scientific reports

Check for updates

OPEN

Feedback-integrated prompt optimiser for problem formulation

Pivithuru Thejan Amarasinghe[1]✉, Su Nguyen[2], Yuan Sun[1] & Damminda Alahakoon[1]

Problem formulation is a critical yet expertise-intensive step in optimisation modelling. Automating this process offers a promising solution, but requires effective methods to replicate the reasoning and decision-making processes that are both crucial and typically performed by human experts. Recent advancements in Large Language Models (LLMs) have introduced the potential for human-like reasoning in this context, making optimisation more accessible, scalable, and intelligent. However, the substantial computational demands of LLMs limit their practicality in many real-world settings. Small Language Models (SLMs) present a more resource-efficient alternative but face challenges such as limited reasoning capabilities and high sensitivity to prompt structure, reducing their effectiveness in complex tasks like automated problem formulation. To address these limitations, we propose FIPO (Feedback-Integrated Prompt Optimiser), a novel approach designed to enhance the problem formulation capabilities of SLMs through iterative, feedback-driven prompt optimisation. FIPO integrates the local search algorithm with structured feedback generated by agentic workflows that simulate expert evaluations from problem formulation and programming perspectives. We evaluate FIPO on the LPWP dataset and observe consistent performance gains compared to existing state-of-the-art prompt optimisation methods. Our findings establish feedback-guided prompt evolution as a promising strategy for enabling cost-efficient, scalable, and accurate automated problem formulation with SLMs.

Keywords Problem formulation, Language models, Prompt optimisation

Optimisation is widely applied across various domains, including operations, economics, engineering, and computer science. Fundamentally, it involves identifying the best set of decisions that either minimise or maximise a specific objective, while satisfying a set of constraints that capture the problem's inherent limitations. These key components (i.e., objectives, decision variables, and constraints) form the foundation of any optimisation model. Real-world optimisation challenges encompass a variety of complex applications, such as production scheduling[1], vehicle routing[2], bin packing[3], and material waste reduction in cutting processes[4]. To solve such problems, a wide array of algorithmic techniques have been developed, including dynamic programming[5], branch and bound[6], random-restart hill climbing, simulated annealing, genetic algorithms, and tabu search[7]. Additionally, several highly efficient solvers such as Gurobi[8], Google OR-Tools[9], and CPLEX[10] provide robust, commercial-grade tools for solving optimisation problems.

A critical step in solving optimisation problems is the process of problem formulation, which involves translating high-level problem descriptions into precise mathematical models suitable for optimisation solvers. Despite the availability of powerful optimisation tools, constructing these models remains a time-consuming and expertise-driven process. It requires a deep understanding of the problem domain and mathematical modelling techniques, making it a significant barrier to adopting optimisation in practice. The automated construction of problem formulations from natural language descriptions is referred to as automated problem formulation[11]. Recently, Large Language Models (LLMs) have gained widespread attention for their ability to perform complex tasks and are now used in various industrial applications[12]. Notably, recent advances demonstrate that LLMs hold significant promise in automated problem formulation[13,14]. Leveraging techniques such as prompt engineering, fine-tuning, and agentic workflows, LLMs can effectively translate natural language problem descriptions into solvable optimisation models. However, these models are computationally expensive and often inaccessible in resource-constrained environments. As a result, there is growing interest in Small Language Models (SLMs), which offer a lightweight alternative with far fewer parameters. However, SLMs have inherent limitations such as being more sensitive to prompt structure, possessing weaker generalisation, and struggling with complex reasoning tasks such as automated problem formulation.

[1]Research Center for Data Analytics and Cognition, La Trobe University, Melbourne, VIC 3086, Australia. [2]College of Business and Law, RMIT University, Melbourne, VIC 3000, Australia. ✉email: p.amarasinghe@latrobe.edu.au

To address these limitations, we explore how prompt optimisation can enhance the problem formulation capabilities of SLMs. Prompt optimisation refers to the process of systematically refining the input prompts given to a language model to improve its performance on a specific task. This can involve modifying instructions, rephrasing questions, adding examples, or reorganising contextual information to elicit more accurate and structured outputs from the model. While existing prompt optimisation methods[15,16] have proven useful in general natural language processing tasks, they often fall short in complex tasks such as automated problem formulation, primarily because they lack mechanisms to incorporate evaluation-based feedback into the prompt optimisation process. In this research, we propose a novel prompt-optimisation approach for automated problem formulation with SLMs. In our approach, we apply feedback-based prompt optimisation using metaheuristic techniques. Specifically, we use local search to iteratively refine prompts based on their performance. To guide this process, we introduce prompt-based operators that leverage feedback from previously generated prompts. Hence, this approach focuses not only on performance outcomes but also on the types of errors made. The feedback is generated using agentic workflows, which simulate expert evaluations from multiple perspectives, such as problem formulation and programming. By combining these components, our proposed approach systematically explores the prompt space to identify high-performing, instruction-rich prompts tailored to the capabilities of SLMs. Although the approach is evaluated using SLMs, the underlying methodology is broadly applicable and can be extended to language models of any scale.

