ORC-BENCH: BENCHMARKING LLMS CAPABILITIES FOR OPTIMIZATION AND REASONING UNDER CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) have achieved great performance on a broad range of knowledge and reasoning benchmarks, yet existing evaluations rarely probe their ability to reason and optimize end-to-end under interacting structural, physical, and operational constraints. We introduce ORC-Bench, a benchmark for evaluating LLMs on Optimization and Reasoning under Constraints over realistic structured inputs. ORC-Bench comprises 30 tasks spanning ten categories, three difficulty levels, and two data modalities, jointly probing six skills: graph understanding, tabular understanding, mathematical formulation, arithmetic execution, problem abstraction, and multi-step optimization. Unlike existing benchmarks, ORC-Bench scores models with constraint-aware metrics derived from physics-informed and domain-specific checkers, going beyond accuracy to measure structured-output validity and constraint-satisfaction rates. Evaluating nine LLMs across architectures, scales, and reasoning regimes, we find that while scale and explicit reasoning improve performance on our tasks, all models (including frontier reasoning models with human performance on MATH-500 and ARC-AGI) struggle to satisfy interacting physical and logical constraints while optimizing domain specific objectives.

1 INTRODUCTION

Large Language Models (LLMs) have shown great capabilities across a wide spectrum of tasks, ranging from open-ended dialogue and code synthesis to mathematical problem solving and scientific question answering (Guo et al., 2025; Srivastava et al., 2023; Wang et al., 2024). As frontier models continue to saturate established benchmarks, the community has shifted its attention towards evaluating ever more demanding cognitive abilities, such as expert-level knowledge (Rein et al., 2024), abstract pattern induction (Chollet, 2019), and software engineering (Yang et al., 2024). Yet, despite this expansion, a fundamental capability remains scarcely measured by existing benchmarks: the ability of an LLM to reason and *optimize* under interacting structural, physical, and operational *constraints*.

Real-world decision making indeed rarely reduces to retrieving a fact, completing a proof, or solving a self-contained puzzle. Instead, it typically requires (i) parsing structured, often noisy, inputs such as graphs or tables; (ii) abstracting the relevant variables from a verbose description; (iii) translating domain laws and operational limits into formal mathematical constraints; (iv) executing multi-step numerical computations; and (v) producing decisions that are jointly feasible with respect to several interacting constraints.

However, existing benchmarks do not isolate the joint capabilities of abstraction, mathematical formulation, arithmetic execution, and constrained multi-step optimization over structured inputs. General-purpose (Wang et al., 2024; Srivastava et al., 2023; Rein et al., 2024; Phan et al., 2025) and mathematical (Hendrycks et al., 2021) benchmarks probe declarative knowledge and chain-of-thought arithmetic on self-contained problems, without structured engineering artifacts. Constraint-satisfaction benchmarks (Chollet, 2019; Wei et al., 2025; Lin et al., 2025) do evaluate feasibility, but on synthetic puzzles that abstract away realistic, noisy domain structure, while agentic ones (Yang et al., 2024) blur the model's intrinsic reasoning with the assistance of external solvers, compilers, or retrieval.

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.

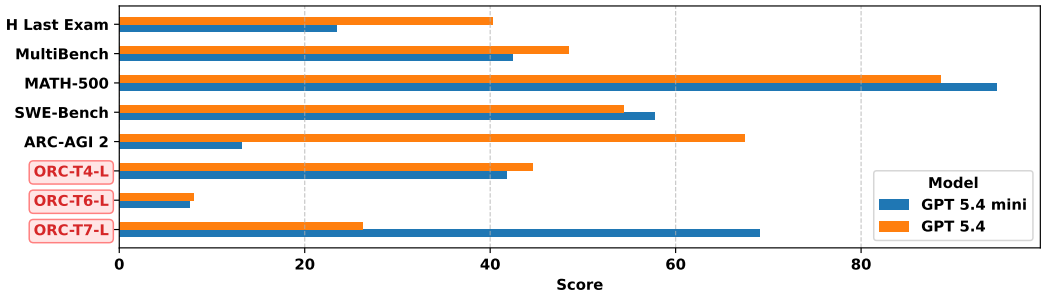


Figure 1: Comparing 3 tasks from our benchmark with popular LLM & AGI Benchmarks

To address this gap, we introduce **ORC-Bench**, a benchmark for evaluating LLMs on Optimization and Reasoning under Constraints, organized along a progression of cognitive skills. ORC-Bench consists of 30 tasks grouped into 10 task categories, each instantiated at three difficulty levels (Small, Medium, Large). The tasks span two structurally rich data modalities: graph-structured data (power grids of increasing topological complexity, from 14 to 118 nodes) and tabular data (credit scoring, URL phishing and network botnet detection). The categories form a difficulty ladder, starting from purely topological queries (path finding, connectivity), through quantitative physical reasoning (linear power flow, contingency analysis), and culminating in full constrained optimization (Optimal Power Flow, tabular constraint imputation) where models must simultaneously satisfy interacting physical or logical constraints. The last three categories invert the problem: rather than satisfying a constraint it is given, the model must recover one from the data. Each task is evaluated with metrics tailored to its nature, including structured-output validity, regression error, and constraint-satisfaction rates derived from physics-informed or domain-specific checkers.

We evaluate nine representative LLMs across three axes of variation: *architecture* (Llama, Qwen, OpenAI GPT families), *scale* (8B vs. 32B vs. frontier proprietary models), and *reasoning regime* (instruction-tuned vs. thinking variants). As highlighted in Figure 3a, ORC-Bench uncovers failure modes that remain largely invisible to existing reasoning benchmarks. While reasoning augmentation and model scale consistently improve topological reasoning, arithmetic accuracy, and output validity, all evaluated models exhibit a sharp degradation when required to jointly satisfy interacting physical or logical constraints. In particular, performance plateaus on coupled constraints such as Kirchhoff power-flow equations and financial invariants, even for frontier reasoning models: while reasoning models achieve more than double ($\times 2.5$) constraint satisfaction on our more complex task (task 6) compared to baselines, bigger models degrade it by 41% on average. Our contributions are:

- **A general benchmark for constrained reasoning and optimization (ORC-Bench)**: 30 tasks spanning ten categories, two data modalities (graphs and tables), three families (Open-ended, Close-ended Question Answering, and Regression) and three difficulty levels.
- **A rigorous evaluation protocol with constraint-aware metrics**, going beyond accuracy and regression error by tracking structured output validity and domain-specific constraint satisfaction rates derived from physics-informed and tabular constraint checkers.
- **An extensive empirical evaluation** of nine LLMs across architectures, scales, and reasoning regimes, showing that current models systematically fail on real-world, coupled constraints. We additionally observe strong miscalibration, weak robustness to distribution shifts, and several task regimes where smaller models outperform substantially larger ones.
- **A public benchmark API** with curated datasets, and a living leaderboard with secret evaluation sets (URL disclosed at publication).

2 Related Work

General-purpose LLM benchmarks. The evaluation of LLMs has rapidly evolved from broad knowledge tests such as MMLU (Wang et al., 2024) and BIG-bench (Srivastava et al., 2023) to expert-level question answering as in GPQA (Rein et al., 2024) and Humanity's Last Exam (Phan

et al., 2025). These benchmarks have been instrumental in tracking progress, but they primarily probe declarative knowledge and short-form reasoning, with limited structured input or constraint enforcement. Mathematical benchmarks including MATH-500 (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and competition-style suites have driven the development of chain-of-thought and reasoning-augmented models (Wei et al., 2022; Guo et al., 2025). However, these problems are self-contained, with a single objective and no interacting feasibility constraints, so high accuracy on them does not imply the ability to operate over structured artifacts such as graphs or tables.

Reasoning and constraint-satisfaction benchmarks. Closer to our setting, several benchmarks specifically target reasoning under constraints. ARC-AGI (Chollet, 2019) evaluates abstract pattern induction; PlanBench (Valmeekam et al., 2023) assesses planning and reasoning about change; CausalBench (Wang, 2024) focuses on causal inference. SATBench (Wei et al., 2025) and ZebraLogic (Lin et al., 2025) explicitly target constraint satisfaction, but rely on synthetic SAT instances or logic puzzles that abstract away from real domain structure. LR^{2}Bench (Chen et al., 2025) introduces long reflective reasoning over CSPs, while EHOP (Duchnowski et al., 2025) shows that LLM performance on NP-hard problems is highly sensitive to surface presentation, suggesting reliance on familiar formulations rather than robust abstraction. Work on natural-language satisfiability (Madusanka et al., 2024) similarly reveals brittleness across logical fragments. Compared to these efforts, ORC-Bench grounds constraint satisfaction in two realistic structured-data modalities (graphs and tables), with constraints arising from physical laws or domain-specific invariants rather than from synthetic templates.

Optimization and modeling benchmarks. A separate line of work evaluates LLMs as optimization *modelers* rather than *solvers*. NL4Opt (Ramamonjison et al., 2023) measures the recovery of variables, objectives, and constraints from natural-language descriptions, and recent benchmarks in the constraint-programming community extend this direction to executable constraint models (Michailidis et al., 2024; 2025). Neuro-symbolic approaches such as Logic-LM (Pan et al., 2023), SatLM (Ye et al., 2023), and LLM-generated streamlining constraints (Voboril et al., 2025) delegate the actual search to external solvers, which improves accuracy but moves away from end-to-end model competence. Domain-specific studies on power systems (Huang et al., 2024; Ren et al., 2025; Bernier et al., 2025) and code-augmented agents (e.g. SWE-Bench (Yang et al., 2024)) similarly rely on tools or solvers. ORC-Bench complements this literature by isolating what the model can do *by itself*: parse structured inputs, formulate constraints, execute arithmetic, and produce feasible solutions without external optimization tools. Concurrent benchmarks target optimization modeling (Tso et al., 2026; Chen et al., 2026; Mostajabdaveh et al., 2025), graph-theoretic tasks (Xu et al., 2025; Wang et al., 2023), and constrained planning (Xie et al., 2024); ORC-Bench goes beyond these by grounding tasks in real power-grid physics and extending to the inverse problem of recovering constraints from data. Supplementary A.1 details this comparison.

3 METHODS

Evaluating Large Language Models on constrained reasoning and optimization presents a unique challenge: it tests their ability to demonstrate true reasoning capabilities by solving multiple complex, iterative sub-problems concurrently. Unlike standard text-based question answering, optimization on graph and tabular data requires a synthesis of several advanced cognitive skills:

- **Graph Understanding:** The model must internalize textual representations of nodes and edges, tracking states and topologies.
- **Tabular Understanding:** The model must internalize textual representations of rows and columns and infer relationships between columns.
- **Mathematical Formulation:** The LLM must translate physical laws (e.g., Kirchhoff's laws), boundary relations (e.g., mean $<$ max), and operational limits into explicit mathematical constraints.
- **Arithmetic Execution:** Beyond formulating equations, the model must execute or estimate numerical computations (e.g., power distributions) and verify candidate constraints against sampled data.

- **Multi-step Optimization:** The model must synthesize these sub-problems to make sequential, constraint-satisfying decisions that optimize the system state.
- **Problem Abstraction:** The model must filter noisy telemetry, textual descriptions, and irrelevant columns to isolate the core problem and variables. This is required for all tasks.

By breaking down the overall optimization challenge into these specific sub-problems, this benchmark provides a rigorous framework for assessing whether an LLM is merely pattern-matching or performing genuine, multi-hop optimization reasoning. We summarize our tasks and skills in table 1.

Table 1: Summary of tasks and associated skills. REG: Regression. O-QA: Open-ended Question Answering. C-QA: Close-ended Question Answering. ZERO: Zero-shot learning. ICL: In context learning. Each Task category 1 to 10 contains 3 variants of increasing complexity: Small (S), Medium (M), Large (L). ● indicates that all variants require this skill, while ◐ means the S variant does not.

Feature	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**	**Task 6**	**Task 7**	**Task 8**	**Tasks 9-10**
Domain	Graph	Graph	Graph	Graph	Graph	Graph	Tabular	Tabular	Tabular
Type	O-QA	C-QA	REG	O-QA	C-QA	REG	REG	C-QA	C-QA
Setting	ZERO	ZERO	ZERO	ZERO	ZERO	ICL	ICL	ZERO	ZERO
Graph Understanding	●	●	●	●	●	●			
Tabular Understanding							●	●	●
Math Formulation			●	●	●	●	●	◐	●
Arithmetic Execution			●			●	◐	◐	●
Multi-step Optimization				●		●	◐		

Each task category consists of 3 tasks of increasing size and difficulty, denoted Small (S), Medium (M), and Large (L). S, M, and L tasks for graph tasks (categories 1 to 6) cover power grid size of 14, 30 and 118 nodes respectively. For tabular tasks (T7-T10), the S, M, and L task consist of the URL Phishing, Credit Scoring and Botnet use cases respectively.

In total our evaluation covers 30 benchmark tasks. We provide in Supplementary A more details about each, in particular examples of prompts and answers, and detailed definitions of the constraints. Grid tasks are illustrated in Fig. 2 for clarity purposes.

