QuantumQUBO Agent: Automating Quadratic Unconstrained Binary Optimization (QUBO) Formulation Generation from Natural Language

Niloy Kumar Mondal [1] **Md Rizwan Parvez** [2]

Abstract

Quadratic Unconstrained Binary Optimization (QUBO) is a central formulation for combinatorial optimization and has gained increasing attention due to its compatibility with quantum, hybrid quantum-classical, and quantum-inspired solvers. However, translating natural-language problem descriptions into correct QUBO formulations remains difficult, requiring the identification of binary variables, constraints, objective functions, penalty terms, and suitable penalty weights. This process is time-consuming and often demands substantial domain expertise. To address this challenge, we propose an end-to-end multi-agent framework that automatically generates QUBO formulations from natural-language problem descriptions, supported by structured or unstructured test cases. To evaluate its performance, We also introduce QUBOBench, a benchmark containing 100 combinatorial optimization problems across 12 application domains, curated from peer-reviewed literature, competitions, and canonical NP-hard problems. Experimental results show that our framework achieves 68% accuracy on QUBOBench, outperforming a direct single-call baseline by 22%. Further analysis identifies iterative self-repair as the most important component contributing to improved performance. The data and code are open-sourced at https://quitttcat.github.io/QuantumQUBOAgent/.

[1]Bangladesh University of Engineering and Technology [2]Qatar Computing Research Institute, HBKU, Doha, Qatar. Correspondence to: Niloy Kumar Mondal <nkm2105044@gmail.com>, Md Rizwan Parvez <mparvez@hbku.edu.qa>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

The Quadratic Unconstrained Binary Optimization (QUBO) problem is a combinatorial optimization problem classified as NP-hard, meaning no efficient algorithm is known to solve it optimally in polynomial time. It has a variety of application areas such as Computer Science, Finance, Physics, Engineering, Medicine, VLSI design, Quantum Computing, Traffic Management, Machine Scheduling, and many more (Punnen, 2022). With advances in quantum computing (Preskill, 2018), QUBO problems have become increasingly important because they can be addressed by quantum, hybrid quantum-classical, and quantum-inspired methods such as quantum annealing (Rajak et al., 2023) and the Quantum Approximate Optimization Algorithm (QAOA) (Farhi & Harrow, 2016). Quantum annealing is a powerful heuristic for solving optimization problems (Somma et al., 2012).

Researchers across many domains are exploring quantum optimization algorithms with the goal of potential speedups and improved accuracy. These algorithms have been applied to machine learning tasks such as feature selection (Mücke et al., 2023; Samanta et al., 2025; Pathak et al., 2025) and Bayesian network structure learning (Shikuri, 2025), computer vision problems including image segmentation and denoising (Heidari et al., 2024; Kerger & Miyazaki, 2023; Venkatesh et al., 2024), supply chain (Moncayo-Martínez & He, 2025; Ciacco et al., 2026), drug discovery (Yang, 2025), RNA folding (Fox et al., 2022), satellite observation (Ferrari et al., 2025), scheduling problems (Zhou et al., 2025; Gebler et al., 2024), cybersecurity (Marini & Arpe, 2025), IoT (Zhang et al., 2025b), and computational chemistry (Negre et al., 2022).

To apply quantum algorithms across domains, optimization problems must first be reformulated as QUBO and often as Ising formulations (Lucas, 2014). In general, given a natural-language description of a combinatorial optimization problem, the task is to find the values of binary decision variables $x \in \{0,1\}^n$ that minimize a cost function of the form

$$\min_{x\in\{0,1\}^n} E(x) = x^\top Q\, x + c, \tag{1}$$

where Q is a matrix of coefficients and c is a constant offset(Glover et al., 2019). The solution space contains 2^n

candidate bitstrings, and finding the optimum is NP-hard in general.

