network waiting and any retries and backoff within that call. It excludes earlier generation or repair calls. Token counts describe the returned response under each provider's conventions, rather than cumulative usage over retries. Reasoning tokens may already be included in output tokens, so we do not add those categories together. Unavailable counts remain missing.

Model size. Figure 5(d,e) reports separate maxima of variable and total constraint counts across each reference's submitted models (Appendix B.2); the maxima need not come from the same model. Total constraints include linear, quadratic, general, and SOS constraints. The size–runtime analysis instead uses the variables, linear constraints, and linear-matrix nonzeros recorded with the timed solve (Appendix C.4). All counts are before presolve.

Memory. In the matched memory analysis, we report the maximum of three process `RSS` snapshots: before preparation, before the outer solver call, and after that call. These samples can miss intervening peaks. `Gurobi peak` is the maximum allocation within the solver environment, which may be shared by several models (Gurobi Optimization, 2026b); it is neither whole-process memory nor a sum of per-model peaks. Both are reported in MiB, converting native Gurobi GB values by $10^9/2^{20}$. Appendix C.5 gives the distributions and measurement coverage.

D.2 Task Selection and Result Aggregation

Accuracy and cost coverage. Numerical accuracy uses all 561 tasks. Paired execution-cost comparisons use the 543 tasks whose recorded ordinary and expert runs reach agreeing optimal objectives (Appendix B.2).

Tasks included in each comparison. Table 1 requires numerically correct outputs and complete measurements for all four cost ratios. The ratios within each row use the same tasks, yielding 4,148 model–task pairs in total; task sets can differ between LLMs. Single-metric analyses require only the relevant measurements within the reference cost subset, so the solver-runtime analysis includes 4,202 correct runs.

Direct LLM comparisons use tasks solved correctly by both models, or by all 11 for the all-model comparison, with valid measurements of the relevant metric (Appendix C.3).

Missing measurements. Each comparison excludes missing or invalid measurements; these are never replaced with zero. We do not substitute time limits for incomplete or timed-out runs.

Cost and size ratios. Table 1 uses shifted geometric ratios. For a cost or size metric X, let X_{im} and X_{ir} denote the measurements for LLM m and reference r on task i. The shifted geometric ratio over the eligible paired task set S is

$$R^X_{m,r}(S) = \exp\left[\frac{1}{|S|}\sum_{i\in S}\log\frac{X_{im}+s_X}{X_{ir}+s_X}\right]. \tag{1}$$

This shifted geometric ratio gives each task equal weight in log space. The shift s_X is one second for time, one solver work unit for `Work`, and one for counts. It accommodates zeros and limits the influence of tiny denominators; any unit conversion must rescale the shift as well as the measurements. A ratio below one indicates lower LLM cost or size under this aggregation. It is neither a ratio of summed costs nor an unshifted multiplicative speedup. Size ratios describe model structure and do not establish faster solving.

Runtime distributions and faster/slower counts. Faster/slower counts compare unshifted measurements. Unless stated otherwise in a sensitivity analysis, differences of at most 10^{-6} in the metric's unit are ties; ties remain in the denominator of win fractions. Appendix C.7 defines the slowest-tenth selection and summed-time comparisons.

Execution conditions. All paired timing measurements were obtained on the same machine for the same task and numerical instance.

Interpreting cost differences. Equal model-size counts do not establish formulation identity. Even algebraically equivalent formulations can differ in input order, starting solutions, or solver parameters that affect performance (Lodi & Tramontani, 2013; Gurobi Optimization, 2026e). Appendix G.1 examines formulation differences; Appendix G.2 shows how preparation affects costs.

E Dataset Quality Control and Expert Verification

This appendix describes the acceptance protocol in Figure 11: independent expert review, mathematical and execution checks, and re-verification after repairs. Acceptance requires valid tasks and references and justified technique use; observed speedup is not a criterion.

E.1 Independent Review and Information Access

OptDachshund produces candidates; independent experts determine acceptance. The protocol draws from five experts in mathematical optimization, modeling, and solver implementation, assigning at least two to each candidate and rotating assignments across techniques and source groups. One reviewer initially sees only the public statement and schema, checking whether the task is understandable without its intended solution. The other sees both reference models and implementations, the target OptTips card, and validation evidence. They submit their assessments independently before comparison. The review panel comprised five researchers, including doctoral students, postdoctoral researchers, and faculty members. Their collective expertise covered mathematical optimization, solver implementation, and LLM-based optimization modeling.

Evaluated LLMs receive the public statement and schema; their code accesses the numerical instance at execution. Reference solutions, objective values, technique labels, and expert hints remain hidden.

E.2 Acceptance Criteria

Task and data validity (Stages I–II). The public specification must define the decisions, objective, domains, constraints, and required outputs without relying on a hidden reference; material ambiguity requires revision or rejection. Automated checks and expert review verify schema–instance consistency, including dimensions, indices, units, bounds, and intended feasibility. The task may expose structure but must not prescribe the target technique. For example, stating that storage cartridges are identical describes the task; prescribing an order for their use variables reveals a symmetry-breaking formulation. Reviewers assess structural difficulty, including trivial reductions and whether recognizing them is the intended skill.