3.1 Task 1: Topological Path Finding

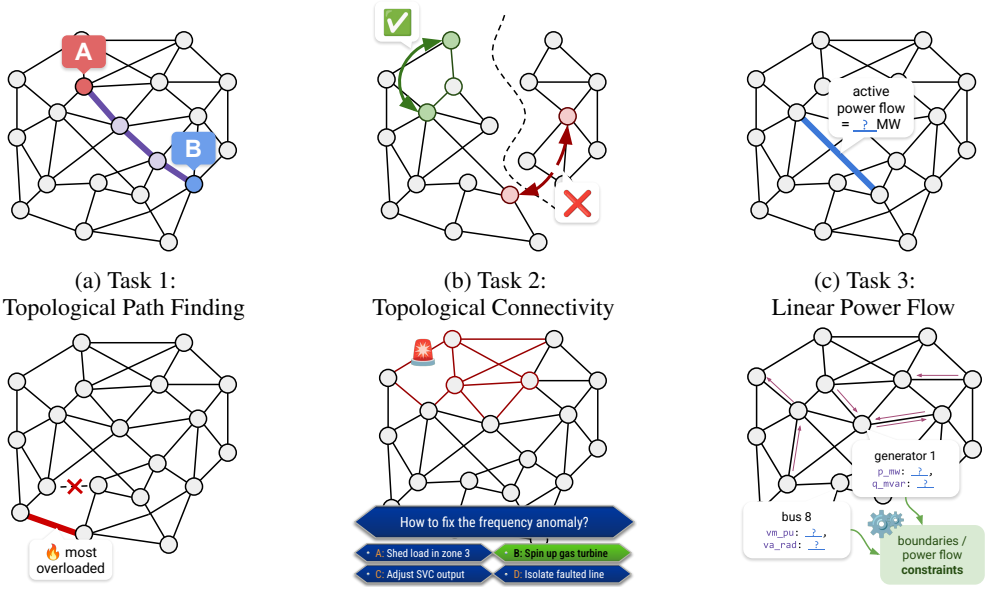
Problem Formulation: We test spatial reasoning by asking for the shortest path between two buses in a power grid whose topology may be modified (e.g., lines removed). **Dataset Generation:** From IEEE 14/30/118-bus cases, we randomly remove up to 2 (IEEE 14) or 5 (30/118) lines, build a NetworkX graph from the rest, and compute ground-truth shortest paths with `all_shortest_paths`. **LLM Evaluation:** Given the textual topology, the model outputs the shortest path, scored by exact match with strict alphanumeric boundaries (to avoid "Bus 2" matching "Bus 22"). Metric: **Accuracy**.

3.2 Task 2: Topological Connectivity

Problem Formulation: The model decides whether two buses are connected in a degraded grid (binary "Yes/No") from a text description. **Dataset Generation:** Using IEEE 14/30/118 graphs, we randomly remove up to 5 edges, sample two buses, and label connectivity via `networkx.has_path`. **LLM Evaluation:** The model answers only "Yes" or "No". Metric: case-insensitive exact-match **Accuracy**.

3.3 Task 3: Linear Power Flow

Problem Formulation: We test physical reasoning by asking for the active power flow (MW) on a target transmission line from a textual grid state. **Dataset Generation:** With PyPSA on IEEE 14/30/118, we randomize loads and generator capacities to 80-120% of base values, solve a DC linear power flow, and record the target line's flow. **LLM Evaluation:** The model predicts the target-line flow, extracted with regex. Metric: **MAE**.



(a) Task 1: Topological Path Finding

(b) Task 2: Topological Connectivity

(c) Task 3: Linear Power Flow

(d) Task 4: Contingency Analysis

(e) Task 5: Power Grid Control

(f) Task 6: Power Flow Optimization

Figure 2: Our 6 graph-based tasks. Each task is available in "S" format (for 14-bus grid), "M" (for 30-bus grid), and "L" (for 118-bus grid).

3.4 Task 4: Contingency Analysis

Problem Formulation: Given a base grid and an N-1 contingency (generator/line loss), the model predicts which line overloads most and explains why. **Dataset Generation:** Using PyPSA, we solve base flows for IEEE 14/30/118, apply a random contingency (trip a generator or remove a line), re-solve, and label the line with the largest flow change. **LLM Evaluation:** We check whether the predicted text mentions the correct overloaded-line identifier. Metric: **Accuracy**.

3.5 Task 5: Power Grid Control

Problem Formulation: As a grid-operator agent, the model selects the best corrective action from multiple choice given telemetry (frequency/voltage) and a disturbance. **Dataset Generation:** With PyPSA-based telemetry simulation, we generate 1,500 scenarios over four states (critical/moderate frequency drop, low voltage, stable), with actions deterministically mapped to states. **LLM Evaluation:** Given telemetry and actions (A-D), we score whether the output contains the correct choice. Metric: **Accuracy**.

3.6 Task 6: Optimal Power Flow Optimization

Problem Formulation: The model solves OPF end-to-end from a serialized heterogeneous graph (buses/loads/generators/lines), producing the optimal operating point under non-linear physics constraints and operational limits. **Dataset Generation:** We adapt the PyG `OPFDataset` for LLMs, serializing grids as structured JSON; few-shot ICL pairs an "Example Input JSON" with its solved "Example Output JSON". **LLM Evaluation:** Given instructions, few-shot examples, and a "Query Input JSON", we evaluate MSE to ground truth plus constraint metrics (Generator/Voltage/Power-Flow Limit Compliance) and an overall Valid Rate for structurally valid JSON.

3.7 Task 7: Tabular Data Imputation (Credit Scoring, Phishing, and Botnet Detection)

Problem Formulation: Given a JSON row with dropped fields, the model (as a finance/cyber expert) predicts missing values consistent with domain constraints across credit scoring, phishing, and botnet detection. **Dataset Generation:** We use Lending Club Loan Data (Lending Club, 2019), a phishing URL dataset (Hannousse & Yahiouche, 2021), and CTU-13 NERIS (Garcia et al., 2014), dropping target columns constrained by other features and building few-shot contexts from tabular rows. **LLM Evaluation:** We score predictions with the `TabularBench` (Simonetto et al., 2024) constraints checker. Metrics: **MSE** and Constraint Satisfaction (CTR), plus the rate of invalid JSON outputs.

3.8 Task 8: Constraint Discovery (Relation Discovery)

Problem Formulation: Given a sample of rows and two arithmetic expressions over the columns, the model determines which hard relation (four relations: $\leq$, $\geq$, $=$, or `none`) holds between them on every row. **Dataset Generation:** From a curated catalog of arithmetic constraints over our three tabular use cases (URL Phishing, Credit Scoring, CTU-13 Botnet), we draw 100 rows per question and present both sides as a CSV table, balanced and verified on the sampled rows. **LLM Evaluation:** The model returns `{"relation": <choice>}`, scored by exact match. Metric: **Accuracy**.

3.9 Task 9: Constraint Choice

Problem Formulation: Given a sample of 100 rows and four candidate constraints over the same columns, the model selects the one satisfied by every row; each of the other three is violated by at least one row. **Dataset Generation:** Reusing the `TabularBench`'s constraint catalog and sampling of Task 8, we extract the constraint expression (same operators, as well as $+, -, *, /, **$) and add three *distractor* options that are mutations of the true constraint (flipped inequalities, swapped operators or division operands), kept only when the sampled rows violate them, guaranteeing a unique, data-grounded answer among the four options (A-D). **LLM Evaluation:** The model returns `{"choice": <letter>}`, scored by exact match. Metric: **Accuracy**.

3.10 Task 10: Constraint Expression

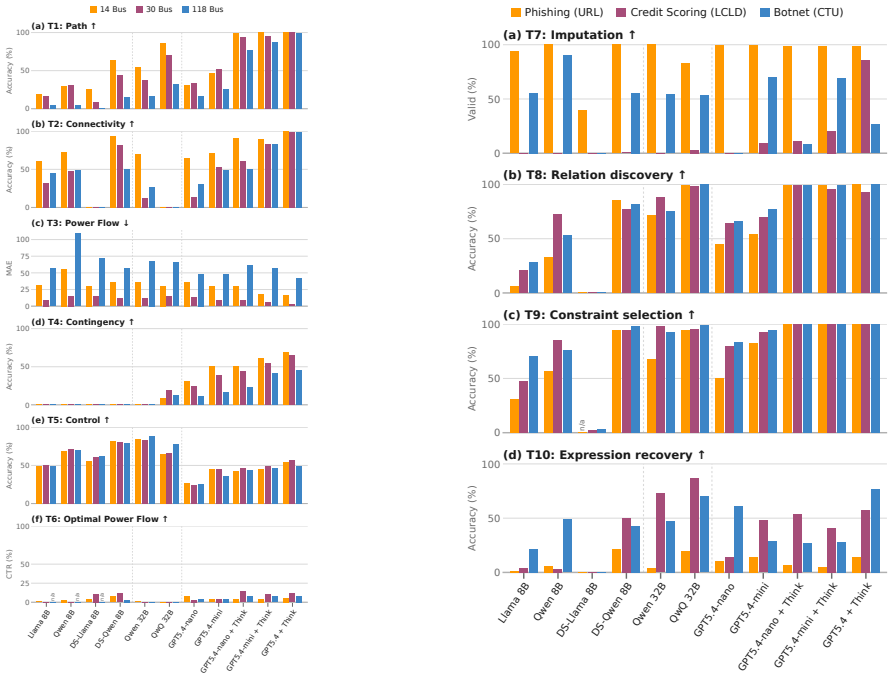
Problem Formulation: This task is essentially the same as Task 9, except the model has to find the constraint expression from scratch, without proposed options: instead of checking a constraint is correct, it has to *recover* it. **Dataset Generation:** The generation is based on Task 9; the prompt is slightly edited and constraint candidates are removed. **LLM Evaluation:** Predictions are parsed symbolically with `SymPy` and accepted when algebraically equivalent to the reference, up to positive constant multiples. Metric: **Accuracy**.

4 Experiments

4.1 Experimental Setup

Models. To highlight the impact of architecture, size and thinking in solving our tasks, we select same size but different (architectures `Llama3.1-8B` vs `Qwen3-8B`), different size (`Qwen3-8B` vs `Qwen2.5-32B`, `GPT-5.4-nano` vs `GPT-5.4-mini` vs `GPT-5.4`), and finally same size and architectures but with extended thinking options: `DeepSeek-R1-Distill-Llama-8B` extending `Llama3.1-8B`, `DeepSeek-R1-Qwen3-8B` extending `Qwen3-8B`, `QwQ-32B` extending `Qwen2.5-32B`, and `GPT-5.4-mini-high` (referred to as GPT-5.4-mini + Think) extending `GPT-5.4-mini-none`. Given the API cost of standard GPT-5.4 models, we do not report the performance of the non-reasoning variant. OpenAI model sizes are undisclosed; we use the $\sim$1.0T/410B/71B estimates of Li (2026) for GPT-5.4/mini/nano respectively.

Evaluation. Models are assessed in their default settings using regular system and per-sample prompts for tasks 1-5 and in-context learning (ICL) for tasks 6 and 7. For every ICL experiment, the model is assessed on 1,000 different conversations, the last message of each being the query input. In



(a) Tasks 1-6 (graph).

(b) Tasks 7-10 (tabular).

Figure 3: Summary of the performance of all the models across all ten tasks.

all cases, performance is computed on the parsed outputs; malformed outputs can't be parsed and are not evaluated.

Benchmark code and API. To facilitate the evaluation of models on ORC-Bench, we provide a Python API, designed to be modular and to standardize tasks running and scoring, which are two independent components. More details in Supplementary B.6.

4.2 Results: Graph Reasoning and Optimization (Tasks 1-6)

We summarize in Fig.3a the performance of the LLMs across all six tasks: panels (a-e) cover the zero-shot graph reasoning and optimization tasks (Tasks 1-5, Table 4), while panel (f) covers the in-context OPF task (Task 6, full table in Supplementary B.4.1), where models must produce JSON solutions jointly satisfying boundary and power-flow constraints. Detailed results with standard deviations, uncertainty and statistical analysis for Tasks 1-5 are provided in Supplementary B.2.2.

Impact of Size. On the largest grids, scaling up the LLMs is beneficial except for task 5. Accuracy of medium size models like Qwen2.5-32B reaches 88% while the GPT models remain below 50%. We identified 3 main reasons behind these failures and discuss them in Supplementary B.1.

Overall, Tasks 1 and 2 do not pose significant challenges for medium and large models, that pass the 50% accuracy threshold on average. They represent however challenging pure abstraction and reasoning benchmarks for small language models (even with reasoning).

On the contrary, Task 4 (S,M,L) remains extremely difficult for open-source small and medium models. The accuracy achieved by these models tops at 12.5% on the 118-bus grid.

For the in-context OPF task (Task 6), the picture differs markedly: while most models trivially satisfy structural validity and variable bound constraints (Bound CTR $\approx 100\%$), the power-flow equation satisfaction (PF CTR) remains strikingly low across all models, never exceeding $\sim$15%. Here, scale primarily improves *structural* reliability rather than physical correctness. On the largest grid (Case 118), small open-source models (Llama3.1-8B, Qwen3-8B) collapse to 0% validity, while frontier models (GPT-5.4, GPT-5.4-mini) maintain near-perfect validity ($\geq$98%). However, larger size yields

only marginal (and sometimes negative) gains on PF CTR (e.g., on Case 14, GPT-5.4-mini reaches 3.9% vs. GPT-5.4-nano's 7.6%; on Case 118, 4.1% vs. 3.8%), indicating that scale alone does not unlock physical-law reasoning.

Impact of Reasoning. We underline the value in Table 4 where the reasoning models are systematically better than their baseline non-reasoning equivalents. Overall, in 53 out of 75 scenarios, introducing reasoning steps improves the performance of the models, for an average accuracy increase of 11.5%.

On Task 6, explicit reasoning provides the largest lift on PF CTR: GPT-5.4-nano improves from 3.2% to 14.4% on Case 30, and GPT-5.4-mini from 4.6% to 10.4%; on Case 14 the effect is mixed (nano slightly decreases from 7.6% to 4.3%, while mini improves from 3.9% to 4.6%). Reasoning also reduces MSE for frontier models (e.g., GPT-5.4-mini on Case 14 drops from $3.29e^{-3}$ to $1.18e^{-3}$). Yet open-source reasoning variants (DeepSeek-Qwen, QwQ32B) often degrade validity, suggesting that reasoning helps only when paired with strong base capabilities.

Impact of constraints. On Task 6, a clear hierarchy emerges: bound constraints are nearly always satisfied, but the interacting Kirchhoff power-flow equations are systematically violated. Even the strongest reasoning model (GPT-5.4 + reasoning) only reaches 5.3-11.8% PF CTR across cases, exposing a fundamental plateau on jointly coupled physical constraints.