The cognitive task therefore involves identifying the binary decision variables, extracting the implicit and explicit constraints, designing the cost function and penalty terms for constraint violations, selecting appropriate penalty weights, and finally solving the resulting QUBO using classical heuristics or state-of-the-art quantum solvers. These steps are extremely challenging and time-consuming, as such a complex task often requires domain expertise, mathematical rigor, and experimental feedback.

To address this cognitive bottleneck, we propose an end-to-end multi-agent framework that generates the QUBO formulation from natural language problem description, together with either structured or unstructured test cases.To evaluate our framework, we introduce QUBOBench, a 100 combinatorial optimization problems 12 application domains, curated from peer-reviewed venues, competitions, and canonical NP-hard problems. Our framework achieves 68% accuracy on QUBOBench, outperforming the direct single-call baseline (46%) by a 22 percentage margin, with iterative self repair identified as the most critical component driving performance.

2. Related Work

Recent work has studied how large language models can translate natural language problem descriptions into formal optimization models. The NL4Opt competition established this direction by asking systems to formulate optimization problems from descriptions in natural language (Ramamonjison et al., 2023). Subsequent systems expand this idea with more structured reasoning and agentic workflows. Chain-of-Experts decomposes complex operations-research problems across multiple expert-style LLMs agents (Xiao et al., 2024), while Optimus uses LLMs to generate scalable mixed-integer and linear programming models for downstream solvers (AhmadiTeshnizi et al., 2024). OptimAI and LLMOPT further explore LLM-powered agents that define, model, and solve general optimization problems from natural language (Thind et al., 2025; Shu et al., 2025).

Prior LLM-based approaches to optimization modeling primarily focus on LP(Linear Programming), MILP (Mixed-Integer Linear Programming), or general solver-ready formulations, whereas the QUBO setting remains comparatively less explored despite its central role in quantum and quantum-inspired optimization. However, AutoQUBO automates QUBO generation from formal programmatic inputs, but still assumes that the user can provide a precise computational description of the problem and does not address formulation from natural-language research descriptions (Moraglio et al., 2022).

LLM-QUBO is the most closely related work which studies the automatic conversion of natural-language optimization descriptions into QUBO formulation. However, its experiments focus mainly on MILP to QUBO conversion and validate correctness through structural checks such as input-format adherence, binarization, slack-variable introduction, and penalty-term validity (Zhang et al., 2025a). Thus, it provides limited support for end-to-end validation of the generated formulation as an executable research artifact. Besides, it does not systematically verify that the natural-language specification, structured formulation, generated implementation, and small-instance optima are mutually consistent, nor does it provide a feedback-driven reformulation process when errors occur. In addition, the empirical evaluation is confined to a small number of canonical textbook problems and does not assess performance across diverse, domain-specific optimization tasks.

3. Methodology

Generating a QUBO formulation from a natural-language problem statement involves several distinct steps: understanding the problem, choosing binary encoding variables and constraints, defining penalty terms, deriving the cost function, generating runnable code, and verifying correctness. Rather than asking a single LLM to handle all of these steps at once, we decompose the task across specialized agents, each responsible for one step. The overall architecture is illustrated in Figure 1. The role of each agent and the end-to-end workflow are described in the following subsections.

3.1. Multi-Agent Architecture

Our framework consists of six LLM agents: Planner Agent, Formulizer Agent, Coder Agent, Debugger Agent, Writer Agent and a Judge Agent.

3.1.1. The Planner Agent

Our first agent is the Planner Agent. It is an LLM agent that reads the natural-language problem description provided by the user and extracts the decision variables, the optimization direction, and every implicit or explicit constraint together with its type (equality, $\leq$, or $\geq$) and hardness. It then returns a structured JSON representation of the problem, which is passed to the Judge Agent for consistency checking against the original description. If the Judge identifies any inconsistency, it returns a list of issues to the Planner, forming a feedback loop. The Planner then performs another round of parsing and planning from the original description guided by the issue list.