Valid and comparable references (Stage III). Both references must solve the same task on identical data and satisfy its output requirements. The ordinary reference must be a plausible conventional

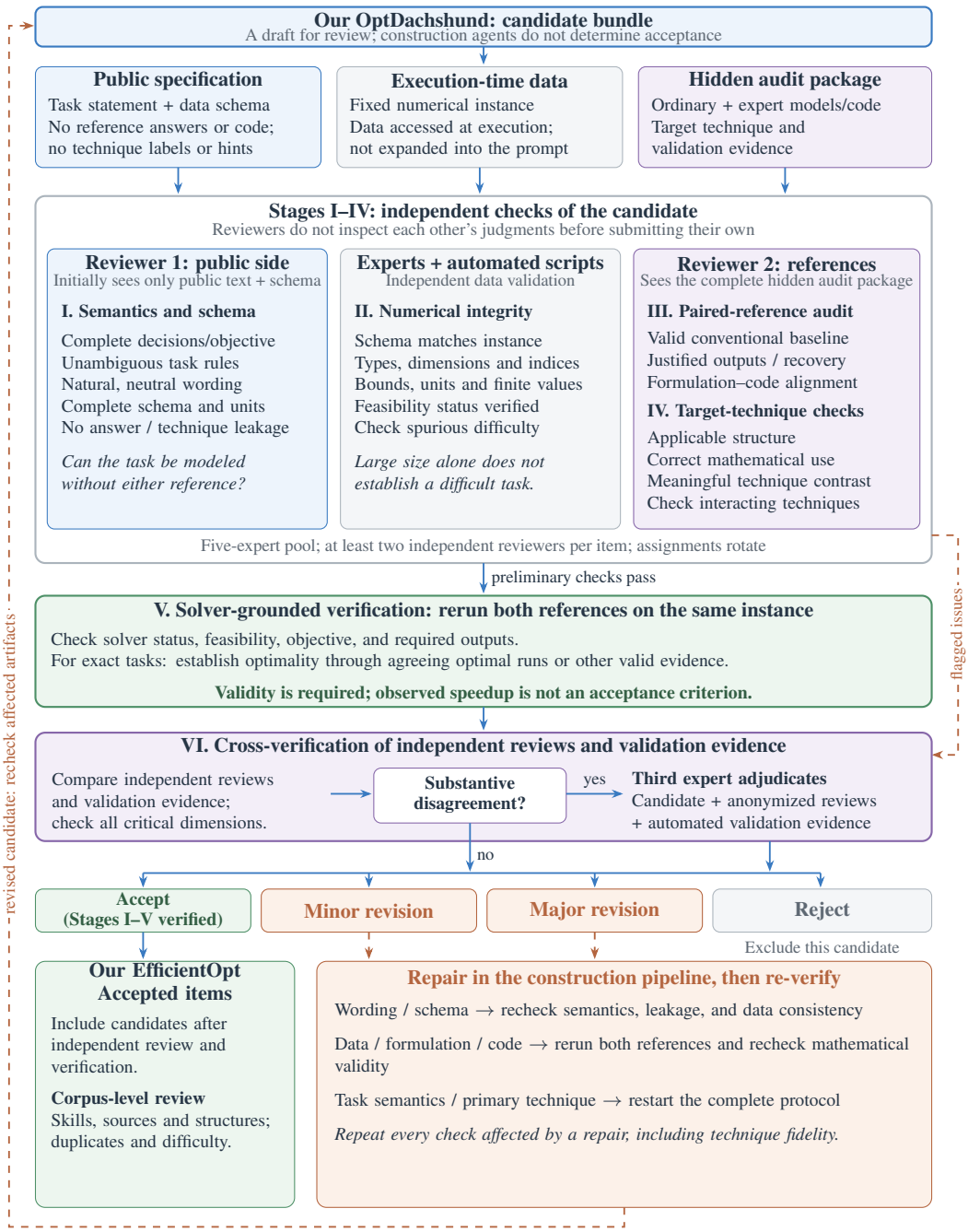


Figure 11: **Dataset quality control and expert verification.** Checks cover required decision recovery and valid termination, not objective agreement alone. Repairs trigger all affected checks. Appendix B.2 summarizes reference-answer verification and cost coverage.

formulation, without artificial delays, invalid bounds, or numerical pathologies introduced to favor the expert. Experts check formulation–code consistency and the mathematical justification for each answer. Different feasible sets are permitted when the transformation is valid, including recovery of requested decisions and justified termination for decompositions (Appendix I.1).

Applicable and faithful technique use (Stage IV). The expert reference must apply the OptTips mechanism to a structure that supports it. A tighter Big-M constant, for example, must follow from valid bounds; a valid inequality must preserve feasible integer solutions. Reviewers check for a

meaningful modeling contrast on the same task, data, and objective, accounting for techniques that interact with the primary target.

Solver-grounded verification (Stage V). The protocol requires executing both references on the same instance to check status, feasibility, objectives, and requested decisions. For exact tasks, optimality can be established by agreeing optimal runs, analytic or exact solutions, primal–dual certificates, or a checked feasible solution with a matching bound from an equivalent formulation. These alternatives allow answer verification when an ordinary run exceeds its time or memory budget. Multi-solve procedures require justified termination, such as a final pricing check for column generation. Appendix B.2 summarizes reference-answer verification and cost coverage.

E.3 Decisions, Repair, and Corpus Review

Reviewers compare their independent findings with validation evidence (Stage VI). A third expert adjudicates substantive disagreements using the candidate, anonymized reviews, and validation results. The outcomes are acceptance, minor revision, major revision, or rejection. Minor revisions preserve task semantics; changes to data, formulations, code, or the target technique require major revision. Acceptance requires resolving critical findings and completing the relevant checks.

Repairs trigger renewed checks of affected content: wording and schema changes require semantic, leakage, and data-consistency review; data or implementation changes require rerunning both references and rechecking mathematical claims. Changes to task semantics or the primary technique restart the full protocol. Review decisions and supporting evidence remain linked to each task.

Corpus review checks duplicates and examines the distributions of techniques, source problems, mathematical structures, and task difficulty.