4.3 Results: Tabular Reasoning and Optimization (Tasks 7-10)

Per-domain numbers are reported in Supplementary B.5.

Impact of Size. Scale reliably improves checking, but not always constraint recovery. The Qwen model significantly improves its average accuracy (+20.3 points) over tasks 8 to 10. GPT-5.4-nano to GPT-5.4-mini gains +8.5 and +18.5 points on discovery and choice, and the Qwen model gains +25.0 and +13.7 with scale; Task 10, by contrast, moves by only 1.6 points from nano to mini. Imputation separates the models far more sharply: LCLD climbs from 0.21% for the 8B base models to 9.6% for GPT-5.4-mini, and up to 85.3% for the largest reasoning model (GPT-5.4+Think), the only model that genuinely "solves" it.

Impact of Reasoning. Reasoning closes the distance to a ceiling rather than raising it. Averaged over GPT reasoning/non-reasoning pairs, reasoning adds +36.0 and +19.2 points on discovery and choice, but degrades the performance on task 10, at $6\times$ the completion tokens. Thinking thus helps verifying a candidate constraint against the rows, it does not produce one. However from the open-source side, the trend doesn't follow, gaining +18.7 (DeepSeek-Qwen-8B) and +17.7 (QwQ-32B) on task 10 because their bases sit far below that ceiling. Reasoning also still requires a competent base: distilled onto Llama3.1-8B, it drives discovery to 0% on most tasks, mostly because of over-thinking and wrong answer formatting (this model specifically tends to answer in Latex instead of a formatted JSON).

Impact of Constraints and Task Difficulty. Difficulty partly depends on how tightly the constraints interact: on imputation, almost every model satisfies URL's constraints, while LCLD's fall only to GPT-5.4, similarly to the Kirchhoff plateau observation of Task 6. Furthermore, identifying a constraint the model is shown (Tasks 8 and 9) is close to solved, with five models above the 95% threshold; whereas writing constraints from scratch (Task 10) stays low and unpredictable: the best score in each domain comes from a different model, and the largest model leads in only one.

5 Discussion

Uncertainty and Calibration. Supplementary B.4.2 reports calibration metrics on the OPF `case14` task (6-S) and Task 7-S (URL). All three open-weight models are severely miscalibrated: mean confidence and ECE coincide (0.70-0.83), indicating that models report high confidence while being almost uniformly wrong on constraint satisfaction. Qwen3 8B is the most overconfident (ECE $= 0.83$, NLL $= 1.91$), and scaling up does not improve it (Qwen 32B showing ECE ≈ 0.70), suggesting that overconfidence on constrained optimization is a systematic failure mode rather than an artifact of capacity. We nonetheless observe an under-confidence (0.11-0.19) on Task 7-S specifically, where the task is easier and constraints are better satisfied, confirming the miscalibration: stated confidence does carry information about competence.

Impact of Prompts. Supplementary B.4.3 compares a basic OPF prompt against an enhanced variant that explicitly enumerates the physical constraints to satisfy. Across both `case14` and `case30`, enhanced prompting yields only marginal gains in constraint satisfaction (at most $+1.0$ point, for Qwen3 8B on `case30`), and validity is already saturated under the basic prompt. For tabular tasks, Supplementary B.1 shows that swapping two columns can create a surprisingly large performance gap, erased by reasoning. These results indicate the bottleneck on constrained OPF tasks is not prompt informativeness: making constraints explicit in natural language is insufficient to elicit constraint-satisfying solutions, reinforcing that LLMs plateau on joint constraint satisfaction.

Analysis of failure modes. We have analyzed the explanation and reasoning trajectories for GPT-5.4. Some span from fundamental misunderstanding of the modalities, the use-cases, or caused by default answers (potentially available in training sets) or wrong formats. We provide examples of traces, reasonings, and associated failures in Supplementary B.1.

ORC-Bench robustness to Manipulation and Optimization. On our harder task (OPF, task 6), Supplementary B.4.4 evaluates fine-tuning LLMs to beat our benchmark via supervised fine-tuning or GRPO, finding the best constraint satisfaction is only 24.6% on the 30-bus grid, and that these gains do not generalize: performance drops under small test-distribution shifts such as N-1 grids (one random line removed). Supplementary B.4.5 then tests whether this task can be solved by an LLM with coding tools and solver access, showing that even on the simplest 14-bus grid, average-sized LLMs remain insufficient and only GPT-5.4 with high Thinking produces the correct program; however, given Python tools, the evaluation mostly measures programming ability rather than reasoning and optimization.

Clear skill separation. The nomenclature in Table 1 reflects a minimum set of skills needed for each task rather than a fully controlled decomposition; more granular studies would be needed to fully separate the variables and mitigate confounders.

Output parsing. Results are sensitive to our regex-based parsing: some fundamentally correct answers may be marked incorrect when they don't match the required output format. We mitigate this by also reporting JSON validity, separating formatting failures from reasoning and optimization ones.

Compute and scaling. Frontier reasoning models generate substantially higher token and latency costs than 8B baselines, which we do not normalize (e.g., evaluating `gpt-5.4` alone requires roughly $500 in credits); constraint-satisfaction gains should be weighed against this cost when considering deployment.

Benchmark Integrity. Fine-tuning or tool augmentation could plausibly inflate scores without reflecting genuine constraint-reasoning gains; however, our results demonstrate that the improvements are minimal when LLMs are tuned against our samples. Unlike static benchmarks such as Humanity's Last Exam or ARC-AGI2, ORC-Bench can continuously generate challenging samples (load/topology variations, larger grids) at minimal human cost. These results also confirm that even if the public tabular datasets we are leveraging can happen to be included in the very large training sets of frontier models (potential data leakage), this is not sufficient for the LLMs to memorize these features.

6 Conclusion

This work introduces ORC-Bench, the first LLM benchmark for end-to-end reasoning and optimization over realistic, structured inputs (graphs and tables) with constraint-aware metrics, probing six skills across ten tasks and four real-world domains (grid optimization, credit scoring, phishing, botnet detection). While scale and explicit reasoning improve arithmetic and structural reliability, all nine evaluated LLMs systematically fail to satisfy complex, jointly coupled constraints; models are also heavily miscalibrated, and fine-tuning fails to generalize. We believe ORC-Bench reflects real-world skills expected of AGI while remaining hard to cheat, acting as a reality check against premature deployment in high-stakes environments and driving research towards neuro-symbolic integration and native constraint-aware architectures.

ETHICS STATEMENT

This work involves no human subjects, user studies, or collection of personal data. All source datasets are publicly available research benchmarks: the IEEE 14/30/118-bus power-grid test cases and the OPFDataset, the Lending Club loan dataset (Lending Club, 2019), the phishing-URL dataset of Hannousse & Yahiouche (2021), and the CTU-13 botnet traffic captures (Garcia et al., 2014). We use only feature-level records from these sources and process no personally identifiable information; ORC-Bench releases procedurally generated task instances derived from them, while its held-out evaluation sets are kept secret to limit benchmark contamination. Although several tasks are set in sensitive domains (credit scoring, cybersecurity, power-grid operation), the benchmark evaluates a model's ability to reason under explicit constraints, not to make decisions about individuals, and none of the tasks produce or improve offensive capabilities such as phishing or malware generation. We caution against deploying them as autonomous decision-makers in safety-critical infrastructure such as power grids, and intend ORC-Bench as a diagnostic to expose such failure modes before deployment.

AI USE STATEMENT

In this work, we used generative AI tools for two purposes: to assist with language editing of the manuscript text (grammar, phrasing, and readability polishing), and to assist in writing parts of the benchmark and evaluation code. We did not use generative AI tools for research ideation, experimental design, data collection or generation, or the analysis and interpretation of results. All AI-assisted code was reviewed and tested by the authors, and all AI-suggested text edits were checked against the underlying experiments and results before inclusion. We take responsibility for the final content of this work, including all text, claims, code, and artifacts.

REPRODUCIBILITY STATEMENT

The construction of each of ORC-Bench's ten task categories, including dataset generation procedures, prompts, evaluation metrics, and constraint checkers, is detailed in Appendix A, which also reports the fixed dataset-generation parameters, inference hyperparameters, and random seeds used identically across all evaluated models. The list of evaluated LLMs and the experimental protocol are described in Section 4.1, and per-task detailed results, statistical analysis, and failure-mode traces are provided in Appendix B. The benchmark evaluation API used to run and score all tasks is described in Appendix B.6 and will be released publicly; the accompanying leaderboard's secret evaluation sets will be disclosed at publication to preserve their value as held-out test data.

REFERENCES

Fabien Bernier, Jun Cao, Maxime Cordy, and Salah Ghamizi. Powergraph-llm: Novel power grid graph embedding and optimization with large language models. *IEEE Transactions on Power Systems*, 2025.

Jianghao Chen, Zhenlin Wei, Zhenjiang Ren, Ziyong Li, and Jiajun Zhang. LR^{2}Bench: Evaluating long-chain reflective reasoning capabilities of large language models via constraint satisfaction problems. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6006–6032, 2025.

Yitian Chen, Cheng Cheng, Yinan Sun, Zi Ling, and Dongdong Ge. Opt-engine: Benchmarking the limits of llms in optimization modeling via complexity scaling, 2026.

François Chollet. On the measure of intelligence, 2019. URL `https://arxiv.org/abs/1911.01547`.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Alex Duchnowski, Ellie Pavlick, and Alexander Koller. A knapsack by any other name: Presentation impacts LLM performance on NP-hard problems. In Christos Christodoulopoulos, Tanmoy

Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 6628–6651, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.352. URL https://aclanthology.org/2025.findings-emnlp.352/.

Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014. doi: 10.1016/j.cose.2014.05.011.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Abdelhakim Hannousse and Salima Yahiouche. Towards benchmark datasets for machine learning based website phishing detection: An experimental study. *Engineering Applications of Artificial Intelligence*, 104:104347, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Chenghao Huang, Siyang Li, Ruohong Liu, Hao Wang, and Yize Chen. Large foundation models for power systems. In *2024 IEEE Power & Energy Society General Meeting (PESGM)*, pp. 1–5. IEEE, 2024.

Lending Club. All lending club loan data: accepted and rejected loans, 2007–2018. Kaggle, 2019. URL https://www.kaggle.com/datasets/wordsforthewise/lending-club.

Bojie Li. Incompressible knowledge probes: Estimating black-box llm parameter counts via factual capacity. 2026. URL https://arxiv.org/abs/2604.24827.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Tharindu Madusanka, Ian Pratt-Hartmann, and Riza Theresa Batista-Navarro. Natural language satisfiability: Exploring the problem distribution and evaluating transformer-based language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15278–15294, 2024.

Kostis Michailidis, Dimos Tsouros, Tias Guns, and P. Shaw. Constraint modelling with llms using in-context learning. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307, pp. 20:1–20:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

Kostis Michailidis, Dimos Tsouros, and Tias Guns. CP-bench: Evaluating large language models for constraint modelling. *arXiv preprint arXiv:2506.06052*, 2025.

Mahdi Mostajabdaveh, Timothy T. Yu, Samarendra Chandan Bindu Dash, Rindranirina Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Evaluating LLM reasoning in the operations research domain with ORQA. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24902–24910, 2025.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824, 2023.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pp. 189–203. PMLR, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024.

Xinxing Ren, Chun Sing Lai, Gareth Taylor, and Zekun Guo. Can large language model agents balance energy systems? *arXiv preprint arXiv:2502.10557*, 2025.

Thibault Simonetto, Salah Ghamizi, and Maxime Cordy. Tabularbench: Benchmarking adversarial robustness for tabular deep learning in real-world use-cases. *Advances in Neural Information Processing Systems*, 37:78394–78430, 2024.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on machine learning research*, 2023.

Joseph Tso, Preston Schmittou, Quan Huynh, and Jibran Hutchins. Constraintbench: Benchmarking llm constraint reasoning on direct optimization, 2026.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36:38975–38987, 2023.

Florentina Voboril, Vaidyanathan Peruvemba Ramaswamy, and Stefan Szeider. Generating streamlining constraints with large language models. *Journal of Artificial Intelligence Research*, 84, 2025.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36:30840–30861, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

Zeyu Wang. Causalbench: A comprehensive benchmark for evaluating causal reasoning capabilities of large language models. In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pp. 143–151, 2024.

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. SATBench: Benchmarking LLMs' logical reasoning via automated puzzle generation from SAT formulas. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33820–33837, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: a benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Hao Xu, Xiangru Jian, Xinjian Zhao, Wei Pang, Chao Zhang, Suyuchen Wang, Qixin Zhang, Zhengyuan Dong, Joao Monteiro, Bang Liu, Qiuzhuang Sun, and Tianshu Yu. Graphomni: A comprehensive and extensible benchmark framework for large language models on graph-theoretic tasks, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580, 2023.

A Tasks and Datasets

A.1 Comparison with Related Benchmarks

Table 2 situates ORC-Bench relative to the closest optimization-, OR-, graph-, and planning-oriented LLM benchmarks. Unlike graph-algorithmic suites (NLGraph, GraphOmni), which pose no interacting feasibility constraints, and OR benchmarks (ORQA, OPT-Engine, ConstraintBench), which are confined to a single text modality with synthetic or templated instances, ORC-Bench grounds constraint reasoning in two structurally distinct, real-world data modalities (power-grid graphs and tabular records) and reports constraint-satisfaction rates derived from physics-informed and domain-specific checkers rather than accuracy alone. ORC-Bench is also the only benchmark in this comparison that requires models to invert the problem, recovering an unknown constraint from data rather than only satisfying one that is given.

Table 2: Comparison of ORC-Bench with related LLM reasoning/optimization benchmarks. ✓: yes, ×: no. *Solver-free* indicates whether the LLM must produce a feasible solution by itself, without calling an external solver. *Constraint-Satisfaction metric* indicates whether feasibility of the model's own output is scored.