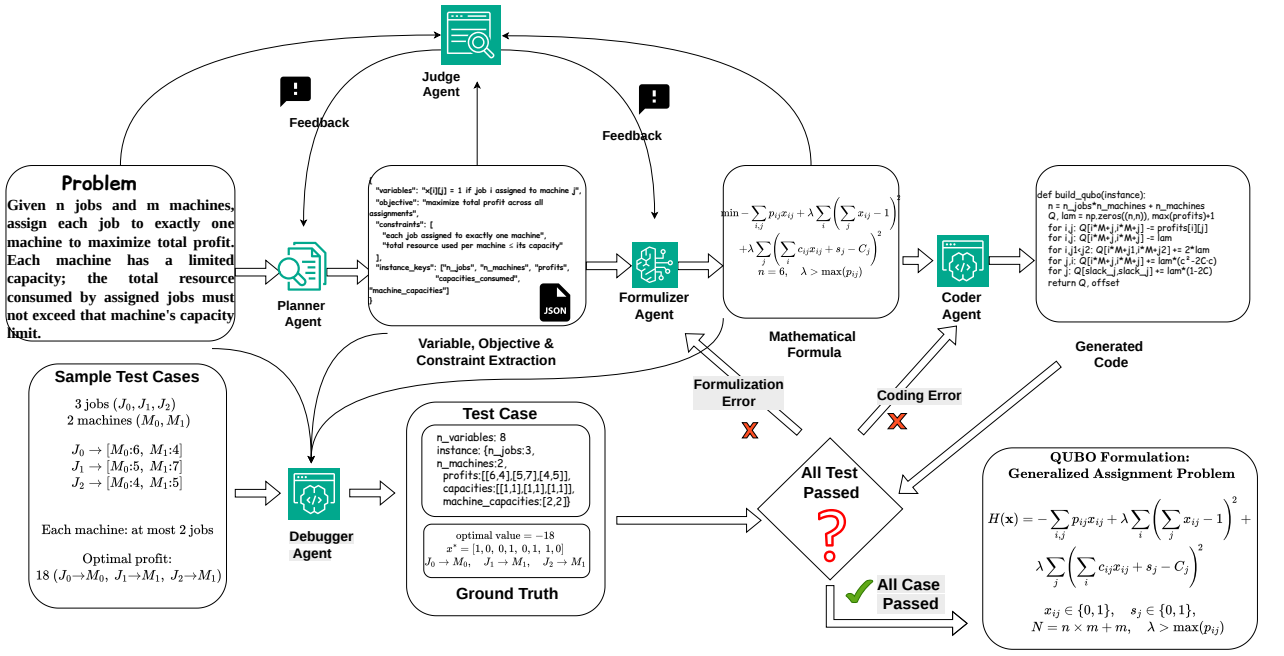


Figure 1. Workflow and Architecture of QuantumQUBO Agent

3.1.2. The Formulizer Agent

Our second agent is the Formulizer Agent. The Formulizer Agent is an LLM agent that takes the structured JSON representation from the Planner and derives the corresponding QUBO cost function from the decision variables, constraints, and the type of optimization, and simplifies the cost function. The returned formulation, in structured JSON containing the variable mapping, objective and penalty terms, penalty-weight justification, and constant offset, is then passed to the Judge Agent; if rejected, the Formulizer regenerates the formulation guided by the reported issues. The Formulizer takes feedback from the Judge Agent as well as from failed test cases at the end of the pipeline, and tries to improve the formulation.

3.1.3. The Judge Agent

Our next agent is the Judge Agent. It is an LLM agent invoked at two checkpoints: it first checks the Planner's structured specification for consistency with the original description, and later checks the Formulizer's QUBO cost for consistency with the specification judging correctness constraint encodings, penalty forms, and objective sign. In both roles it returns a pass/fail verdict with a list of specific issues, which are looped back to the originating agent as feedback for a targeted retry.