F Assessing Modeling Technique Use

This appendix separates technique recognition from numerical correctness and computational cost. Appendix F.1 defines the automated labels and reports results by LLM and OptTips group; Appendix F.2 combines these labels with execution and timing checks; Appendix F.3 presents manual inspections of selected formulations and code.

F.1 Automated Assessment of Technique Use

Assessment protocol. We report technique-use assessments for all 6,171 model–task pairs: 561 tasks for each of the 11 LLMs. A frontier LLM serves as judge. The recorded protocol supplies the problem statement, OptTips knowledge, candidate code, both reference implementations, and execution measurements.

Label definitions. Each record receives one of four labels. *Target* indicates the designated technique or an implementation judged equivalent. *Alternative* indicates another modeling technique; *none* means neither is identified. *N/A* covers cases judged inapplicable or unassessable. Labels include failed executions and incorrect objectives, so a target label does not certify a valid implementation or lower computational cost.

Results across models and task groups. Target, alternative, none, and N/A account for 46.3%, 24.6%, 21.3%, and 7.8% of all records, respectively (Figure 7(b)). Target-label rates range from 33.9% to 55.4% across models, each using all 561 tasks as the denominator (Table 15). Table 16 gives counts for the 29 OptTips groups; Table 4 describes their task variants.

Interpreting the labels. Target labels can cover different implementations. In T20_011, both Kimi's cumulative auxiliary variables and Qwen 3.5 122B's direct coverage constraints receive target labels (Appendix F.3).

F.2 Combining Technique Labels with Execution Outcomes

The C1–C5 columns in Table 15 combine technique labels with execution outcomes and solver times using the rules below. They are distinct from the judge's raw labels.

Table 15: Technique-use assessments by LLM. Each model row classifies the same 561 records in two ways: technique labels (left) and combined diagnostic categories (right; Appendix F.2). The two column blocks each sum to 561 and must not be added together.

	LLM technique label				Combined diagnostic category				
Model	Target	Alt.	None	N/A	C1	C2	C3	C4	C5
Claude Opus 4.6	311	131	98	21	253	52	66	68	122
Gemini 3.1 Pro	297	173	79	12	272	67	99	73	50
DeepSeek-V4 Flash	296	139	100	26	247	62	62	71	119
Qwen 3.6 27B	286	132	128	15	196	44	75	79	167
GPT-5.5	284	147	106	24	223	68	71	82	117
Kimi K2.6	282	138	85	56	233	59	74	65	130
GLM-5.1	275	123	142	21	198	44	60	88	171
Qwen 3 32B	231	129	173	28	132	45	54	80	250
MiniMax M2.5	204	138	109	110	147	40	78	50	246
Qwen 3.5 122B	202	133	137	89	164	48	78	78	193
Qwen 3.6 Plus	190	135	158	78	67	45	50	42	357
Total	2,858	1,518	1,315	480	2,132	574	767	776	1,922

Table 16: Automated technique labels by OptTips group. n includes all model–task records from 11 LLMs, including failed executions. Target gives count (% of n); other columns give counts.

ID	Target OptTips group	n	Target (%)	Alt.	None	N/A
T01	Equality/inequality transformations	352	283 (80.4)	10	39	20
T02	Auxiliary variables and epigraphs	165	156 (94.5)	0	1	8
T03	Bound tightening and scaling	209	12 (5.7)	115	79	3
T04	LP relaxation	275	23 (8.4)	94	147	11
T05	Big-M modeling	209	171 (81.8)	7	10	21
T06	Indicators, semicontinuous variables, SOS	231	58 (25.1)	149	15	9
T07	Valid inequalities and cuts	176	134 (76.1)	17	24	1
T08	Substitution and product linearization	506	382 (75.5)	54	43	27
T09	Piecewise-linear modeling	88	78 (88.6)	0	3	7
T10	Convex hulls and convexification	330	137 (41.5)	109	63	21
T12	Dual transformation	66	1 (1.5)	64	0	1
T13	Lagrangian relaxation	165	0 (0.0)	46	116	3
T15	Block decomposition	264	0 (0.0)	151	112	1
T16	Benders decomposition	143	0 (0.0)	75	68	0
T17	Dantzig–Wolfe / column generation	55	0 (0.0)	25	28	2
T18	Network flow and matching	231	80 (34.6)	72	70	9
T19	Covering, packing, and knapsack	154	85 (55.2)	51	15	3
T20	Scheduling and temporal formulations	473	393 (83.1)	22	19	39
T21	Routing and path formulations	231	195 (84.4)	0	8	28
T22	Robust counterparts	297	174 (58.6)	29	78	16
T24	Preprocessing and dominance	198	4 (2.0)	69	109	16
T25	Symmetry breaking	209	114 (54.5)	29	26	40
T30	Warm starts	176	2 (1.1)	139	14	21
T34	Slack and surplus variables	176	135 (76.7)	0	16	25
T35	Variable elimination	143	14 (9.8)	11	89	29
T38	Convex conjugacy and Fenchel duality	77	49 (63.6)	15	0	13
T44	Transportation and network simplex	154	34 (22.1)	38	54	28
T48	Subtour, cover, clique, and capacity cuts	165	84 (50.9)	27	9	45
T49	Linking and activation strengthening	253	60 (23.7)	100	60	33
Total		6,171	2,858 (46.3)	1,518	1,315	480

Assignment rules. A record must report `OPTIMAL` or `SOLVED` and pass the archived objective check to enter C1–C4. All other records, including errors and timeouts, enter **C5**.

- **C1:** a target label, with no requirement for a runtime advantage.
- **C2:** an alternative label meeting the timing criterion below.
- **C3:** an alternative label not meeting that criterion.
- **C4:** a none or N/A label.