Benchmark	Domain	Modality	Solver-free	Real-world grounded	Constraint-Sat. metric	Constraint recovery
NLGraph (Wang et al., 2023)	Graph algorithms	Graph (NL)	✓	×	×	×
TravelPlanner (Xie et al., 2024)	Travel planning	Text + tool APIs	×	✓	✓	×
GraphOmni (Xu et al., 2025)	Graph algorithms	Graph (NL)	✓	×	×	×
ORQA (Mostajabdaveh et al., 2025)	OR modeling (QA)	Text (NL)	QA only	✓	×	×
OPT-Engine (Chen et al., 2026)	OR optimization	Text (NL)	Partial	×	✓	×
ConstraintBench (Tso et al., 2026)	OR optimization	Text (NL)	✓	×	✓	×
ORC-Bench (ours)	Power systems, finance, security	Graph & Tabular	✓	✓	✓	✓

A.2 Task 1: Topological Path Finding

Problem Formulation This task assesses the spatial reasoning ability of the model by asking it to determine the shortest path between two specific buses in a given power grid state. The grid state may include modifications, such as the temporary removal of transmission lines, requiring the model to dynamically reason over the provided text-based topology rather than relying on memorized standard grid structures.

Dataset Generation The dataset is generated using standard Institute of Electrical and Electronics Engineers (IEEE) 14, 30, and 118 bus cases. To ensure diversity and require active reasoning, up to 2 unique lines (for IEEE 14) or 5 unique lines (for larger cases) are randomly removed. A NetworkX MultiGraph is constructed using the remaining operational lines (buses can be connected by more than 1 line in IEEE 118). Two random buses are selected as the source and target. The `all_shortest_paths` algorithm is used to compute the valid paths. If a path exists, the ground truth is formatted as a sequence of nodes (e.g., `Bus A -> Bus B -> Bus C`); if none exists, the answer is `No path exists.`

LLM Evaluation The LLM is provided with the textual representation of the grid topology (including the base system and any offline lines) and the query asking for the shortest path. **System Prompt:** "*You are a power systems expert. Provide the shortest path between the requested buses in the following format including the starting and ending buses: Bus A -> Bus B -> Bus C. If no path exists, respond with 'No path exists.' Do not provide any other text.*"

Performance Metrics Performance is evaluated using an exact path-matching algorithm. To prevent false positives from partial matches (e.g., predicting "Bus 22" when the ground truth is "Bus 2"), the evaluation script enforces strict alphanumeric word boundaries when searching for the ground truth path within the model's output. The primary metric is **Accuracy**, computed both overall and per IEEE case.

A.3 Task 2: Topological Connectivity

Problem Formulation This task evaluates the model's ability to deduce graph connectivity. The model must determine whether a path exists between two specific buses in a degraded power grid, effectively answering a binary "Yes" or "No" question based on the textual grid description.

Dataset Generation Similar to Task 1, the dataset is generated using **NetworkX**, built upon IEEE 14, 30, and 118 bus networks. Up to 5 edges are randomly removed from the base graph to create disconnected subgraphs or alternate routes. Two random buses are sampled, and the `networkx.has_path` function determines the ground truth connectivity state ("Yes" or "No"). The textual prompt lists the offline lines.

LLM Evaluation The model is prompted with the grid state and asked a direct question: "Is [Start Bus] connected to [End Bus]?" The exact system prompt used is: *You are a power systems expert. Determine if two buses are connected. Respond only with 'Yes' or 'No'. Do not provide any other text.*

Performance Metrics The metric for this task is binary **Accuracy**. The model's predicted response is compared against the ground truth using a case-insensitive exact match. Accuracy is reported across the entire dataset as well as per grid case.

A.4 Task 3: Linear Power Flow

Problem Formulation This task transitions from topological to physical reasoning by asking the model to calculate the active power flow (in Megawatts [MW]) on a specific transmission line. This tests the LLM's capability to understand electrical parameters and perform implicit physical calculations or heuristics based on grid state descriptions.

Dataset Generation The dataset is generated using the **PyPSA** (Python for Power System Analysis) library. It uses IEEE 14, 30, and 118 bus systems. To create unique scenarios, the active power load (`p_set`) and generator capacities (`p_nom`) are randomized uniformly between 80% and 120% of their base values. A Direct Current (DC) linear power flow (LPF) is solved using PyPSA's `network.lpf()` solver to obtain the steady-state active power flows. A target line is randomly chosen, and its simulated flow (rounded to two decimal places) is recorded as the ground truth.

LLM Evaluation The model is provided with the complete, randomized grid state (including node injections) and asked: "What is the active power flow (MW) on [Target Line]?" The exact system prompt used is: *You are a power systems expert. Calculate the requested physical value (e.g., Power in MW). Respond only with the numerical value. Do not provide units or any other text.*

Performance Metrics Because the output is continuous, performance is measured using **Mean Absolute Error (MAE)** and **Root Mean Square Error (RMSE)**. A regular expression extracts the first floating-point number from the model's response to compare against the ground truth flow.

A.5 TASK 4: CONTINGENCY ANALYSIS

Problem Formulation This task tests N-1 (N minus one) contingency analysis and counterfactual reasoning. Given a base grid state and a scenario describing the sudden loss of a generator or transmission line, the model must identify the transmission line most likely to overload (experience the highest flow increase) and provide a logical explanation for its choice.

Dataset Generation The dataset is generated using **PyPSA**. Base flows are calculated for IEEE 14, 30, and 118 cases using the PyPSA Linear Power Flow (LPF) solver. A random N-1 contingency is applied programmatically: either a non-slack generator is tripped (output set to 0), or a transmission line is removed. The system is re-solved, and the absolute difference in line flows is calculated to find the line with the maximum spike. The dataset includes a Chain-of-Thought (CoT) answer that explains the lost element, the rerouting of power, and the identification of the overloaded line.

LLM Evaluation The model receives the grid state and the contingency scenario. It must predict the overloaded line and justify it. The exact system prompt used is: *You are a power systems expert. Provide the line ID of the most likely overloaded transmission line in double brackets (e.g., [[L0]] for line 0) and then provide a step-by-step explanation.*

Performance Metrics The primary metric is **Accuracy** of identifying the correct overloaded line. Evaluation is performed by verifying if the ground truth target line string is explicitly mentioned within the predicted response.

A.6 TASK 5: POWER GRID CONTROL

Problem Formulation This task assesses the model's ability to act as a grid operator (agent). Given a set of system telemetry readings (frequency drop and voltage levels) and a sudden disturbance (e.g., wind generation drop), the model must choose the best corrective action from a multiple-choice list.

Dataset Generation The dataset is generated using **Python**'s standard libraries to simulate telemetry logic. For IEEE systems, 1,500 scenarios are generated representing four states: critical frequency drop, moderate frequency drop, low voltage, or stable system. Random telemetry parameters are generated within specific bounds for each state (e.g., critical frequency is between 49.0 and 49.49 Hz). Ground truth actions are deterministically mapped to the scenarios (e.g., a critical frequency drop requires spinning up a gas turbine).

LLM Evaluation The model is prompted with the grid state, telemetry data, and four possible actions (A, B, C, D). The exact system prompt used is: *You are a power grid operator. Based on the telemetry provided, select the most appropriate control action from the given choices. Respond by providing the choice label in brackets, e.g., [B].*

Performance Metrics The metric is **Accuracy**, calculated by checking whether the correct textual explanation or the correct action bracket (e.g., `[B]`) is present within the model's lowercased prediction string.

A.7 TASK 6: OPTIMAL POWER FLOW OPTIMIZATION

Problem Formulation This task represents the most complex numerical reasoning challenge, assessing the model's ability to act as an end-to-end mathematical solver for the Optimal Power Flow (OPF) problem. Given a serialized heterogeneous graph representation of a power grid—including buses, loads, generators, and transmission lines—the model must compute the optimal operating state. This requires balancing complex non-linear physical constraints (Kirchhoff's laws) and operational limits (voltage bounds, generator capacities) to determine active and reactive power injections and bus voltage magnitudes and angles.

Dataset Generation The dataset is procedurally generated using **PyTorch Geometric (PyG)** library. Specifically, we are curating and adapting the `OPFDataset` used for traditional GNN evaluation for a LLM setting. Complex grid topologies are parsed into a heterogeneous graph format,

which is then serialized into structured JSON dictionaries mapping node features and edge indices. To facilitate in-context learning, few-shot prompt examples are constructed, pairing an "Example Input JSON" with its corresponding solved "Example Output JSON".

LLM Evaluation The model is prompted with the system instructions, the few-shot examples, and a final "Query Input JSON" that it must solve. The exact system prompt used is: *You are a powergrid operator running Optimal Power Flow simulation and you need to return a JSON formatted responses based on the provided input JSON. The input is the description of the components of the grid, including the buses, generators, loads, lines, and external grid. The output is the solution to the optimal power flow problem. You will get a few examples of Input and Output json. You need to return the correct Output for the last given Input.* A robust regex-based extraction script parses the model's text output to isolate the final predicted JSON object.

Performance Metrics

- **Constraints:** We evaluate each predicted OPF solution using three feasibility criteria. For every bus (i), we compute the active and reactive power-balance residuals $\Delta P_i = P_i^{\text{spec}} - P_i^{\text{imp}}$ and $\Delta Q_i = Q_i^{\text{spec}} - Q_i^{\text{imp}}$, where the specified injections are given by the predicted generation minus demand, and the implied injections are those induced by the predicted bus voltages under the instance topology. A bus is deemed to satisfy the power-flow equations if
$$\sqrt{\Delta P_i^2 + \Delta Q_i^2} \le \varepsilon_{\text{pf}}.$$
Voltage feasibility is enforced via the bounds $v_i^{\min} \le \hat{v}_i \le v_i^{\max}$, and generator feasibility via the active-power limits $P_g^{\min} \le \hat{P}_g \le P_g^{\max}$.
For each sample, we compute the fraction of buses (or generators) that satisfy each condition, and report the dataset-level mean of these per-sample fractions. We additionally report an overall constraint satisfaction score obtained by micro-averaging all individual bus- and generator-level checks over the entire evaluation set, so that each constraint instance is weighted equally. Specifically, let (s) index evaluation samples, and let $n_{\text{pf}}^{(s)}$, $n_{\text{v}}^{(s)}$, and $n_{\text{g}}^{(s)}$ denote the numbers of evaluated buses and generators for the power-flow, voltage, and generator constraints, with $n_{\text{pf,sat}}^{(s)}$, $n_{\text{v,sat}}^{(s)}$, and $n_{\text{g,sat}}^{(s)}$ the corresponding numbers satisfied.
- **MSE:** To evaluate the numerical accuracy of the predicted OPF solutions, we compute the Mean Squared Error (MSE) between the model outputs and the ground-truth solver results in the dataset. We report the total of two components: (i) a generator MSE, calculated over each generator's active and reactive power outputs, and (ii) a bus MSE, calculated over the voltage magnitude and voltage angle at each bus. In addition to constraint satisfaction, the MSE quantifies how closely the predictions match the optimal dispatch numerically.
- **Valid rate:** tracks the model's ability to produce structurally correct JSON (can be parsed without raising error).

A.8 Task 7-S: Phishing Detection

Problem Formulation This task tests the model's capability in cyber-security, specifically acting as an expert in phishing detection. The model is provided with characteristics extracted from a URL (e.g., in JSON format) and must predict missing security details, such as the length of the URL, number of '.com' domains, presence of HTTP in the path, ratio of digits in the URL and host, and number of redirections.

Dataset Generation The dataset is generated from a Phishing URL tabular dataset (Hannousse & Yahiouche, 2021). It contains features from both benign and malicious URLs, as well as features obtained from external services, to build classifiers that distinguish phishing URLs from legitimate ones. URL-based features include counts of special substrings (e.g., "www", "&", ".", "$", "and"), the URL length, the port number, whether a brand name appears in the domain/subdomain/path, and whether the scheme is `http` or `https`. External-service features include signals such as Google indexing, page rank, and whether the domain appears in DNS records. A complete feature list is provided in their reproduction package.

There are 14 relational constraints over the URL features; among these, 7 are linear constraints (e.g., constraints that tie together component lengths so that they are consistent with the total URL length) and 7 are Boolean implication constraints of the form $a > 0 \Rightarrow b > 0$ (e.g., if the number of occurrences of `http` is greater than 0, then the number of slashes "/" must also be greater than 0).

A structured prompt is built containing a specific number of few-shot examples. Each example splits the tabular row into an input JSON containing URL characteristics and an output JSON holding the target security fields.

LLM Evaluation The model is evaluated on its ability to infer the missing URL characteristics from the given inputs. The exact system prompt used is: *You are a cyber-security expert in phishing detection. Your task is to analyze the provided URL data and predict the missing security details. You need to return a JSON formatted response based on the provided input JSON. The input contains various URL characteristics. The output should contain the predicted missing values for specific fields such as length_url, nb_com, http_in_path, ratio_digits_url, ratio_digits_host, and nb_redirection. You will receive an Input JSON and you must return the correct Output JSON.*

Performance Metrics The evaluation computes the **Mean Squared Error (MSE)** for continuous and discrete predicted URL properties. We also utilize the `tabularbench` (Simonetto et al., 2024) constraints checker to report the **Constraint Satisfaction (CTR)** rate, ensuring that the model's output satisfies real-world data constraints (e.g., non-negative lengths and ratios). The **Invalid JSON** error rate is also measured to assess the structural formatting reliability of the LLM.

A.9 TASK 7-M: CREDIT SCORING

Problem Formulation This task assesses the model's ability to act as a financial expert to analyze loan application data and predict missing financial details. Given a structured tabular representation of an applicant's characteristics and loan parameters (in JSON format), the model must predict missing values such as installment, open accounts, bankruptcies, term, and various financial ratios. This evaluates the LLM's capability to reason over financial constraints and tabular data structures.