3.1.4. The Debugger Agent

Our next agent is the Debugger Agent. It is an LLM agent that is essentially a parser. It converts the user supplied test cases and ground truth from natural language into structured JSON records suitable for automated verification. It uses the variable mapping passed from the structured JSON formulation to encode the ground optimal selections as bitstrings. The Debugger never sees the Coder's output or the generated cost encoding matrix, which ensures optima from the generated formula is never stored as ground truth.Besides, the coder never sees the ground truth as well.

3.1.5. The Coder Agent

The Coder Agent is an LLM agent that translates the QUBO formulation into executable Python code. It is given the structured specification,formulation, and the parameters of a single sample instance but not the ground-truth bitstrings which prevents information leak. The generated function is then handed to the Test Runner for empirical verification, and any failure message is returned to the Coder as feedback for the next attempt.

3.1.6. The Test Runner

The Test Runner is a deterministic component (not an LLM agent) that validates the Coder's output against the ground truth on each test case. It classifies any failure as either a *coding error* (an exception or a malformed function) or a *formulation error* (a valid output whose optimum disagrees

with the ground truth). Coding errors are routed back to the Coder Agent, while formulation errors are routed back to the Formulizer Agent for a fresh derivation guided by the reported issues.

3.1.7. The Writer Agent

Our final agent is the writer agent that produces a visual representation and summary of the output.

4. QUBOBench

To evaluate our framework on domain specific problems, we introduce QUBOBench, a benchmark comprising 100 combinatorial optimization problems and 200 small instance test cases spanning 12 application domains. The problem instances are curated from three complementary sources: 76 drawn from peer-reviewed venues, 5 from competitions and hackathons, and 19 canonical NP-hard problems primarily adapted from Lucas (Lucas, 2014). A data sample consists of hand curated problem description and sample small instance test cases both in natural language. For more details, refer to Appendix B.

5. Experimental Setup

5.1. Evaluation Protocal

We evaluate QuantumQUBO Agent on the QUBOBench dataset. Performance is measured across two metrics: *accuracy*, defined as the fraction of benchmarks for which the pipeline produces a QUBO formulation and matches all ground-truth optimal solutions; and *total token consumption*.

5.2. Baseline Metric

As a lower-bound baseline, we conduct a direct evaluation in which the natural-language problem description is presented directly to the LLM with a single prompt. To ensure fair and fully automated evaluation, we provide the model with structured instance dictionaries previously generated by the Debugger agent, constraining the output to a fixed function signature. This eliminates the possibility of format-induced false negatives in automated scoring. Ground-truth optimal solutions are never exposed to the model at any stage.

5.3. Model Selection

All reasoning agents (Planner, Judge, Formulizer, Debugger) use `qwen/qwen3-235b-a22b`, while the Coder agent uses `qwen/qwen3-coder-next`. We believe our pipeline is LLM-agnostic. For the direct evaluation we used `qwen/qwen3-coder-next`.

5.4. Ablation Study

We conduct two ablation studies: removing the iterative feedback once and removing the judge together with feedback, to isolate the contribution.

6. Results and Discussions

Planner	Judge	Formulizer	Coder	Iter. Feedback	Acc. (%)	Tokens (k)
✓	✓	✓	✓	✓	**68**	48.1
✓	✓	✓	✓	×	35	24.7
✓	×	✓	✓	×	36	22.1
×	×	×	×	×	46	3.2

Table 1. Ablation results on QUBOBench. Rows from top: QuantumQUBO Agent, without iterative feedback, without Judge and iterative feedback, Direct Method.

Table 1 shows that the full pipeline achieves 68% accuracy, outperforming all ablated variants. Removing iterative feedback causes the largest drop to 35%, confirming that self-repair is the most critical component. Removing the Judge while keeping all other components fixed yields 36%, nearly identical to the no-retry setting, indicating that the Judge provides value only when paired with a sufficient retry budget to act on its feedback.