Timing criterion for C2. Let $r = t_{\mathrm{LLM}}/t_{\mathrm{expert}}$ and $s = t_{\mathrm{LLM}}/t_{\mathrm{ordinary}}$ be unshifted solver `Runtime` ratios. An alternative enters C2 if $r \leq 1.5$ or $(r < 2$ and $s < 1)$. The first condition requires LLM and expert timings; the second also requires the ordinary-reference timing. Meeting this solver-time threshold does not guarantee a speedup over either reference and says nothing about generation or construction cost.

Relation to the main results. These categories retain the archived runs and objective checks, yielding 4,249 records in C1–C4. The main evaluation uses its own selection of runs and verified reference objectives, yielding 4,264 numerically correct records. The two sets are not interchangeable: these diagnostics do not replace the accuracy results in Appendix C.1 or the cost comparisons defined in Appendix D.2.

F.3 Manual Inspection of Selected Modeling Choices

We inspected variables, constraints, and objectives in selected formulations and executable code to identify the implemented modeling choices. Solver status, model size, and runtime alone cannot identify a technique; valid alternatives may differ from the expert reference. Table 17 summarizes three examples and points to their detailed analyses.

Table 17: Selected modeling choices (RQ3). The cases are illustrative, not a random sample.

Task and output	Implemented choice	What the evidence supports
T10_011, GPT-5.5	Binary pairwise products with separate product inequalities; the expert uses continuous interactions with row–column marginals.	Both formulations enforce the same products at integer assignments, but their LP relaxations differ. The GPT model does not use the expert's marginal formulation.
T13_006, Gemini	Primal resource-allocation LP with explicit implied upper bounds; the expert uses a dual LP.	The approaches attain the same objective with similar recorded `Build+opt` times.
T20_011, Kimi / Qwen 3.5 122B	Kimi uses cumulative auxiliary variables; Qwen uses direct coverage constraints.	Eliminating Kimi's auxiliaries preserves feasible integer shift-start decisions and the objective. Qwen's longer construction time largely offsets its solver-time advantage.

G Case Studies on Modeling Choices and Efficiency

These cases examine what alternative formulations preserve and how construction affects their recorded costs. See Appendix D.1 for timing scopes and Appendix I.1 for required outputs.

G.1 What the Compared Formulations Preserve

Technology mixing (T13_006). The primal LP uses nonnegative technology shares summing to one per block, giving a bounded feasible region. On the evaluated instance, choosing each block's least-resource technology satisfies the resource cap, establishing feasibility. LP strong duality therefore gives equal optimal values for the primal and expert dual. The original technology shares correspond to the dual multipliers of the expert LP's mode constraints $u_i + \lambda r_{ik} \geq v_{ik}$, providing a direct recovery map.

Department relocation (T10_011). For departments p, q assigned to cities i^*, j^*, the expert's nonnegative interaction variables have row and column sums fixed by the assignments. This forces $y^{pq}_{i^*j^*} = 1$ and all other entries to zero. Both formulations therefore enforce $y^{pq}_{ij} = x_{pi}x_{qj}$ at integer

assignments and preserve the common objective. Their LP relaxations differ: for a single department pair with fractional assignments $(1/2, 1/2)$ over two cities, the product inequalities allow $y = 0$, but the marginal equalities do not. The product links already make interactions integral under binary assignments. The expert also changes the linking constraints, so the comparison does not isolate the effect of removing binary declarations.

G.2 How Construction Changes the Cost Comparison

Here, `Runtime` is solver time; `Build+opt` includes construction and externally timed optimization (Appendix D.1). Appendix C.5 gives the corresponding benchmark-wide analysis.

Shift scheduling (T20_011). Eliminating Kimi's cumulative auxiliary variables preserves the feasible integer shift-start decisions and objective of Qwen 3.5 122B's direct coverage model. Their solver times are 566.443 and 2.071 seconds, respectively, while their recorded `Build` times are 6.927 and 547.507 seconds. Their recorded `Build+opt` times are 573.362 and 549.577 seconds. The Kimi/Qwen ratio therefore shrinks from $273.512\times$ for solver time to $1.043\times$ for `Build+opt`: Qwen remains faster, but construction offsets most of its solver-time advantage.

Transit duty scheduling (T20_013). The ordinary formulation introduces one continuous active-duty variable and one equality link for each feasible start–period pair. The expert formulation instead uses the integer start variables directly in the period-coverage constraints. The two formulations encode the same feasible start decisions and objective, and both attain an objective value of 68,835,006. The ordinary model has 1,620,420 variables and 1,620,420 constraints, compared with 180,000 of each in the expert model; their constraint matrices have 4,321,260 and 1,440,420 nonzeros, respectively. Their Gurobi `Runtime` values are 5.420 and 2.989 seconds, while their recorded `Build` times are 26.382 and 3.345 seconds. Their recorded `Build+opt` times are 31.803 and 6.334 seconds. The ordinary/expert ratio thus increases from 1.813 for `Runtime` to 5.021 for `Build+opt`. In this pair, the expert formulation is faster to build as well as to solve, so including Build widens its measured advantage.

H OptDachshund: Agent Roles and Information Flow

OptDachshund uses six agents to redesign source problems around OptTips techniques and construct paired references. Their roles follow the three stages in Section 3.2 and Figure 4.

Technique Matcher. The matcher selects OptTips techniques whose applicability conditions fit the source problem and proposes a redesign that exposes the relevant modeling challenge. Expected difficulty and efficiency gains guide this design; they remain hypotheses to test.

Reformulation Designer. Using the source problem and matching plan, the designer creates a task that requires a substantive modeling choice beyond rewording or numerical substitution. It writes the public statement and data schema, together with an internal formulation and intended solution.