Dataset Generation The dataset is derived from the Lending Club Loan Data (LCLD) dataset[1]. Each record corresponds to a loan that was approved by Lending Club. Some of these loans are not repaid and are instead written off ("charged off"). The goal is to predict, at the time the loan is requested, whether the borrower will fully repay the loan or whether it will be charged off.

The raw dataset contains 151 features. Only examples whose `loan_status` is either `Fully Paid` or `Charged Off` are kept, since these are the only definitive outcomes; all other statuses correspond to uncertain outcomes. The label is binary, where `Fully Paid` is 0 and `Charged Off` is 1. Variables that are not available at the time of the loan request were removed to prevent bias. In addition, variables that are redundant (e.g., grade and sub-grade) or overly granular for classification (e.g., address-like fields) were excluded. Categorical variables were then one-hot encoded. This process yields 47 input features and one target variable. The test set comprises 305K examples and is imbalanced: only 20% of loans are charged off (class 1).

The objective is to predict the value of a dropped column that is involved in constrained relationship with other columns. For the few-shot in-context learning evaluation, multiple context examples are selected, where each example presents an "Example Input JSON" (containing applicant and loan features) and an "Example Output JSON" (containing the target financial columns). The target variables include continuous and discrete fields.

The 19 variables are involved in 10 relationship constraints, including 3 linear and 7 non-linear constraints.

LLM Evaluation The model is prompted with the few-shot examples and asked to predict the target columns for a "Query Input JSON". The exact system prompt used is: *You are a financial expert. Your task is to analyze the provided loan application data and predict the missing financial details. You need to return a JSON formatted response based on the provided input JSON. The input*

[1] `https://www.kaggle.com/wordsforthewise/lending-club`

contains various applicant and loan characteristics. The output should contain the predicted missing values for specific fields such as installment, open accounts, bankruptcies, term, and various financial ratios. You will receive an Input JSON and you must return the correct Output JSON.

Performance Metrics Evaluation is conducted using a constraints checker from the `TabularBench` framework. The primary quantitative metric is **Mean Squared Error (MSE)** between the predicted numerical values and the ground truth. Additionally, structural and logical validity is measured via the **Constraint Satisfaction (CTR)** metric, which checks if the predicted fields satisfy intrinsic tabular constraints and mathematical invariants defined by the dataset. The proportion of **Invalid JSON** responses is also tracked.

A.10 TASK 7-L: BOTNET DETECTION

Problem Formulation This task assesses the model's proficiency in analyzing network traffic to identify botnet attacks. Acting as a cyber-security expert, the LLM analyzes a JSON representation of network traffic characteristics and must predict missing security details related to aggregated communication protocols (e.g., ICMP, UDP, TCP sums, byte flows, and durations).

Dataset Generation The dataset is processed from the CTU-13 NERIS tabular dataset. We reuse the feature-engineered variant of the CTU-13 dataset introduced by (Simonetto et al., 2024). The resulting dataset contains 0.74% of the examples labeled as botnet traffic. It includes 756 features in total, with two families of constraints. The first family constrains connection-count features so that an attacker cannot decrease the number of observed connections. The second family encodes protocol- and networking-consistency rules (e.g., the maximum packet size for TCP/UDP traffic is 1500 bytes). Altogether, 360 constraints are specified.

The data is partitioned, and a series of prompt contexts are generated. Each prompt includes multiple few-shot examples ("Example Input JSON" and "Example Output JSON"), isolating target columns like `icmp_sum_s_1`, `udp_sum_s_1`, `tcp_sum_s_1`, and byte/packet transfer sums from the rest of the network telemetry.

LLM Evaluation The LLM receives the prompt with few-shot network traffic examples and a target query. The exact system prompt used is: *You are a cyber-security expert in botnet attack detection. Your task is to analyze the provided network traffic data and predict the missing security details. You need to return a JSON formatted response based on the provided input JSON. The input contains various network traffic characteristics. The output should contain the predicted missing values for specific fields such as icmp_sum_s_1, udp_sum_s_1, tcp_sum_s_1, bytes_in_sum_d_2, bytes_out_sum_d_2, bytes_out_sum_s_1, pkts_out_sum_s_1, bytes_out_sum_d_3, pkts_out_sum_d_3, duration_max_d_1, and duration_max_s_OTHER. You will receive an Input JSON and you must return the correct Output JSON.*

Performance Metrics Performance is validated using the `TabularBench` framework. The predictions are scored using **Mean Squared Error (MSE)** against the ground truth ports values. A constraint satisfaction check calculates the **Constraint Satisfaction (CTR)** metric, which ensures that the interdependent packet and byte counts adhere to logical network constraints, alongside tracking the **Invalid JSON** structure rate.

A.11 TASK 8: TABULAR RELATION DISCOVERY

Problem Formulation This task tests whether a model can read a hard relation off data. Given a sample of rows and two expressions over the dataset columns, denoted LEFT and RIGHT, the model must decide which of four relations holds on *every* row: $\leq$, $\geq$, $=$, or `none`. The last class covers pairs whose ordering is inconsistent across rows, and prevents a model from succeeding by always asserting an ordering.

Dataset Generation This task reuses the three domains and the constraint sets already described for Task 7: URL Phishing (Section A.8), Credit Scoring (Section A.9) and Botnet (Section A.10). Only the arithmetic comparisons of those sets are retained (8 constraints for URL, 8 for LCLD and

360 for CTU-13) since a single comparison must serve as the answer; the Boolean implications of the URL dataset are therefore excluded. Guarded division is written `safe_div(numerator, denominator, fallback)`, which returns the fallback when the denominator is zero. Every question shows 100 rows drawn from a pool of 30,000 complete rows, questions cycle over the constraints so that each is covered uniformly, no two questions of a domain share the same sampled rows, and the seed of each draw is stored with the question for reproducibility. We generate 1,000 questions per task and per domain, and evaluate on all of them.

Questions are balanced over the four relation classes. The $\geq$ class is obtained by reversing the operands of the $\leq$ constraints, and the `none` class draws on dedicated column pairs (three per domain) known not to be ordered. Labels are grounded in the rows actually shown: a candidate sample is accepted only if the relation empirically observed on its 100 rows matches the target class, and is redrawn otherwise. When a constraint involves more than eight columns (as is common for CTU-13) the prompt reports the two evaluated columns `LEFT_VALUE` and `RIGHT_VALUE` instead of the raw features, keeping the table readable without changing the task.

LLM Evaluation The model receives the LEFT and RIGHT expressions followed by the sampled rows in CSV form. The exact system prompt used is: *You are given rows from a tabular dataset. Determine the hard relation that holds between the LEFT and RIGHT expressions in every row. Choose exactly one relation: <=, >=, ==, or none. Use none when the left expression is smaller on some rows and larger on others. LEFT_VALUE and RIGHT_VALUE, when present, are evaluated values of LEFT and RIGHT. Return only JSON in this format:* `{"relation": "<choice>"}`

Performance Metrics The metric is **Accuracy**. The first JSON object in the response is parsed and its `relation` field compared to the ground truth, case-insensitively; responses that cannot be parsed count as errors. Accuracy is reported overall, per relation class and per domain, the per-class breakdown revealing models that default to a single relation.

A.12 Task 9: Tabular Constraint Selection

Problem Formulation This task isolates constraint *recognition* from constraint *writing*. The model is shown a sample of rows together with four candidate constraints over the same columns, exactly one of which is satisfied by every row, and must return the corresponding letter. Compared with Task 8 it tests discrimination between complete arithmetic expressions rather than between four fixed relation symbols.

Dataset Generation The domains, constraints and sampling protocol are those of Task 8 (Section A.11); what is specific here is the construction of the three distractors accompanying each true constraint. Distractors are produced by mutating the true constraint: flipping the direction of an inequality, replacing an inequality by an equality, swapping arithmetic operators ($+/-$, $\times/\div$), exchanging the operands of a division or the first two arguments of a `safe_div` call, and, as fillers, offsetting the right-hand side by a small constant. Candidates are ordered from the most subtle transformation to the coarsest. Crucially, a candidate is retained only after it is *numerically verified to be violated* by at least one of the sampled rows, using a tolerance of 10^{-6} (10^{-2} for constraints annotated as approximate); a sample that does not yield three such distractors is discarded and redrawn. This guarantees that the question has a single correct answer that is grounded in the displayed data rather than in the catalog. The four options are then shuffled with a per-question seed, so the correct label is uniformly distributed over A–D.

LLM Evaluation The model receives the sampled rows and the four labelled candidates. The exact system prompt used is: *You are given rows from a tabular dataset and four candidate constraints. Exactly one candidate constraint is satisfied by every row; the other three are violated by at least one row. Select the one that holds. ANSWER FORMAT: Return exactly one JSON object:* `{"choice":"<letter>"}` *where* `<letter>` *is one of A, B, C, or D. Do not add prose, code fences, or additional JSON keys.*

Performance Metrics The metric is **Accuracy** on the selected letter. To avoid penalising formatting rather than reasoning, the parser accepts a JSON object, a `choice` field in free text, a bracketed

letter, or a bare letter, in that order; only responses from which no letter can be recovered are counted as errors.

A.13 TASK 10: TABULAR CONSTRAINT EXPRESSION RECOVERY

Problem Formulation This is the generative counterpart of Tasks 8 and 9, and the hardest of the three. Given a sample of rows and the list of columns it may use, the model must write out one complete hard arithmetic constraint that the data obeys, with no candidates provided. Success requires selecting the relevant columns, recovering the functional form and fixing any constants.

Dataset Generation The domains, constraints and sampling protocol again follow Task 8 (Section A.11). Each question exposes the sampled rows restricted to the columns of one constraint, together with an explicit `ALLOWED COLUMNS` list; constraints that hold only approximately carry a short note in the prompt. For equalities whose left-hand side is a single column, that column is recorded as the target of the constraint and used at scoring time. Answers must be written in a restricted grammar (numeric constants, parentheses, $+$, $-$, $\times$, $\div$, exponentiation, `safe_div`, and the allowed column names) containing exactly one of ==, <= or >=.

LLM Evaluation The exact system prompt used is: *You are given rows from a tabular dataset. Recover one complete hard arithmetic constraint that the allowed columns obey. ANSWER FORMAT: Return exactly one JSON object: `{"constraint":"<comparison>"}`. The constraint must contain exactly one comparison chosen from ==, <=, or >=. It may be algebraically rearranged and either side may be reversed when the comparison direction is changed accordingly. Use only numeric constants, parentheses, `+`, `-`, `*`, `/`, `**`, `safe_div(numerator, denominator, fallback)`, and the column names listed under ALLOWED COLUMNS in the message. Do not use strict comparisons, prose, assignments, code fences, or additional JSON keys.*

Performance Metrics The metric is **Accuracy**, but scoring is symbolic rather than textual, so that any correct algebraic form is accepted. The predicted constraint is parsed into an abstract syntax tree, rejected if it uses a name outside the allowed columns or more than one comparison, and converted to a SYMPY expression. Both prediction and ground truth are reduced to a signed residual ($\text{left} - \text{right}$ for $=$ and $\leq$, $\text{right} - \text{left}$ for $\geq$) which is simplified and rejected if identically zero, so that tautologies such as $x \leq x$ score no credit. A prediction is correct when its residual is a constant multiple of the reference residual: the multiple must be positive for inequalities, since a negative factor reverses the direction, and may be any non-zero real for equalities. When the reference constraint has a designated target column, both constraints are additionally solved for that column and their solution sets compared, which rules out expressions that are proportional yet define a different value for the target.

A.14 HYPERPARAMETERS

We deliberately keep the inference configuration minimal and identical across models, so that the reported differences reflect model capability rather than per-model tuning: no temperature, top-p or repetition-penalty sweep is performed, and every model is queried with its provider's default decoding settings. The only inference hyperparameters we set explicitly are the generation budget, capped at 8192 completion tokens, and the context window, set to 128k tokens for the open-weight models so that the largest in-context prompts of Tasks 6 and 7 fit without truncation. The reasoning regime is the single setting we vary, since it is one axis of comparison: thinking is disabled for the instruction-tuned variants and enabled for their reasoning counterparts (DeepSeek-R1 distills, QwQ-32B), while for the GPT-5.4 family it is controlled through the `reasoning_effort` parameter (`none` for the base variants, `high` for the $+$ Think ones). Dataset generation uses fixed parameters rather than tuned ones: 1,500 samples per grid size for Tasks 1-5, with up to 2 (IEEE 14) or 5 (IEEE 30/118) lines taken offline and loads and generator capacities drawn uniformly in $[80\%, 120\%]$ of their base values; eight solved examples in the few-shot context of Tasks 6 and 7; and 1,000 questions per domain for Tasks 8-10, each showing 100 rows sampled from a pool of 30,000. Scoring introduces three tolerances, all inherited from the checkers rather than chosen to flatter the models: $\varepsilon_{\text{pf}} = 1.414 \times 10^{-3}$ per unit for the bus power-flow residual of Task 6, 10^{-2} for the approximate equalities of the TabularBench constraints (Tasks 7-10), and 10^{-6} for the exact ones. All random draws are seeded and every

generated question stores the seed of its own sample, so each task instance can be regenerated independently.