Our research makes the following key contributions:

- We develop a feedback-integrated prompt optimisation approach based on a local search algorithm to enhance automated problem formulation with SLMs.
- We propose novel prompt-based operators that explore the prompt search space using performance-based heuristics and feedback on past formulation errors.
- We conduct a comprehensive evaluation of our approach across various prompt engineering strategies such as Standard Prompt, and Chain-of-Thought (CoT) to demonstrate its effectiveness in improving SLM performance on problem formulation tasks.
- We benchmark our approach against state-of-the-art prompt optimisation methods, showing consistent performance gains in terms of accuracy, convergence, and prompt quality in the context of automated problem formulation.

Literature review

Automated problem formulation

The motivation behind automated problem formulation comes from the inherent complexity of constructing problem formulations, which typically requires specialised human expertise. The automated problem formulation has the potential to significantly reduce the time and cost involved, while making optimisation techniques accessible to users without deep optimisation knowledge. However, automated problem formulation remains a challenging task, as optimisation models are highly problem-specific, and errors during the construction of problem formulations can result in infeasible or suboptimal solutions. Recent research highlights the important role of large language models (LLMs) in enabling automated problem formulation. Various LLM-based approaches have emerged, leveraging techniques such as prompt engineering[13], in-context learning[14], fine-tuning[17], agentic workflows[18], and autonomous agents[19]. In parallel, benchmark datasets have been introduced to evaluate these methods across a range of optimisation techniques, including linear programming[20,21], mixed-integer programming[13,22], and constraint programming[23]. At the same time, the explainability and generalisability of developed solutions are critical[24], underscoring the need for benchmarks that go beyond technical performance to capture these broader dimensions.

Large language models

Large Language Models (LLMs) are advanced computational models with massive parameter counts, capable of understanding and generating human language[25]. They operate by predicting the probability of word (token) sequences based on a given input. LLMs follow a transformer-based architecture[26] and self-attention available in transformers, works as the fundamental principle for the success of language modelling tasks. GPT[27], Claude[28], Gemini[29], and LLaMA[30] are prominent series of large language models that are made out of deep neural networks with billions of parameters. These LLMs are trained on a large corpus of text data, and the underlying transformer architecture enables them to learn intricate linguistic patterns, grammar, context and semantics[31]. Recently, multimodal LLMs have been developed to process data types such as images and videos[32]. However, this research focuses specifically on text-generation LLMs. Given their powerful capabilities, LLMs are now used across a wide range of application domains[12]. Key techniques for leveraging LLMs in real-world systems include prompt engineering, in-context learning, fine-tuning, and agentic workflows.

In this research, we focus particularly on Small Language Models (SLMs), which require significantly fewer computational resources than Large Language Models (LLMs), yet demonstrate competitive performance across a range of language tasks with greater efficiency[33]. SLMs leverage lightweight architectures along with efficient training and post-processing techniques to accelerate both training and inference, while significantly reducing computational requirements. To maintain a lightweight architecture, SLMs typically utilise either the encoder or the decoder component of the Transformer, rather than the full encoder–decoder setup. During training, SLMs employ several techniques to enhance efficiency and stability. These include mixed-precision training[34], memory-efficient optimisers[35], gradient clipping to mitigate exploding gradients[36], and linear attention mechanisms to reduce computational overhead[37]. Additionally, careful initialisation strategies are adopted to ensure effective convergence. To further compress the model while preserving performance, SLMs apply

post-training techniques such as weight pruning[38], quantisation[39], and knowledge distillation[40], all of which contribute to reducing model size and complexity while maintaining the performance.

Prompt optimisation

A prompt is a natural language input provided to an LLM to obtain a desired output. The input contains context that activates relevant knowledge of the LLM. This input can be in various forms like questions, instructions or contextual information[41]. However, unlike humans, LLMs rely on statistical patterns rather than true semantic understanding. Therefore, prompts need to be optimised either manually or automatically. Hard prompts refer to prompts that incorporate artificially constructed sentences or phrases deliberately added before or after the original prompt. Hard prompts are portable as they can be discovered with one LLM and applied in another. Furthermore, hard prompts can be edited by hand to change their behaviour. However, discovering hard prompts is a specialised skill that involves trial and error[42]. Soft prompts tune the parameters of some input tokens. Unlike hard prompts, soft prompts are fed as continuous vectors. The discovery of soft prompts is a mathematical science. For instance, gradient-based methods can generate highly performant prompts for specialised tasks. But soft prompts require parameters of LLMs. Therefore, soft prompts are not feasible for LLMs accessed via black-box APIs. A hybrid prompt is a combination of hard and soft prompts. The purpose of a hybrid prompt is to combine multiple sources of information to generate richer and more diverse results[43].