Quality Auditor. The auditor checks mathematical validity and consistency across the candidate's statement, schema, formulation, and intended solution. It also checks applicability against the selected OptTips card, source traceability, technique leakage, duplicate risk, and diversity. A candidate proceeds to reference implementation after passing this design audit.

Repair Agent. This agent addresses the auditor's findings and returns the candidate for re-audit. Local repairs preserve the source problem and primary technique; changes to task semantics or the technique trigger the full review in Appendix E.

Reference Implementation Agent. This agent builds ordinary and expert implementations for the same final task, fixed instance, and required outputs. The expert applies the selected technique. Reference implementations support both model-building and algorithmic interfaces, with cost measurements defined in Appendix D.1.

Reference Audit Agent. This agent checks both references against the final task and selected OptTips card, using their formulations, code, and execution evidence. It examines executability, required outputs, task consistency, and valid technique use. Failures require revision and re-audit.

Target techniques, references, and review records remain hidden from evaluated LLMs, which receive the public task and data through the interface in Section 3.3. Final acceptance requires independent expert review and solver-grounded verification, repeated after relevant repairs (Appendix E). Agent approval alone is insufficient, and observed speedup is not required.

EXAMPLE AGENT PROMPT (EXCERPT)

The excerpt below retains the Technique Matcher's main inputs and requested outputs, using OptTips terminology and descriptive runtime placeholders in double braces. Full prompts, schemas, and configurations are available in the anonymous repository.[2]

Technique Matcher

```
[System message]
You are the Technique Matching Agent in a multi-agent framework for optimization-modeling
    ↪ benchmark reformulation. Select suitable OptTips techniques for one raw benchmark
    ↪ problem. Return only valid JSON.

[User message: selected fields]
{
  "source_problem": "{{source_problem}}",
  "required_candidate_count": "{{techniques_per_problem}}",
  "technique_matching_rules": "{{matching_rules}}",
  "coverage_rules": "{{coverage_rules}}",
  "available_techniques": "{{OptTips_cards}}",
  "output_schema": {
    "candidate_techniques": [
      {
        "tech_id": "Txx",
        "reason": "why this source naturally supports this technique",
        "expected_general_llm_failure": "e.g. enumeration, loose Big-M, no dualization",
        "transformation_operator": "short operator name"
      }
    ]
  }
}
```

I TASK VALIDITY AND COMPUTATIONAL COST

This appendix expands Definitions 1 and 2 by explaining what a solution procedure must preserve and which stages contribute to its cost.

I.1 VALID SOLUTION PROCEDURES

For a task $p = (q, I, \mathcal{O})$, validity is judged against the original problem specification q, the fixed input I, and the required outputs and accuracy $\mathcal{O}$. If the task requests decisions, the returned decisions must satisfy the original domains and constraints and attain the required objective quality. Reporting the optimal value alone does not meet that requirement; any requested certificate must also be provided.

The solution procedure $\widehat{\mathcal{A}} = (\widehat{\mathcal{M}}, \widehat{C})$ combines a mathematical representation with executable code. Its variables and feasible set may differ from those of the original task, provided the transformation is justified and the code recovers the required answer. For example, an extended formulation may introduce auxiliary variables, with projection recovering the original decisions. A dual formulation requires the relevant duality conditions to justify equality of optimal values and, when original decisions are requested, a procedure for recovering them. A decomposition requires coordinated subproblem solves and a stopping rule that establishes the required solution quality for the full task.

[2] https://anonymous.4open.science/r/efficiency-EE0F/

I.2 Resource Accounting and Comparison Scope

An efficiency comparison concerns the same task, numerical input, and required answer quality, with an explicit resource and accounting scope. Time and memory are assessed separately. Costs depend on implementation and execution conditions as well as the formulation.

Complete request time. For a fully traced serial request, time can be partitioned into nonoverlapping phases, accumulated over all attempts and repairs:

$$\begin{aligned} T_{\text{request}} &= T_{\text{API}} + T_{\text{orchestration}} + T_{\text{execution}} + T_{\text{check}}, \\ T_{\text{execution}} &= T_{\text{setup/data}} + T_{\text{build}} + T_{\text{opt}} + T_{\text{between/recover}}. \end{aligned} \tag{2}$$

Here, T_{API} sums generation-call latency, including service and network waiting. Orchestration and checking cover work outside generation and execution. Within execution, T_{build} is model construction; T_{opt} includes every solver call, including those invoked during construction or within a decomposition. Computation between calls and answer recovery belong to $T_{\text{between/recover}}$. A recorded `Build` interval may span several of these phases; solver time already included in that interval must not be counted again.

Memory and modeling tradeoffs. Memory measurements require a stated scope, such as process memory or solver-managed allocation, and a sampling method; stagewise peaks cannot simply be summed. A technique may reduce solver time while increasing construction time or memory. Model-size counts help describe such changes but do not measure efficiency, and an expert reference need not be the least costly valid implementation.

Scope of the reported evidence. The final-call generation measurements and recorded execution and memory observations do not reconstruct the complete request budget in Equation 2. Appendix D.1 defines their measurement scopes; Appendix D.2 specifies eligible comparisons, aggregation, and execution conditions.

J OptTips: Technique Catalog, Coverage, and Extension

This appendix describes the coverage and structure of OptTips and lists all 50 techniques in Table 19. The complete cards are available in our anonymous repository.

J.1 Rationale and Coverage

Table 18: Primary reporting families of OptTips. Codes identify the assignments in Table 19; techniques can apply across families.