B Detailed results

B.1 Analysis of failure modes

- **Confusion and misidentification of generators (Task 3-6)**: Example of LLM's Claim: "G1 is most likely the generator at Bus 1, which is the largest source..." While there are no generators connected to the specific bus 1.
- **Misunderstanding the slack bus mechanism (Tasks 4-6)** : Example of LLM's Claim: "The lost power has to be replaced by the remaining generators at Buses 2, 3, 6, and 8." While in practice, the slack bus (Bus 1 / G0) compensates for 100% of the active power mismatch caused by a tripped generator.
- **Flawed power flow tracing (Tasks 4 and 6)**: Example of LLM's Claim: "Therefore, Line L2 (Bus 2-3) is the prime stress point because it serves the biggest demand sink." When the LLM hallucinates local load pockets, it consistently defaults to predicting specific critical bottlenecks (L30: 63 times; L2: 61 times, L10: 50 times...) It defaults back to the original bottlenecks of such grid without considering the change of loads we applied or the lines or the generator that tripped (bottlenecks described in the literature, including its training datasets).
- **Schema drift on tabular outputs (Task 7, CTU):** On the Botnet (CTU-13) sub-task, GPT-5.4-nano systematically deviates from the expected output schema by either adding extra columns (e.g., re-emitting input fields) or renaming target columns (most often `pkts_out_sum_d_3`, typically into `pkts_out_sum_d_2`). This schema drift directly explains the sharp drop in validity rate to almost 0%. Reasoning limits this effect and drives the model to rise the bar to 8.55%.
- **Blindness to the comparison direction (Tasks 8-9):** For the non-reasoning models the binding difficulty is the operator. On Task 8, GPT-5.4-mini identifies $\leq$ on 83.6% of the items but its mirrored $\geq$ counterpart on only 9.5%, routing half of those to `none`; GPT-5.4-nano shows the same asymmetry (61.5% against 34.0%). Task 9 confirms the diagnosis from the other side: 78-79% of the errors of both non-reasoning models select either the direction-flipped or the (in)equality distractor, whereas the structural mutations (swapped arithmetic operators, exchanged division operands) attract at most 9%. Models discriminate the *direction* of a constraint poorly; **reasoning removes the asymmetry** almost entirely, both directions exceeding 98%. More details in Table 3.

Table 3: Task 8 accuracy (%) by relation class

Model	URL		LCLD		CTU		mean $\lvert\Delta\rvert$
	$\leq$	$\geq$	$\leq$	$\geq$	$\leq$	$\geq$	
Llama 8B	4.8	4.8	5.2	6.0	4.8	12.0	2.7
Qwen 8B	66.5	23.7	97.2	15.6	57.2	50.4	43.7
DeepSeek-Llama 8B	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DeepSeek-Qwen 8B	91.0	76.6	58.0	62.4	80.4	59.6	13.2
gpt-5.4-nano	47.3	32.1	76.4	42.8	65.6	27.6	28.9
gpt-5.4-nano + Think	99.4	99.1	97.6	97.2	99.6	98.8	0.5
Qwen 32B	74.6	45.3	60.8	90.8	96.0	73.6	27.2
QwQ 32B	99.1	98.4	96.8	97.6	100.0	99.1	0.8
gpt-5.4-mini	61.7	4.5	91.2	12.0	98.0	12.0	74.1
gpt-5.4-mini + Think	99.7	99.4	96.0	98.8	99.2	97.6	1.6
gpt-5.4 + Think	100.0	100.0	100.0	100.0	100.0	100.0	**0.0**

- **Answering without consulting the rows (Task 9):** A further 14-19% of the errors of the non-reasoning models fall on the constant offset distractors of the form $\text{left} = \text{right} + k$,

which a single displayed row is enough to refute. These selections indicate that the candidate is judged on plausibility rather than checked against the sample.

- **Over-strict equality checking (Task 8, GPT-5.4):** Reasoning does not eliminate every systematic error. GPT-5.4 + Think is perfect on both inequality classes yet falls to 84.8% on equalities, and 13.4% of them are declared `none`. Every such failure is a Credit Scoring ratio column (e.g. `ratio_open_acc_total_acc == open_acc / total_acc`): the model treats the rounding of the printed decimals as evidence that no exact equality holds, and reports the absence of a relation rather than a numerical tolerance.
- **Valid but unintended constraints (Task 10):** The low expression-recovery scores do not indicate malformed or arbitrary answers. Malformed JSON, invalid syntax, disallowed columns and tautologies together account for less than 1% of the errors, and 93–95% of the rejected constraints are in fact satisfied by all 100 displayed rows. Models therefore recover *one* true relation rather than *the* dataset invariant, and they do so by generalizing too little: 43% of the rejected answers are structurally simpler than the reference, and the smaller models restrict themselves to a subset of the allowed columns in 21-35% of their errors, against 8% for GPT-5.4, whose errors instead have the right scope and the wrong algebra (59%). Three recurring patterns illustrate this: latching onto a column that happens to be constant in the sample (`nb_colon == 1`), restating a variable's domain (`ip * (ip - 1) == 0`), and tightening a bound with slack observed in the sample (`length_url >= length_hostname + 7` in place of `length_hostname <= length_url`). This also delimits what the metric measures: an answer that is strictly stronger or weaker than the reference, while still true of the data, is counted as incorrect.

B.2 Zero-shot Graph Reasoning and Optimization

B.2.1 Full results

Table 4: Accuracy (%) for the QA benchmarks: Shortest path and contingency analysis open-ended question, bus connectivity and power grid control close-ended questions. MAE for regression task: The linear powerflow task (T3). ↑ means higher is better, ↓ means lower is better. Underline the cases where the thinking variant is better.

Model	T1: Path ↑			T2: Connectivity ↑			T3: PowerFlow ↓			T4: Contingency ↑			T5: Control ↑		
	14	30	118	14	30	118	14	30	118	14	30	118	14	30	118
Llama 8B	18.5	16.3	4.3	60.9	32.3	45.5	30.9	9.1	56.2	0.2	0.7	0.1	49.6	51.0	49.8
Qwen 8B	29.9	31.4	4.6	72.5	48.0	49.6	55.6	15.4	109.0	0.0	0.0	0.0	68.6	71.4	70.0
DeepSeek-Llama 8B	25.0	8.0	1.25	0.0	0.0	0.0	30.1	14.5	71.5	0.0	1.3	1.3	56.0	61.3	62.0
DeepSeek-Qwen 8B	63.3	44.3	14.7	93.2	82.4	50.0	35.6	11.1	56.2	0.0	0.0	0.0	82.0	81.2	78.8
Qwen 32B	53.9	37.2	16.1	70.4	12.5	26.4	36.5	12.0	67.6	0.0	0.3	0.0	85.2	83.0	88.0
QwQ 32B	86.0	70.0	32.0	0.0	0.0	0.0	30.3	14.2	65.7	9.1	19.2	12.5	65.0	66.0	78.0
GPT-5.4-nano	31.0	33.0	16.6	64.6	13.6	30.6	36.4	12.7	48.1	31.0	23.8	11.6	26.9	24.5	25.1
GPT-5.4-nano + Think	99.0	94.0	76.2	91.2	61.2	50.6	29.6	8.4	61.5	51.0	43.6	23.0	42.5	46.4	43.8
GPT-5.4-mini	46.6	52.4	25.2	72.0	53.4	49.2	29.8	8.3	47.3	50.2	38.6	16.6	45.8	45.3	36.2
GPT-5.4-mini + Think	100	95.6	87.4	89.2	83.8	83.2	17.8	5.8	56.5	60.8	53.8	41.8	45.2	49.4	46.3
GPT-5.4 + Think	100	100	99.4	100	99.6	99.6	16.3	3	41.8	68.8	65.4	44.6	54.3	56.4	49.3

B.2.2 Statistical Analysis

We provide further statistical details for regression tasks (task 3).

Table 5: Evaluation results for Task 3 split by Model, Reasoning level, and Grid Size. Values are reported as mean ± uncertainty (standard error of the mean, computed over 5 random splits of the per-sample errors).

Model	Reasoning	Grid Size	MAE	RMSE
gpt-5.4-mini	high	14	17.81 ± 1.75	31.22 ± 2.40
		30	5.77 ± 0.45	9.82 ± 0.85
		118	56.53 ± 11.61	256.89 ± 94.84
gpt-5.4-mini	none	14	29.81 ± 1.74	51.27 ± 4.49
		30	8.26 ± 0.39	11.44 ± 0.71
		118	47.26 ± 2.64	76.84 ± 5.13
gpt-5.4-nano	high	14	29.55 ± 1.74	48.41 ± 3.09
		30	8.43 ± 0.51	13.83 ± 1.23
		118	61.52 ± 3.03	111.82 ± 9.04
gpt-5.4-nano	none	14	36.36 ± 2.43	56.56 ± 3.71
		30	12.67 ± 0.36	17.94 ± 0.52
		118	48.13 ± 2.81	78.81 ± 5.25
gpt-5.4	high	14	16.29 ± 1.10	25.93 ± 1.47
		30	3.03 ± 0.29	6.04 ± 1.22
		118	41.75 ± 2.84	71.41 ± 8.28

B.2.3 TRACES AND RESPONSES

We report for each paragraph a category of failure then one example per category.

Line Trip Error (Meshed KVL Redistribution) In the **ieee14** grid size analysis, the target ground truth was identified as **L4**, while the model predicted **L11**.

Correct Reasoning: 1. In the ieee14 system, L10 is lost. 2. Power previously flowing through the tripped line must now find alternative paths. 3. Comparing the new flows to the baseline, L4 sees the highest absolute increase in loading. Therefore, L4 is most likely to reach its limit.

Wrong Reasoning: The model's incorrect prediction was based on the following logic: 1. *L10 (Bus 7–Bus 8) is a radial generator tie.* Bus 8 connects to the rest of the system only through L10, so when L10 trips, Bus 8's generation is effectively disconnected. 2. *Lost generation replacement.* The remaining sources are mainly the upper network generators and Bus 6. 3. *Affected area.* Power to the eastern pocket (Buses 7/9/10/14) must route through T1 (4–9), T0 + L11 (4–7–9), or paths from Bus 6. 4. *Impedance.* The 4–7–9 path is lower impedance, picking up a larger share of flow. 5. *Conclusion.* This makes L11 (7–9) the most likely line to overload as the key feeder to the Bus 9 area.

Slack Bus Misunderstanding (Assumed Distributed Slack) In the **ieee14** grid size analysis, the target ground truth was identified as **L0**, while the model predicted **T2**.

Correct Reasoning: 1. In the ieee14 system, G4 is lost. 2. The slack bus must increase output to compensate for the lost generation. 3. Comparing the new flows to the baseline, L0 sees the highest absolute increase in loading. Therefore, L0 is most likely to reach its limit.

Wrong Reasoning: The model's incorrect prediction was based on a local source redistribution logic: 1. *Generator mapping.* The model assumes G4 is at Bus 6. 2. *Local source loss.* Tripping G4 removes support for the Bus 6 pocket (Buses 11, 12, 13). 3. *Replacement paths.* Power must move from the western core to the Bus 6 area. 4. *Transformer pathing.* The model identifies T2 (5-6) as the strongest import path, assuming alternative paths through T0/T1 are too high-impedance. 5. *Conclusion.* The replacement power is expected to concentrate on the 5-6 corridor, incorrectly identifying T2 as the primary overload candidate.

Slack Bus Misunderstanding (Failed to identify primary flow path) Within the **ieee14** system analysis, the target ground truth was identified as **L0**, while the model predicted **T2**.

Correct Reasoning: 1. In the ieee14 system, G4 is lost. 2. The slack bus is responsible for increasing output to compensate for the lost generation. 3. Analysis of the updated flow shows that **L0** sees the highest absolute increase in loading, making it the most likely element to reach its limit.

Wrong Reasoning: The model's error stems from a localized load-balancing perspective: 1. *G4 Placement.* It assumes G4 is at Bus 6. 2. *Regional Deficit.* Loss of G4 impacts the load pocket at Buses 6, 11, 12, and 13. 3. *Transfer Logic.* It assumes replacement power travels from the western generation core toward Bus 6. 4. *Path Preference.* T2 (5-6) is identified as the direct import path, whereas T0 and T1 are dismissed as electrically longer. 5. *Conclusion.* The model incorrectly assumes the largest incremental transfer will concentrate on T2.

Generator Trip Error (Other) In the **ieee118** grid size analysis, the target ground truth was identified as **L89**, while the model predicted **L30**.

Correct Reasoning: 1. In the ieee118 system, G13 is lost. 2. To maintain system balance, the slack bus must increase output to replace the lost generation. 3. Flow comparison shows that **L89** sees the highest absolute increase in loading, identifying it as the primary candidate for reaching its limit.

Wrong Reasoning: The model's error stems from a localized import-dependency logic: 1. *Local Mapping.* G13 is assumed to be at Bus 27. 2. *Shift to Import.* The loss of G13 forces Bus 27 to import power from the western subnetwork. 3. *Path Determination.* The model identifies the 26–25–27 corridor as the strongest source of replacement power. 4. *Sensitivity.* **L30 (25–27)** is identified as the highest-sensitivity corridor into the deficit area. 5. *Conclusion.* The model incorrectly assumes the resulting transfer surge will concentrate on L30, leading to an overload.

B.3 In-Context Learning Ablation on Tasks 1-5

Tasks 1-5 are otherwise evaluated zero-shot (Table 4); here we instead prompt each model with up to 30 worked examples drawn from other rows of the same IEEE case, automatically truncated on ieee118 to fit the context window.

Table 6: Accuracy (%) / MAE under 30-shot in-context learning for Tasks 1–5, same metrics and column layout as Table 4. Underline marks cells where in-context learning improves over the zero-shot baseline of Table 4.

Model	T1: Path ↑			T2: Connectivity ↑			T3: PowerFlow ↓			T4: Contingency ↑			T5: Control ↑		
	14	30	118	14	30	118	14	30	118	14	30	118	14	30	118
Llama 8B	22.0	24.3	10.8	56.6	95.4	12.2	10.92	6.87	54.11	75.9	63.6	13.3	43.0	38.2	26.1
Qwen 8B	34.5	38.6	6.4	79.7	95.4	26.6	7.14	5.83	57.42	72.0	65.5	28.6	77.8	84.8	54.2
DeepSeek-Llama 8B	75.9	20.0	10.9	0.0	0.0	0.0	26.28	10.79	515.89	74.1	56.0	20.3	75.3	79.3	25.4
DeepSeek-Qwen 8B	85.7	78.2	26.3	96.9	91.4	36.2	9.09	11.03	77.68	87.7	70.8	27.3	77.0	80.2	38.8
Qwen 32B	68.3	45.6	14.7	79.3	71.3	45.7	7.15	6.29	64.47	87.4	67.8	25.5	87.4	85.5	71.2
QwQ 32B	98.3	88.1	33.0	0.0	0.0	0.0	29.09	14.52	76.27	70.1	66.4	47.5	50.9	59.3	51.1

B.4 In-Context Graph Reasoning and Optimization

B.4.1 Full results

Table 7: Q2: Graph Reasoning and Optimization. Results (± standard error) are grouped by grid size with metrics as subcolumns: Structure validity, boundaries constraint satisfaction, power flow equations satisfaction, and mean squared error.