6.1. Failure mode analysis

A *formulation error* occurs when the agent produces an incorrect QUBO objective, wrong penalty terms, incorrect variable mapping, or an energy landscape that does not encode the problem correctly. A *coding error* occurs when the QUBO mathematics is correct but the `build_qubo` implementation fails due to index bugs, wrong array shapes, or incorrect NumPy operations.

Method	Code Error (%)	Form. Error (%)	Other (%)
QuantumQUBO Agent	79.3	15.5	5.2
Direct Method	26.4	66.0	7.5

Table 2. Error type distribution on QUBOBench. Percentages are of total errors by QuantumQUBO Agent vs direct method.

In the QuantumQUBO Agent, 79.3% of all errors originate from the Coder agent, while only 15.5% from formulation failures, indicating that the Planner-Formulizer-Judge chain successfully resolves most mathematical ambiguities. In contrast, the direct method shows the inverse pattern: 66.0% of failures are formulation errors, confirming that without structured reasoning, a single LLM call struggles primarily with deriving correct QUBO mathematics rather than implementing it.

7. Future Work

Future work includes fine-tuning separate agents for specific tasks, adding Ising Formulator incorporated with quantum hardware feedback.

References

AhmadiTeshnizi, A., Gao, W., and Udell, M. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024.

Ciacco, A., Guerriero, F., and Saccomanno, F. P. Quantum annealing for the two-level facility location problem. *Future Generation Computer Systems*, 174:107961, 2026.

Farhi, E. and Harrow, A. W. Quantum supremacy through the quantum approximate optimization algorithm. *arXiv preprint arXiv:1602.07674*, 2016.

Ferrari, B., Ganzerli, G., Iori, M., and Orosei, R. A qubo formulation for scheduling satellite observations of mars. In *2025 IEEE International Conference on Quantum Artificial Intelligence (QAI)*, pp. 228–233. IEEE, 2025.

Fox, D. M., MacDermaid, C. M., Schreij, A. M., Zwierzyna, M., and Walker, R. C. Rna folding using quantum computers. *PLOS Computational Biology*, 18(4):e1010032, 2022.

Gebler, C., Rethmann, J., and Ueberholz, P. Qubo models for the fifo stack-up problem and experimental evaluation on a quantum annealer. *SN Computer Science*, 5(7):818, 2024.

Glover, F., Kochenberger, G., and Du, Y. Quantum bridge analytics i: a tutorial on formulating and using qubo models. *4or*, 17(4):335–371, 2019.

Heidari, S., Dinneen, M. J., and Delmas, P. Quantum annealing for computer vision minimization problems. *Future Generation Computer Systems*, 160:54–64, 2024.

Kerger, P. and Miyazaki, R. Quantum image denoising: a framework via boltzmann machines. *QUBO, and Quantum annealing*, 2023.

Lucas, A. Ising formulations of many np problems. *Frontiers in physics*, 2:74887, 2014.

Marini, R. and Arpe, R. Cyber risk scoring with qubo: A quantum and hybrid benchmark study. *arXiv preprint arXiv:2512.18305*, 2025.

Moncayo-Martínez, L. A. and He, N. Quantum optimisation for supply chain: Qubo formulations and qaoa solutions for facility location and load balancing. *Results in Engineering*, pp. 108373, 2025.

Moraglio, A., Georgescu, S., and Sadowski, P. Autoqubo: Data-driven automatic qubo generation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 2232–2239, 2022.

Mücke, S., Heese, R., Müller, S., Wolter, M., and Piatkowski, N. Feature selection on quantum computers. *Quantum Machine Intelligence*, 5(1):11, 2023.

Negre, C. F., Lopez-Bezanilla, A., Zhang, Y., Akrobotu, P. D., Mniszewski, S. M., Tretiak, S., and Dub, P. A. Toward a qubo-based density matrix electronic structure method. *Journal of chemical theory and computation*, 18 (7):4177–4185, 2022.