Code	Methodological family	Cards
A	Formulation standardization and dimension reduction	6
B	Relaxation strengthening and convexification	5
C	Duality, optimality, and equilibrium	4
D	Specialized combinatorial and structured models	8
E	Discrete and logical modeling	4
F	Convex, nonlinear, and structure-preserving modeling	9
G	Decomposition and large-scale optimization	6
H	Robust, stochastic, multiobjective, and solver-aware modeling	8
	Total	50

The collection combines three kinds of coverage. *Foundational coverage* includes recurring LP/MIP choices such as bounds, Big-M, linearization, cuts, and variable elimination. *Mechanism diversity* includes changes to representations, relaxation strength, and the organization of computation through duality and decomposition. *Broader methodological scope* includes uncertainty, equilibrium, nonsmooth, and matrix/tensor modeling. Table 18 groups the cards into eight families; the collection is a starting point, not an exhaustive catalog.

Each card receives one *primary reporting assignment* for counting, even when it applies across families. These counts describe knowledge coverage, not equally frequent or independent competencies. The reported Gurobi evaluation uses 561 tasks covering 29 cards. The remaining 21 cards have 63 supplementary tasks, three per card, outside the reported performance analysis. This count excludes extension tasks for already-covered cards.

J.2 Card Structure and Extension

Every card uses the same seven components shown in Figure 3: *identifier*, *core idea*, *applicable patterns*, *inefficiency symptoms*, *expert actions*, *mathematical example*, and *benchmark guidance*. Applicable patterns state the required structure and conditions; symptoms identify the modeling issue; expert actions specify the transformation. The mathematical example illustrates the transformation; benchmark guidance covers paired implementation, validation, and performance measurement.

Figures 12 and 13 give one worked example from each family, including applicability conditions, shared model terms, mathematical justification, and benchmark guidance. Case identifiers and recorded solver `Runtime` values refer to the illustrated instances.

New cards can extend the collection by following this schema and stating their mathematical justification. Associated tasks and paired references follow the validation process in Appendix E.

Future work: broader solver and algorithm coverage. Evaluation could extend to additional solver interfaces and the complete iterative and factor-based procedures of T14, T47, and T50, at matched solution accuracy and with the same hidden-technique task interface. Some supplementary tasks already use constructs supported by Gurobi, including second-order-cone formulations (T11) and SOS, indicator, and piecewise-linear formulations (T29) (Gurobi Optimization, 2026a); their exclusion from the current evaluation does not imply solver incompatibility.

Broader variants include T47's nuclear-norm SDP with positive-semidefinite constraints (MOSEK ApS, 2025) and T29's interval `NoOverlap` in CP-SAT (Google, 2024). Evaluating T14's ADMM requires its full multiplier-update iterations and residual-based stopping (Boyd et al., 2011).

J.3 Complete Technique Inventory

Table 19: All 50 OptTips techniques. Family codes refer to Table 18. The catalog includes formulation, algorithmic, and solver-aware choices; full cards are available in the anonymous repository.

ID	Technique	Family	ID	Technique	Family
T01	Equality/inequality equivalent transformation	A	T26	Complementarity and KKT/MPEC modeling	E
T02	Auxiliary-variable epigraph/hypograph modeling	A	T27	Multi-objective and lexicographic optimization	H
T03	Bound tightening and scaling	A	T28	Penalty, slack, and feasibility-relaxation modeling	H
T04	LP relaxation of IP/MIP	E	T29	Solver-native modeling constructs	H
T05	Big-M modeling	E	T30	Warm starts and incumbent injection	H
T06	Indicator, semicontinuous, and SOS constraints	E	T31	Algorithm–structure matching	H
T07	Valid inequalities and cutting planes	B	T32	Solver-log-guided diagnosis	H
T08	Variable substitution and product linearization	F	T33	Standard/canonical-form transformation	A
T09	Piecewise-linear modeling	F	T34	Slack, surplus, and artificial variables	A
T10	Convex-hull/envelope reformulation and convexification	B	T35	Variable elimination and dimension reduction	A
T11	Conic reformulation	F	T36	Convexity recognition and DCP modeling	F
T12	Dual transformation	C	T37	Smooth–nonsmooth composite modeling	F
T13	Lagrangian relaxation	G	T38	Convex conjugacy and Fenchel duality	F
T14	Augmented Lagrangian and ADMM	G	T39	Subgradient, normal-cone, and optimality-condition modeling	F
T15	Block decomposition	G	T40	Projection and proximal-operator modeling	F
T16	Benders decomposition	G	T41	Variational-inequality and equilibrium-constraint modeling	C
T17	Dantzig–Wolfe decomposition and column generation	G	T42	Sensitivity and parametric optimization	C
T18	Network-flow and matching formulations	D	T43	Dynamic programming and Bellman recursion	G
T19	Covering, partitioning, packing, and knapsack formulations	D	T44	Transportation and network-simplex structure	D
T20	Scheduling and temporal formulations	D	T45	Constraint qualifications and KKT validity	C
T21	Routing and path formulations	D	T46	Interior-point-friendly modeling	F
T22	Robust counterpart reformulation	H	T47	Matrix-variable and low-rank modeling	D
T23	Stochastic programming and scenario decomposition	H	T48	Cover, clique, subtour, and capacity cuts	B
T24	Preprocessing and dominance elimination	B	T49	Linking constraints and activation strengthening	B
T25	Symmetry breaking	D	T50	Tensor-variable and low-rank modeling	D

(a) A Implied variable bounds

Identifier: T03 **Case:** T03_001

Core idea: Expose bounds implied by precedence and conflict structure, strengthening variable domains before solving the integer model.

Applicable patterns: Integer energy quantities obey separation constraints; conflicting tasks require distinct operating windows with ordered binary activation indicators.