Model	Validity	Bound CTR	PF CTR	Gen MSE	Bus MSE
Case 14					
Llama 8B	100.0	100.0 ± 0.0	1.0 ± 0.1	4.68e-03 ± 1.85e-04	1.22e-04 ± 4.50e-06
Qwen 8B	99.8	100.0 ± 0.0	2.8 ± 0.2	3.52e-03 ± 1.37e-04	9.83e-05 ± 3.80e-06
DeepSeek-Llama 8B	28.8	99.2 ± 0.4	3.9 ± 0.4	6.47e-02 ± 1.35e-02	6.29e-04 ± 1.17e-04
DeepSeek-Qwen 8B	84.9	99.3 ± 0.1	7.5 ± 0.3	2.95e+01 ± 2.71e+01	2.82e-02 ± 4.37e-03
gpt-5.4-nano	99.9	100.0 ± 0.0	7.6 ± 0.3	3.10e-03 ± 1.18e-04	9.48e-05 ± 3.61e-06
gpt-5.4-nano + reasoning	100.0	100.0 ± 0.0	4.3 ± 0.2	5.73e-03 ± 3.15e-03	8.46e-05 ± 3.15e-06
Qwen 32B	98.8	100.0 ± 0.0	1.2 ± 0.1	3.19e-03 ± 1.39e-04	1.27e-04 ± 5.23e-06
QwQ 32B	93.0	100.0 ± 0.0	0.4 ± 0.1	3.20e-03 ± 1.20e-04	1.07e-04 ± 3.84e-06
gpt-5.4-mini	100.0	100.0 ± 0.0	3.9 ± 0.2	3.19e-03 ± 1.26e-04	9.78e-05 ± 3.84e-06
gpt-5.4-mini + reasoning	100.0	100.0 ± 0.0	4.6 ± 0.2	1.13e-03 ± 3.45e-04	5.56e-05 ± 1.47e-05
gpt-5.4 + reasoning	100.0	99.9 ± 0.0	5.3 ± 0.2	2.48e-03 ± 1.52e-04	7.31e-05 ± 3.94e-06
Case 30					
Llama 8B	100.0	100.0 ± 0.0	0.2 ± 0.0	1.60e-03 ± 5.83e-05	4.68e-05 ± 1.41e-06
Qwen 8B	91.2	100.0 ± 0.0	0.2 ± 0.0	1.51e-03 ± 5.28e-05	4.64e-05 ± 1.44e-06
DeepSeek-Llama 8B	14.8	99.7 ± 0.3	10.6 ± 0.8	1.34e-02 ± 4.56e-03	2.65e-04 ± 8.24e-05
DeepSeek-Qwen 8B	78.3	99.4 ± 0.1	11.7 ± 0.4	7.06e+00 ± 5.04e+00	4.17e-01 ± 3.60e-01
gpt-5.4-nano	99.8	100.0 ± 0.0	3.2 ± 0.2	1.62e-03 ± 5.71e-05	3.70e-05 ± 1.17e-06
gpt-5.4-nano + reasoning	98.9	100.0 ± 0.0	14.4 ± 0.4	1.32e-03 ± 4.28e-05	3.79e-05 ± 1.09e-06
Qwen 32B	89.8	100.0 ± 0.0	0.1 ± 0.0	1.58e-03 ± 8.51e-05	6.75e-05 ± 4.87e-06
QwQ 32B	95.5	100.0 ± 0.0	0.2 ± 0.0	1.42e-03 ± 4.96e-05	4.56e-05 ± 1.40e-06
gpt-5.4-mini	99.8	100.0 ± 0.0	4.6 ± 0.2	1.76e-03 ± 6.64e-05	3.56e-05 ± 1.36e-06
gpt-5.4-mini + reasoning	100.0	100.0 ± 0.0	10.4 ± 0.4	1.03e-03 ± 8.32e-05	3.38e-05 ± 1.16e-06
gpt-5.4 + reasoning	100.0	100.0 ± 0.0	11.8 ± 0.3	1.77e-03 ± 6.40e-05	3.74e-05 ± 1.15e-06
Case 118					
Llama 8B	0.0	—	—	—	—
Qwen 8B	0.0	—	—	—	—
DeepSeek-Llama 8B	0.0	—	—	—	—
DeepSeek-Qwen 8B	15.3	99.9 ± 0.0	2.8 ± 0.3	2.56e-02 ± 1.16e-03	4.01e+00 ± 1.52e+00
gpt-5.4-nano	95.9	100.0 ± 0.0	3.8 ± 0.1	2.75e-02 ± 1.88e-03	6.42e-04 ± 1.99e-05
gpt-5.4-nano + reasoning	99.1	100.0 ± 0.0	7.8 ± 0.0	3.68e-02 ± 4.27e-03	7.42e-04 ± 2.22e-05
Qwen 32B	21.4	97.9 ± 0.3	0.0 ± 0.0	6.77e-02 ± 9.65e-03	1.41e-03 ± 8.72e-05
QwQ 32B	62.0	99.9 ± 0.0	0.0 ± 0.0	1.75e-01 ± 1.18e-01	5.80e-04 ± 1.99e-05
gpt-5.4-mini	99.9	100.0 ± 0.0	4.1 ± 0.1	3.16e-02 ± 7.34e-04	6.94e-04 ± 2.17e-05
gpt-5.4-mini + reasoning	99.8	100.0 ± 0.0	7.6 ± 0.0	3.55e-02 ± 8.21e-04	8.06e-04 ± 2.41e-05
gpt-5.4 + reasoning	98.4	99.8 ± 0.0	8.0 ± 0.1	1.42e-01 ± 2.30e-02	4.73e-04 ± 1.57e-05

B.4.2 Uncertainty and Calibration

Table 8: Calibration and scoring metrics on `case14`. Lower is better for ECE, Brier, and NLL.

Model	Mean Confidence ↓	ECE (20 bins) ↓	Brier ↓	NLL ↓
Llama 3.1 8B	0.6963	0.6963	0.4849	1.1921
Qwen3 8B	0.8277	0.8277	0.6891	1.9131
Qwen2.5 32B	0.6980	0.6980	0.4904	1.2155

Table 9: Calibration on Task 7-S (URL Phishing). Lower is better for ECE, Brier, and NLL.

Model	CTR	Mean Confidence	ECE (20 bins) ↓	Brier ↓	NLL ↓
Llama 3.1 8B	1.000	0.8067	0.1933	0.0382	0.2154
Qwen3 8B	1.000	0.8895	0.1105	0.0143	0.1184

B.4.3 Impact of Prompts

Table 10: Basic vs. enhanced prompt results for Task 6 (validity, constraint satisfaction, total MSE).

Case	Model	basic			enhanced		
		Valid (%)	CTR (%) ↑	MSE ↓	Valid (%)	CTR (%) ↑	MSE ↓
case14	llama3.1-8b	100.0	58.0 $\pm$ 0.0	4.80e-03	100.0	58.0 $\pm$ 0.0	4.92e-03
	qwen3-8b	99.8	58.8 $\pm$ 0.1	3.61e-03	99.6	59.0 $\pm$ 0.1	3.61e-03
	qwen2.5-32b	98.8	58.1 $\pm$ 0.0	3.32e-03	99.3	58.0 $\pm$ 0.0	4.58e-03
case30	llama3.1-8b	100.0	54.6 $\pm$ 0.0	1.65e-03	99.9	55.4 $\pm$ 0.0	1.57e-03
	qwen3-8b	91.2	54.7 $\pm$ 0.0	1.56e-03	98.5	60.4 $\pm$ 0.2	1.66e-03
	qwen2.5-32b	89.8	54.6 $\pm$ 0.0	1.65e-03	90.0	55.7 $\pm$ 0.1	2.05e-03

The prompt was enhanced using ChatGPT and resulted in the following:

```
You are a power grid operator performing Optimal Power Flow (OPF)
    simulations. Your task is to compute and return a valid JSON-
    formatted OPF solution based on the provided input JSON describing
    the grid topology and parameters.

---

## Input Format Description

The input JSON contains the following fields:

- **`bus`**: List of buses, each with attributes `[base_kv, bus_type,
    vm_min_pu, vm_max_pu]`
  - `bus_type`: 3 = slack/reference bus, 2 = generator bus (PV), 1 = load
     bus (PQ)
  - `vm_min_pu`, `vm_max_pu`: voltage magnitude bounds (hard constraints)
- **`load`**: List of loads, each with `[p_mw, q_mvar]` (fixed active and
     reactive power demand)
- **`generator`**: List of generators, each with `[sn_mva, p_mw, p_min_mw
    , p_max_mw, q_min_mvar (signed), q_min_mvar_bound, q_max_mvar,
    vm_pu_setpoint, ..., cost_coefficient, ...]`
  - Generator output must satisfy: `p_min_mw <= p_mw <= p_max_mw` and `
    q_min_mvar <= q_mvar <= q_max_mvar`
- **`('bus', 'ac_line', 'bus')`**: AC line connectivity via `edge_index`
    `[from_bus_list, to_bus_list]`
- **`('bus', 'transformer', 'bus')`**: Transformer connectivity via `
    edge_index`
```

```
- **`('generator', 'generator_link', 'bus')`**: Generator-to-bus connections
- **`('load', 'load_link', 'bus')`**: Load-to-bus connections
- **`('shunt', 'shunt_link', 'bus')`** *(if present)*: Shunt element connections

---

## Output Format Description

Return a JSON object with exactly two keys:

### `res_gen`
For each generator (indexed from `"0"`), provide:
- `p_mw`: Optimal active power output (MW), satisfying `p_min_mw <= p_mw <= p_max_mw`
- `q_mvar`: Optimal reactive power output (MVAr), satisfying `q_min_mvar <= q_mvar <= q_max_mvar`

### `res_bus`
For each bus (indexed from `"0"`), provide:
- `vm_pu`: Voltage magnitude in per unit, satisfying `vm_min_pu <= vm_pu <= vm_max_pu`
- `va_rad`: Voltage angle in radians (slack bus angle = 0.0)

---

## Critical Constraints to Satisfy

1. **Power balance**: At every bus, total generation = total load + net power flow out (Kirchhoff's laws must hold)
2. **Voltage bounds**: `vm_min_pu <= vm_pu <= vm_max_pu` for every bus, strictly enforced
3. **Generator active power bounds**: `p_min_mw <= p_mw <= p_max_mw`
4. **Generator reactive power bounds**: `q_min_mvar <= q_mvar <= q_max_mvar`
5. **Slack bus**: The bus with `bus_type = 3` has a fixed voltage angle of `va_rad = 0.0` and absorbs the system's active/reactive power imbalance
6. **PV buses**: Generator buses (`bus_type = 2`) hold their voltage magnitude at the generator's setpoint (`vm_pu_setpoint`), subject to voltage bounds
7. **Optimality**: Minimize total generation cost (weighted by cost coefficients); generators with zero cost are used preferentially or as needed
8. **Angle reference**: The slack bus defines `va_rad = 0.0`; all other angles are relative to this

---

Now, using the above instructions and the examples provided, compute and return the correct OPF solution JSON for the new input provided.
```

Table 11: Basic vs. enhanced prompt results for Tasks 1, 2, 3.

Case	Model	basic			enhanced		
		Task 1	Task 2	Task 3	Task 1	Task 2	Task 3
case14	llama3.1-8b	19.6	61.5	30.9	19.3	61.0	31.2
	qwen3-8b	30.8	72.9	55.6	30.1	73.4	56.0
case30	llama3.1-8b	16.0	33.0	9.1	16.1	32.4	9.1
	qwen3-8b	32.3	48.4	15.4	32.9	49.3	15.6

B.4.4 Impact of Fine-tuning

We further study the impact on fine-tuning on constraint satisfaction. Table 12 reports models scores after fine-tuning them.

SFT. To quantify how conventional alignment affects constraint satisfaction, we perform Supervised Fine-Tuning (SFT) on the base instruct models. Concretely, we train lightweight LoRA adapters (Hu et al., 2022) on the OPF training set, optimizing the model to reproduce ground-truth solutions in a zero-shot setting. By imitating high-quality demonstrations, the model learns both the required output formatting and the fundamental physical constraints implicit in the targets. *Hyperparameters:* LoRA $r = 8$, $\alpha = 16$, dropout 0.05, target modules: all-linear; 8-bit quantization; learning rate 2×10^{-4}; batch size 1 with 8 gradient-accumulation steps (effective batch size 8); 1 epoch.

GRPO. We train the reasoning-oriented variants with Group Relative Policy Optimization (GRPO) (Liu et al., 2024), a reinforcement-learning objective that encourages outputs with higher structured validity and feasibility. In contrast to standard RLHF, GRPO does not rely on a separate critic: for each prompt, it samples a group of candidate completions and computes policy gradients using each completion's reward relative to the group mean. This setup incentivizes the model to iteratively reason *through* constraints. The overall reward R is the sum of a structural and a constraint-satisfaction term:

$$R = r_{\text{json}} + r_{\text{CS}}.$$

Here, $r_{\text{json}} \in \{0, 0.5\}$ rewards compliance with the required JSON schema (0 for unparsable outputs and 0.5 for well-formed ones). The feasibility term $r_{\text{CS}} \in [0, 1]$ measures the proportion of evaluated constraints satisfied by the proposed solution, as defined in A.7. *Hyperparameters:* LoRA $r = 8$, $\alpha = 16$, dropout 0.05; group size (completions per prompt) 4; learning rate 2×10^{-5}.