Pathak, I. P., Kim, T., Yilmazer, N., and Roach, D. Feature selection for imbalanced network intrusion detection systems dataset using quantum annealer. In *2025 IEEE International Conference on Electro Information Technology (eIT)*, pp. 1–6. IEEE, 2025.

Preskill, J. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. ISSN 2521-327X. doi: 10.22331/q-2018-08-06-79. URL https://doi.org/10.22331/q-2018-08-06-79.

Punnen, A. P. The quadratic unconstrained binary optimization problem. *Springer International Publishing*, 10: 978–3, 2022.

Rajak, A., Suzuki, S., Dutta, A., and Chakrabarti, B. K. Quantum annealing: An overview. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2241), 2023.

Ramamonjison, R., Yu, T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pp. 189–203. PMLR, 2023.

Samanta, S., Wang, D., Hodges, T., and Ferenczi, A. Parametric quantum feature selection methods for fraud and default detection. In *Proceedings of the AAAI Symposium Series*, volume 7, pp. 390–397, 2025.

Shikuri, Y. Decomposed quadratization: efficient qubo formulation for learning bayesian network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 11345–11352, 2025.

Shu, X., Qian, H., Lu, X., ZHOU, J., Zhou, A., Yu, Y., et al. Llmopt: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, volume 2025, pp. 101580–101606, 2025.

Somma, R. D., Nagaj, D., and Kieferová, M. Quantum speedup by quantum annealing. *Physical review letters*, 109(5):050501, 2012.

Thind, R., Sun, Y., Liang, L., and Yang, H. Optimai: Optimization from natural language using llm-powered ai agents. *arXiv preprint arXiv:2504.16918*, 2025.

Venkatesh, S. M., Macaluso, A., Nuske, M., Klusch, M., and Dengel, A. Q-seg: Quantum annealing-based unsupervised image segmentation. *IEEE Computer Graphics and Applications*, 44(5):27–39, 2024.

Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., et al. Chain-of-experts: When llms meet complex operations research problems. In *International Conference on Learning Representations*, volume 2024, pp. 48519–48537, 2024.

Yang, P.-K. Ligand pose generation via qubo-based hotspot sampling and geometric triplet matching. *arXiv preprint arXiv:2507.20304*, 2025.

Zhang, H., Emu, M., and Choudhury, S. Llm-qubo: An end-to-end framework for automated qubo transformation from natural language problem descriptions. In *Proceedings of the AAAI Symposium Series*, volume 7, pp. 411–418, 2025a.

Zhang, W., Zhou, C., Liu, W., Han, T., and Ansari, N. Hybrid quantum-inspired optimization for aigc-driven iot task offloading in mec networks. *IEEE Internet of Things Journal*, 2025b.

Zhou, Z., Chen, Q., Zhang, C., Tan, M., and Li, S. Hybrid quantum annealing for large-scale exam scheduling: Validation in real-world educational scenarios. *Applied Soft Computing*, pp. 113756, 2025.

A. Algorithm of QuantumQUBO Agent

Input: A natural-language problem description P, a set of test cases $\mathcal{T}$, retry budgets $R_{\text{planner}}, R_{\text{formulizer}}, R_{\text{coder}}$ controlling the maximum number of repair attempts at each stage, and a pass threshold $\tau \in (0, 1]$ defining the minimum fraction of test cases that must pass for a solution to be accepted.

Output: A verified QUBO formulation $\mathcal{F}$, a correctness-verified builder function f that constructs the QUBO matrix for any problem instance, a worked example, and a LaTeX write-up with compiled PDF deliverable.