Inefficiency symptoms: Nonnegative quantity bounds and unrestricted binary window indicators leave useful implications implicit during search.

Expert actions: Propagate quantity lower bounds and fix the first activation indicators using a proven clique lower bound.

Mathematical example

Ordinary reference	Expert reference
$x - y \geq a, \quad y - z \geq b,$ $x, y, z \in \mathbb{Z}_{\geq 0},$ $u_w \geq u_{w+1}$	$x \geq a + b, \quad y \geq b,$ $u_w = 1 \quad (1 \leq w \leq \omega)$

Here $a, b \geq 0$ are separations and $\omega = 8$ is a verified conflict-clique size. Keep quantity upper bounds, capacities, task assignments, conflicts, activation order, and all costs in both models.

Benchmark guidance: Use identical energy plans and conflicts; check bound validity and objective agreement, then compare preprocessing, solver work, and `Runtime`.

Runtime: 2,619.12 s → 21.20 s both OPTIMAL

(b) B Pairwise assignment marginals

Identifier: T10 **Case:** T10_011

Core idea: Couple pairwise product variables through assignment marginals, giving an extended representation of each pair of joint choices.

Applicable patterns: Departments choose one city each, with binary assignments, city capacities, and communication costs coupling selected department pairs.

Inefficiency symptoms: Separate product inequalities allow weak fractional combinations; binary joint variables add discrete decisions.

Expert actions: Use continuous nonnegative joint variables and row/column marginal equations, preserving one-hot assignments, capacities, and all cost terms.

Mathematical example

GPT-5.5	Expert reference
$y_{ij}^{pq} \leq x_{pi}, y_{ij}^{pq} \leq x_{qj},$ $y_{ij}^{pq} \geq x_{pi} + x_{qj} - 1,$ $x_{pi}, y_{ij}^{pq} \in \{0, 1\}$	$\sum_j y_{ij}^{pq} = x_{pi},$ $\sum_i y_{ij}^{pq} = x_{qj},$ $x_{pi} \in \{0, 1\}, 0 \leq y_{ij}^{pq} \leq 1$

p, q index departments and i, j cities; y represents joint assignments. With nonnegativity and one-hot marginals, each pair has its joint-choice convex hull; shared capacities remain.

Benchmark guidance: Compare both formulations on identical assignment and interaction data; verify feasible decisions and equal objectives, then measure relaxation bounds, dimensions, and solver work.

Runtime: 3,693.24 s → 20.75 s both OPTIMAL

(c) C LP duality for absolute residuals

Identifier: T12 **Case:** T12_004

Core idea: Use LP duality to exchange observation-wise residual inequalities for bounded dual variables and coefficient-wise orthogonality equations.

Applicable patterns: Fit three unrestricted coefficients by minimizing total absolute residual; the primal LP must be feasible with finite optimum.

Inefficiency symptoms: Two residual inequalities per observation create a large explicit constraint matrix.

Expert actions: Derive the dual with correct signs and bounds; recover fitted coefficients from the dual solution when required.

Mathematical example

Ordinary reference	Expert reference
$\min_{\beta \in \mathbb{R}^3, d \geq 0} \mathbf{1}^T d$ $-d \leq A\beta - y \leq d$	$\max_w \; y^T w$ $A^T w = 0, \; -1 \leq w_i \leq 1$

A contains three features per observation, y the responses, and d absolute-residual bounds. The primal minimizes $\mathbf{1}^T d$; its dual maximizes the same value with $-1 \leq w_i \leq 1$.

Benchmark guidance: Use identical observations for both models; verify strong-duality conditions, matching objectives, and requested coefficient recovery, then compare dimensions, construction cost, and solver work.

Runtime: 279.26 s → 2.78 s both OPTIMAL

(d) D Network flow for tour connectivity

Identifier: T18 **Case:** T18_001

Core idea: Carry one commodity from a depot to every city, enforcing connectivity through flow conservation and selected-arc capacities.

Applicable patterns: The task requires a minimum-cost Hamiltonian tour, with binary arcs and one incoming and outgoing selected arc per city.

Inefficiency symptoms: Order-variable connectivity constraints can leave weak fractional tours in the LP relaxation.

Expert actions: Replace order constraints with depot supply, unit city demands, and nonnegative flows bounded by selected-arc capacities.

Mathematical example

Ordinary reference	Expert reference
$1 \leq u_i \leq n - 1,$ $u_i - u_j + n x_{ij} \leq n - 1$ $(i \neq j; \; i, j \neq 0)$	$0 \leq f_{ij} \leq (n-1) x_{ij},$ $\Delta f_0 = n - 1, \; \Delta f_i = -1$ $(i \neq 0; \; f_{i0} = 0)$

Both models minimize $\sum_{i \neq j} c_{ij} x_{ij}$ and retain degree equations. Δf is outflow minus inflow: depot 0 supplies $n - 1$ units, each other city consumes one, and $f_{i0} = 0$.

Benchmark guidance: Use identical cities and distances; verify one complete tour and equal objectives, then compare relaxation bounds, model dimensions, and branch-and-bound work.

Runtime: 249.47 s → 12.97 s both OPTIMAL

Figure 12: **OptTips cards for families A–D.** Each expanded card includes all seven components of the schema. Paired models highlight the transformation, with shared terms described in the card. Panel (b) retains the GPT-5.5–expert comparison from Figure 1; the other panels use ordinary and expert references. Recorded solver `Runtime` values have matching reported objective values.

(e) E Native semicontinuous domains

Identifier: T06 **Case:** T06_010

Core idea: Encode zero-or-positive production directly with a native semicontinuous domain, replacing explicit activation binaries and their linking inequalities.