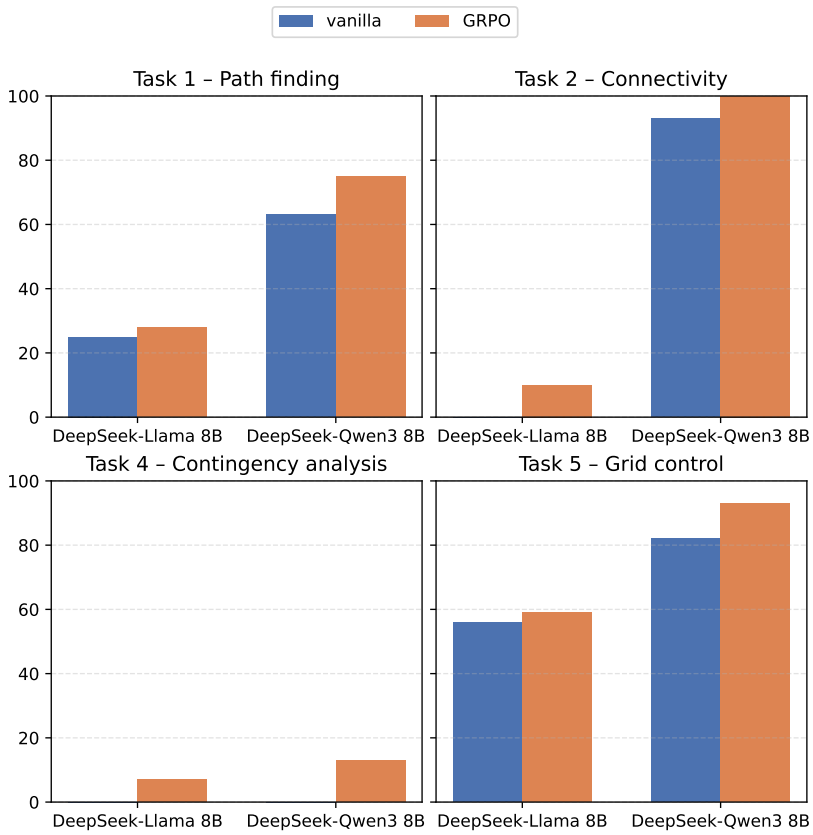


Figure 4: % of satisfied constraints by model and task, before and after GRPO tuning.

Table 12: OPF results by model and case (%). Reasoning models are optimized using GRPO, while the others are fine-tuned using SFT.

Model	Case	Scenario	Gen bounds CTR	Voltage bounds CTR	PF CTR	Total MSE
LLaMA 3.1-8B	14	Vanilla	100% ± 0	100% ± 0	1% ± 0.0822	0.0048 ± 0.000189
		SFT	100% ± 0	99.9% ± 0.0325	0.609% ± 0.0698	0.00391 ± 0.000137
	30	Vanilla	100% ± 0	100% ± 0	0.217% ± 0.0264	0.00165 ± 5.97e-05
		SFT	100% ± 0	98.8% ± 0.137	2.35% ± 0.107	0.00422 ± 0.000326
	118	Vanilla	–	–	–	–
		SFT	–	–	–	–
Qwen3-8B	14	Vanilla	100% ± 0	100% ± 0	2.77% ± 0.165	0.00361 ± 0.00014
		SFT	100% ± 0	100% ± 0	1.92% ± 0.106	0.00352 ± 0.000147
	30	Vanilla	100% ± 0	100% ± 0	0.241% ± 0.0313	0.00156 ± 5.43e-05
		SFT	100% ± 0	100% ± 0	4.22% ± 0.183	0.00143 ± 5.27e-05
	118	Vanilla	–	–	–	–
		SFT	100% ± 0	92.9% ± 0.844	1.77% ± 0.0767	0.0971 ± 0.0105
DS-R1-Qwen3-8B	14	Vanilla	100% ± 0	98.7% ± 0.198	7.5% ± 0.321	29.5 ± 27.1
		GRPO	100% ± 0.0303	99.6% ± 0.102	6.25% ± 0.304	29 ± 26
	30	Vanilla	100% ± 0	98.9% ± 0.245	11.7% ± 0.433	7.48 ± 5.4
		GRPO	100% ± 0.0273	99.5% ± 0.144	12.5% ± 0.455	0.19 ± 0.158
	118	Vanilla	100% ± 0	99.9% ± 0.0308	2.77% ± 0.264	4.04 ± 1.52
		GRPO	–	–	–	–
DS-R1-LLaMA-8B	14	Vanilla	99.9% ± 0.0693	98.4% ± 0.715	3.94% ± 0.417	0.0653 ± 0.0136
		GRPO	100% ± 0.0242	99.9% ± 0.112	1.37% ± 0.119	2.233 ± 3.95
	30	Vanilla	100% ± 0	99.3% ± 0.651	10.6% ± 0.827	0.0137 ± 0.00464
		GRPO	100% ± 0	99.6% ± 0.407	24.6% ± 0.611	0.00943 ± 0.00365
	118	Vanilla	–	–	–	–
		GRPO	–	–	–	–
Qwen2.5-32B	14	Vanilla	100% ± 0	100% ± 0.00723	1.19% ± 0.0858	0.00332 ± 0.000144
		SFT	100% ± 0	100% ± 0	1.41% ± 0.0954	0.00248 ± 0.000102
	30	Vanilla	100% ± 0	100% ± 0.0148	0.134% ± 0.0224	0.00165 ± 8.99e-05
		SFT	100% ± 0	100% ± 0	2.51% ± 0.0959	0.000837 ± 3.15e-05
	118	Vanilla	99.8% ± 0.131	96% ± 0.621	0.0158% ± 0.00785	0.0692 ± 0.00973
		SFT	93.5% ± 0.503	93.9% ± 0.887	0.0072% ± 0.00414	1.68 ± 0.268

Model	Case	Scenario	Gen. limits	Voltage limits	PF
Deepseek-R1 Qwen3-8B	14	Vanilla N	100%	98.7%	7.5%
		GRPO N	100%	99.6%	6.25%
		GRPO $N-1$	100%	98.6%	3.19%
	30	Vanilla N	100%	98.9%	11.7%
		GRPO N	100%	99.5%	12.5%
		GRPO $N-1$	99.8%	97%	9.71%
Deepseek-R1 Llama3.1-8B	14	Vanilla N	99.9%	98.4%	3.94%
		GRPO N	100%	99.9%	1.37%
		GRPO $N-1$	99.9%	99.8%	0.83%
	30	Vanilla N	100%	99.3%	10.6%
		GRPO N	100%	99.6%	24.6%
		GRPO $N-1$	100%	100%	17.8%

Table 13: % of satisfied constraints by model, case, and scenario.

B.4.5 IMPACT OF TOOLS

We designed a coding task to assess whether the Large Language Models (LLMs) can use tools (like external solvers). We allowed the models to build small grids and call a solver of their choice. The specific prompt provided to the models was:

> *"Write a Python script using PyPSA/Pyomo/JuMP to solve an OPF for a 3-bus system. Bus 1 has Gen 1 (cost $21/MWh), Bus 2 has Gen 2 (cost $32/MWh), and Bus 3 has a load of 132 MW. The line limit between Bus 1 and Bus 2 is 62 MW. All other lines have 500 MW capacity. Buses are connected in a triangle (1-2, 2-3, 3-1). Print the objective value."*

The simplest Python implementation used for verification is:

```
import pypsa
n = pypsa.Network()
for i in range(1, 4):
    n.add("Bus", f"Bus {i}")
n.add("Line", "L1-2", bus0="Bus 1", bus1="Bus 2", x=0.1, s_nom={
    line_limit})
n.add("Line", "L2-3", bus0="Bus 2", bus1="Bus 3", x=0.1, s_nom=500)
n.add("Line", "L3-1", bus0="Bus 3", bus1="Bus 1", x=0.1, s_nom=500)
n.add("Generator", "Gen 1", bus="Bus 1", p_nom=1000, marginal_cost={
    gen1_cost})
n.add("Generator", "Gen 2", bus="Bus 2", p_nom=1000, marginal_cost={
    gen2_cost})
n.add("Load", "Load 3", bus="Bus 3", p_set={load_val})
n.optimize(solver_name='highs')
print(n.objective)
```

We run the code generated by the LLM and the ground truth code, then extract and compare the resulting objective values (the final print output). For clarity, we report the performance as a **success rate**. The LLM is considered successful when the Mean Absolute Error (MAE) between the LLM solution and the ground truth is lower than 1×10^{-2}. Otherwise, we follow the same protocol as the main study.

We restricted our study to a 3-bus triangular grid and an IEEE 14-bus grid. *Sample size:* 20 generated instances total across case types; the 3-bus and 14-bus success rates above are computed over 7 and 6 instances respectively.

Table 14: LLM Performance Comparison (Success Rate)

LLM	Qwen-8B	Qwen-32B	GPT5.4 High Reasoning
3-bus	14.28%	100%	100%
14-bus	0%	16.67%	100%

B.4.6 Comparison with Baseline GNNs

Table 15: Comparison of Model Performance across Different Cases

Case	Model	Constraints %	MSE
case14	GNN	54.55%	2.76×10^{-3}
	GPT5.4 + Reasoning	57.8%	2.56×10^{-3}
case30	GNN	48.48%	1.04×10^{-3}
	GPT5.4 + Reasoning	55.2%	1.81×10^{-3}
case118	GNN	47.24%	3.57×10^{-2}
	GPT5.4 + Reasoning	59.8%	1.42×10^{-1}

GNN baseline details: graph autoencoder (encoder-decoder over bus/generator/load node features and line connectivity), hidden dimension 16, latent dimension 32; trained for 60 epochs (learning rate 10^{-4}, batch size 128) on the OPFDataset training split, jointly on case14 and case118.

Our results confirm that even for the simplest grid of our benchmark (IEEE14), and with access to solvers, average size LLMs still cannot perform sufficiently well. We believe however that when provided with access to python tools, the tasks assessed becomes about programming skills, given the large available of tutorials and coding examples to use the solvers, and not reasoning and optimization skills, that we focus on.

B.5 In-Context Tabular Reasoning and Optimization

Table 16: Q3: Tabular Reasoning and Optimization.

Model	Phishing		Credit Scoring		Botnet	
	Validity	CTR	Validity	CTR	Validity	CTR
Llama 8B	100.0	93.7	99.9	0.0	100.0	55.6
Qwen 8B	100.0	100.0	100.0	0.21	100.0	90.5
DeepSeek-Llama 8B	44.9	39.4	32.5	0.0	0.0	0.0
DeepSeek-Qwen 8B	100.0	100.0	97.7	0.64	72.7	55.3
gpt-5.4-nano	100.0	99.2	100.0	0.43	0.33	0.33
gpt-5.4-nano + Think	100.0	98.4	100.0	11.2	8.55	8.55
Qwen 32B	100.0	100.0	100.0	0.27	97.0	54.3
QwQ 32B	83.5	82.7	88.0	3.20	58.2	53.3
gpt-5.4-mini	100.0	99.2	100.0	9.59	70.7	70.1
gpt-5.4-mini + Think	100.0	98.4	100.0	20.7	69.7	69.1
gpt-5.4 + Think	100.0	98.4	100.0	85.3	26.3	26.3

Table 17: Zero-shot tabular constraint reasoning (Tasks 8-10), Accuracy (%). Per task the Small/Medium/Large variants correspond to the URL Phishing (URL), Credit Scoring (LCLD) and Botnet (CTU) use cases.

	Task 8: Discovery			Task 9: Choice			Task 10: Expression		
Model	URL	LCLD	CTU	URL	LCLD	CTU	URL	LCLD	CTU
Llama 8B	6.0	20.4	27.8	30.8	47.5	70.6	1.1	4.3	21.2
Qwen 8B	33.0	72.7	53.3	56.9	84.8	75.6	6.2	3.5	48.8
DeepSeek-Llama 8B	0.0	0.0	0.0	–	2.0	3.4	0.0	0.0	0.0
DeepSeek-Qwen 8B	85.1	76.5	81.9	94.2	94.8	98.3	21.8	49.9	42.9
gpt-5.4-nano	44.7	64.1	66.2	50.6	79.9	83.7	10.2	14.0	61.4
gpt-5.4-nano + Think	99.0	98.7	99.5	100.0	99.8	100.0	6.5	54.1	26.7
Qwen 32B	71.5	87.6	74.8	67.9	98.1	92.5	4.1	72.7	47.3
QwQ 32B	99.1	97.7	100.0	94.1	95.5	99.1	19.9	87.3	70.0
gpt-5.4-mini	53.6	70.0	76.8	82.6	92.9	94.2	13.9	47.8	28.6
gpt-5.4-mini + Think	99.4	95.3	99.2	99.6	99.9	99.7	4.8	41.2	28.4
gpt-5.4 + Think	99.6	92.4	100.0	100.0	100.0	100.0	14.6	57.5	76.8

B.6 ORC-BENCH API

The benchmark API is fully accessible from the code supplementary material.

Running tasks. One should either set the `OPENAI_API_KEY` environment variable to use `gpt` models (by their OpenAI API names), or provide a port to a locally running ollama server.

Then, run inference on models as easily as:

```python
from orcbench import run_tasks

run_tasks(
    model_id="llama3.1:8b",
    port=11434,  # ollama server port
    tasks=["1", "2", "6S", "6M", "7S", "7L"],
    results_dir="results"
)
```

Evaluation. Once results of inferences are stored in the results directory, one can compute their score with:

```python
from orcbench import eval_tasks

scores = eval_tasks(
    tasks=["1", "2", "6S", "6M", "7S", "7L"],
    results_dir="results"
)

print(scores)
```

B.7 SETUP DETAILS

Experiments were run on Intel(R) Xeon(R) Silver 4416+ @2.00GHz with 4 NVIDIA L40S GPU (48GB), and 503GB RAM. OpenAI calls cost $218.07 for Tasks 1-5 (Zero shot) and $346.16 for Tasks 6-7 (In context Learning).