Algorithm 1 QuantumQUBO Agent Pipeline

$R_{\text{planner}}, R_{\text{formulizer}}, R_{\text{coder}} \leftarrow$ number of maximum retry for each stage
$\tau \leftarrow$ minimum pass rate to accept a solution

$spec \leftarrow$ PlannerAgent(P)
$check \leftarrow$ JudgeAgent$(P,\ spec)$
for $i \leftarrow 1$ **to** R_{planner} **do**
  if not $check.passed$ **then**
    $spec \leftarrow$ PlannerAgent$(P,\ check.issues)$
    $check \leftarrow$ JudgeAgent$(P,\ spec)$
  end if
end for
if not $check.passed$ **then**
  return Fail
end if

$formulation \leftarrow$ FormulizationAgent$(spec)$
$check \leftarrow$ JudgeAgent$(spec,\ formulation)$
for $i \leftarrow 1$ **to** $R_{\text{formulizer}}$ **do**
  if not $check.passed$ **then**
    $formulation \leftarrow$ FormulizationAgent$(spec,\ check.issues)$
    $check \leftarrow$ JudgeAgent$(spec,\ formulation)$
  end if
end for
if not $check.passed$ **then**
  return Fail
end if

$test_cases \leftarrow$ DebuggerAgent$(spec,\ mapping(formulation),\ sample_input)$

for $i \leftarrow 1$ **to** R_{coder} **do**
  $code \leftarrow$ CoderAgent$(spec,\ formulation,\ error)$
  $results,\ class \leftarrow$ TestRunner$(code,\ test_cases)$
  $pass_rate \leftarrow results.passed\ /\ |test_cases|$
  if $pass_rate \geq \tau$ **then**
    $latex \leftarrow$ WriterAgent$(spec,\ formulation)$
    return $formulation,\ code,\ latex$
  end if
  if $class =$ *formulation_error* **then**
    $formulation \leftarrow$ FormulizationAgent$(spec,\ results.errors)$
    $check \leftarrow$ JudgeAgent$(spec,\ formulation)$
  end if
  $error \leftarrow results.errors$
end for
return Fail

B. Details of Dataset

B.1. Overview

QUBOBench is a benchmark designed to evaluate Large Language Models (LLMs) on the task of formulating Quadratic Unconstrained Binary Optimization (QUBO) problems from natural-language descriptions. The dataset comprises 100 problem instances and 200 test cases spanning 12 application domains, encompassing classical graph-theoretic and combinatorial problems (e.g., Max-Cut, Maximum Independent Set, Graph Coloring, Hamiltonian Path), logistics and scheduling (Vehicle Routing, Job Shop Scheduling, Bin Packing), machine learning tasks formulated as discrete optimization (feature selection, k-means and k-medoids clustering, Bayesian network structure learning), computational biology and chemistry (RNA folding, protein lattice folding, molecular conformation selection), finance (portfolio optimization, credit scorecard selection), computer vision (image segmentation, denoising, stereo matching), networks and infrastructure (sensor placement, distribution-network reconfiguration), and quantum computing itself (qubit allocation, circuit compilation, prime factorization).

B.2. Dataset Structure

Each datapoint in QUBOBench consists of two files:

- `prompt.txt` : a natural-language description of the optimization problem, expressed in the style and terminology of its source domain.
- `sample_cases.txt` : one or more natural-language test cases that specify a concrete problem instance, the optimal solution configuration, and the corresponding ground-truth objective value. Each test case explicitly cites its source.

This format preserves the natural-language character of the input, ensuring that the benchmark evaluates an LLM's ability to parse domain-specific terminology, identify decision variables and constraints, and derive a valid QUBO formulation without relying on pre-structured inputs.

B.3. Problem Sources

The 100 problem instances are curated from three complementary sources to balance rigor, diversity, and coverage.