Applicable patterns: Laboratory reagent batches have positive minimum and finite maximum sizes; activation appears only in quantity links, without separate costs.

Inefficiency symptoms: Each product-shift pair adds one binary and two linking rows to the model.

Expert actions: Declare production semicontinuous with the same bounds; preserve inventory balances, shared capacities, and production and holding costs.

Mathematical example

Ordinary reference

$$L_{pt} y_{pt} \le q_{pt} \le U_{pt} y_{pt}, \quad y_{pt} \in \{0, 1\}$$

Expert reference

$$q_{pt} \in \{0\} \cup [L_{pt}, U_{pt}] \quad \text{(native semicontinuous)}$$

Here q_{pt} is production and y_{pt} activation, with $0 < L_{pt} \le U_{pt}$. Both minimize production plus inventory cost. The zero-or-interval domain retains the discrete decision.

Benchmark guidance: Build both representations on identical data; verify feasibility and objective agreement, then compare variable counts, solver work, and `Runtime`.

Runtime: 411.77 s → 31.68 s both OPTIMAL

(f) F Convex piecewise-linear costs

Identifier: T09 **Case:** T09_008

Core idea: Represent convex piecewise-linear costs by supporting lines, removing discrete segment choices while preserving the minimum cost.

Applicable patterns: Allocations have four ordered segments, nonnegative nondecreasing slopes, group quantity requirements, and shared resource limits.

Inefficiency symptoms: Binary filling indicators introduce unnecessary integrality and enlarge the model for a convex cost function.

Expert actions: Replace incremental quantities and filling indicators with one bounded quantity and one epigraph cost per unit.

Mathematical example

Ordinary reference

$$x = \sum_k y_k, \quad y_k \ge \Delta_k z_k, \quad y_{k+1} \le \Delta_{k+1} z_k$$

Expert reference

$$c \ge m_k x + \beta_k \quad (\forall k), \quad 0 \le x \le b_4$$

For each unit, $\Delta_k = b_{k+1} - b_k$, $\beta_k = F(b_k) - m_k b_k$, and $0 \le y_k \le \Delta_k$. Ordinary cost is $\sum_k m_k y_k$; binary z_k links segments. Minimize summed costs; $k = 0, \ldots, 3$, with links for $k < 3$.

Benchmark guidance: Preserve group requirements and resource capacities; verify convexity and objective agreement, then compare binary counts, matrix size, and solver work.

Runtime: 378.62 s → 42.75 s both OPTIMAL

(g) G Lagrangian resource dualization

Identifier: T13 **Case:** T13_006

Core idea: Price the shared labor resource to separate technology blocks, then express their adjusted values through an exact LP dual.

Applicable patterns: Technology intensities form continuous simplices coupled by one labor budget; the primal LP must be feasible with finite optimum.

Inefficiency symptoms: A monolithic model retains many intensity variables despite limited coupling between blocks.

Expert actions: Dualize the labor limit with one nonnegative price and represent each block value using a free variable.

Mathematical example

GPT-5.5

$$\max_{x \ge 0} \ \sum_{i,k} v_{ik} x_{ik} \quad \sum_k x_{ik} = 1 \ (\forall i), \quad \sum_{i,k} r_{ik} x_{ik} \le R$$

Expert reference

$$\min_{\lambda, u} \ R\lambda + \sum_i u_i \quad u_i + r_{ik}\lambda \ge v_{ik} \ (\forall i, k), \quad \lambda \ge 0, \quad u_i \in \mathbb{R}$$

v_{ik}, r_{ik} denote value and labor, and R the budget. At price λ, each block maximizes $v_{ik} - \lambda r_{ik}$; minimizing $R\lambda + \sum_i u_i$ gives the primal optimum.

Benchmark guidance: Use identical values, labor requirements, and budget; verify strong duality and objective agreement, recover requested mixing decisions, then compare model size and solver work.

Runtime: 1,048.04 s → 4.44 s both OPTIMAL

(h) H Compact robust counterpart

Identifier: T22 **Case:** T22_008

Core idea: Dualize each budgeted deviation problem to replace enumerated scenarios with a compact formulation of the robust feasible region.

Applicable patterns: Production and deviations are nonnegative; each resource row has bounded uncertainty, with integer Γ for the displayed subset enumeration.

Inefficiency symptoms: Enumerating allowed deviation subsets creates combinatorially many constraints and increases construction workload.

Expert actions: Dualize the uncertainty budget and component bounds with nonnegative prices; add the resulting counterpart to each resource row.

Mathematical example

Ordinary reference

$$a^T x + \sum_{j \in Q} d_j x_j \le b \quad \forall Q \subseteq S, \quad |Q| \le \Gamma$$

Expert reference

$$a^T x + \Gamma\mu + \sum_j \lambda_j \le b, \quad \mu + \lambda_j \ge d_j x_j \ (\forall j \in S), \quad \mu, \lambda_j \ge 0$$

For each row, a, d, b give nominal coefficients, deviations, and capacity; $0 \le z \le 1$ and $\sum_{j \in S} z_j \le \Gamma$ define uncertainty. Preserve profit and other restrictions.

Benchmark guidance: Use identical uncertainty sets, objectives, and production restrictions; check inner LP duality and robust-feasibility equivalence, then compare rows, construction cost, and `Runtime`.

Runtime: 157.49 s → 13.63 s both OPTIMAL

Figure 13: **OptTips cards for families E–H.** The same seven-component schema continues with four further examples. Panel (g) retains the GPT-5.5–expert comparison from Figure 1; other panels use ordinary and expert references. Times report solver `Runtime` under the execution protocol in Appendix D; panel (h) lacks historical input-hash verification.