(1) Peer-reviewed research papers (76 problems). The majority of instances are drawn from journal articles and conference papers with verifiable DOIs. Contributing venues include:

Journals — The Journal of Finance, *Management Science*, *Artificial Intelligence*, *PLOS Computational Biology*, *PLOS ONE*, *Nature Scientific Reports*, *npj Quantum Information*, *npj Unconventional Computing*, *Physical Review Research*, *Journal of Chemical Theory and Computation*, *Theoretical Computer Science*, *Quantum Machine Intelligence*, *Quantum Information Processing*, *Annals of Operations Research*, *Journal of Combinatorial Optimization*, *Journal of Optimization Theory and Applications*, *Applied Soft Computing*, *Future Generation Computer Systems*, *Neurocomputing*, *Electric Power Systems Research*, *Discrete Optimization*, *IEEE Transactions on Quantum Engineering*, *IEEE Internet of Things Journal*, *IEEE Computer Graphics and Applications*, *Entropy*, *Applied Sciences*, *Journal of Risk and Financial Management*, *Frontiers in Physics*, *Frontiers in Computer Science*, *Frontiers in ICT*, *Bioinformatics Advances*, *International Journal for Numerical Methods in Engineering*, *EPJ Quantum Technology*, *Results in Engineering*, *Array*, *SN Computer Science*, *Journal of Membrane Computing*, *IEICE Communications Express*, and *Journal of Information Processing*.

Conferences — AAAI, *ACM SIGKDD*, *ACM/SIGDA FPGA*, *IEEE BigData*, *IEEE GLOBECOM*, *IEEE QAI*, *IEEE QCE*, *IEEE EIT*, *ICCSA*, *JSME Robotics and Mechatronics*, and *IFAC*.

(2) Competitions and hackathons (5 problems). To capture practitioner-style framings and applied challenges, we include problems posed in:

- Fixstars Amplify Benchmark Suite
- Aqora / U.S. DOE Global Industry Challenge 2026
- iQuHACK 2025 D-Wave Hackathon

- AtCoder Educational DP Contest
- D-Wave Examples Repository

(3) Canonical NP-hard problems (19 problems). Classical decision and optimization problems including Max-Cut, Hamiltonian Cycle, Travelling Salesman, Subset Sum, and Vertex Cover which are drawn primarily from Lucas (Lucas, 2014), supplemented with hand-curated textbook instances. These problems serve as foundational test cases against which any general-purpose QUBO-formulation system should perform reliably.

All natural-language prompts and test cases were hand-curated by the authors to ensure clarity, consistency, and faithful representation of each source problem.

B.4. Domain-wise Distribution

Table 3 summarizes the per-domain distribution of problems and test cases. Graph theory dominates due to the centrality of graph-theoretic formulations in the QUBO literature, while domains such as Robotics and Electronics & VLSI remain comparatively sparse, reflecting the relative scarcity of published QUBO formulations in these areas.

Table 3. Per-domain distribution of problems and test cases in QUBOBench.

Domain	#Problems	#TC
Graph Theory	23	49
Scheduling	11	25
Machine Learning	11	22
Logistics & Transportation	12	21
Biology & Chemistry	9	19
Networks & Infrastructure	7	15
Combinatorics & Logic	8	11
Finance	5	10
Computer Vision	5	10
Quantum Computing	5	10
Robotics	2	4
Electronics & VLSI	2	4
Total	**100**	**200**

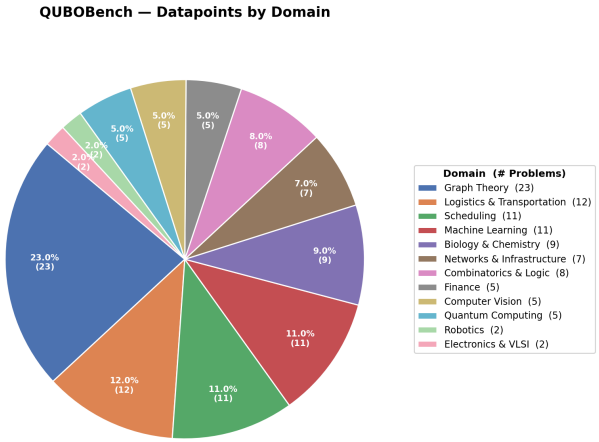


Figure 2. Per-domain distribution of problems in QUBOBench.