Automatic prompt optimisation focuses on reflection and resampling as the two key strategies. Reflection-based strategies iteratively improve the prompts based on experience with past prompts. Resampling-based strategies focus on combining multiple prompts sampled from a set of existing prompts. Reference[44] introduce a reflection-based technique to optimise the behaviour of the LLM directly. They have evaluated their technique in object counting, navigation, snarks, and question selection tasks. Reference[45] use metaheuristics (Monte Carlo Search) in a black-box setting to generate instructions to optimise the prompt. They use execution accuracy and log probability to evaluate their technique in NLP tasks. Reference[46] use textual gradients with reflection strategies to criticise the current prompt similar to how numerical gradients point in the direction of error ascent. Reference[47] view prompt optimisation as a strategic planning problem and employ Monte Carlo tree search to navigate the expert-level prompt space strategically. Reference[48] instruct an LLM to deduce new hints for selected samples from incorrect predictions, and then summarise from per-sample hints and add the results back to the initial prompt to form a new, enriched instruction. Reference[16] start from a population of prompts and iteratively generate new prompts with LLMs based on the evolutionary operators. As the above-mentioned current techniques have been experimented with generic NLP tasks, there is an opportunity to expand these techniques to more complex tasks like mathematical reasoning and problem formulation for optimisation problems.

Agentic workflows

In this research, we leverage the capabilities of LLM-based agentic workflows. An agentic workflow can execute a task by following a predefined sequence of processes with multiple LLM invocations[49]. These agentic workflows can be generic or domain-specific. Compared to an autonomous agent[50], an agentic workflow has less flexibility in making autonomous decisions. However agentic workflows need less specific actions and environment-specific design patterns. This simplicity allows agentic workflows to be constructed based on human experience and they can be improved iteratively offering strong potential for task automation. An agentic workflow is typically composed of a set of prompts, hyperparameters, and an overall workflow design. Manually identifying these components can be time-consuming and labour-intensive. As a result, recent work has proposed automated methods to optimise these components efficiently[51].

Metaheuristics

Classic optimisation techniques make strong assumptions about the nature of the problems that need to be solved, which can limit their applicability to certain problem domains. Therefore, more versatile optimisation methods have been developed. Metaheuristics is a subfield of stochastic optimisation which makes weaker assumptions and sometimes no assumptions[7]. They provide high-level strategies designed to efficiently explore search spaces, aiming to identify optimal or near-optimal solutions. Unlike problem-specific heuristics, metaheuristics guide underlying heuristics without being tailored to a particular problem, enhancing their general applicability[52]. Although metaheuristics is an iterative process, it provides mechanisms to avoid getting trapped in local optimums. Metaheuristic algorithms are approximate and usually non-deterministic. Ant Colony Optimisation (ACO), Evolutionary Computation (EC) including Genetic Algorithms (GA), Iterated Local Search (ILS), Simulated Annealing (SA), and Tabu Search (TS) are some metaheuristic algorithms[53].

In this research, we focus on local search, which can serve as a stand-alone optimisation technique. The local search begins with some initial solutions and moves from neighbour to neighbour to improve the objective value[54]. The key elements of local search are picking the initial solution, defining neighbourhoods, and selecting a neighbour for a given solution. The local search has low empirical complexity and has wide applicability[55]. If a reasonable definition of neighbourhoods and an efficient way of searching them can be found, local search can quickly produce good solutions for large-scale instances of a problem. However, local search has known drawbacks. The most common drawback is that it can become trapped in a local optimum. There is no guarantee that this arbitrary local optimum comes close to the global optimum. This inherent drawback of the local search can be mitigated in some cases by multiple starts. However, NP-hard problems often have numerous local optimums, making simple mitigation strategies insufficient. Advanced local search techniques, such as Simulated Annealing and Tabu Search, provide effective methods to escape these local optimums.

Proposed method

Overview

Our approach enables automated problem formulation for Small Language Models (SLMs) by introducing a novel prompt optimisation technique. Generating problem formulations for optimisation tasks using SLMs involves two key components:

1. Providing domain knowledge: SLMs generally possess less domain knowledge compared to Large Language Models (LLMs). Therefore, prompts for SLMs must include additional information to compensate for this gap and guide the model effectively.
2. Ensuring verifiable outputs: The generated problem formulations should be verifiable as either correct or incorrect. To support this, the SLM's output is instructed to be in a predefined format that is adapted to the requirements of the target domain. While LLMs can often follow output formatting with minimal prompting due to their extensive training, SLMs require more explicit and carefully constructed instructions.

In fulfilling these key components, two major challenges arise. To address the first challenge, namely generating the necessary instructions for the SLM, we introduce two operators that analyse the SLM's responses to optimisation problems and generate feedback. These operators then convert the feedback into clear, human-understandable instructions. However, directly inserting these instructions into the prompt can mislead the SLM, leading to the second challenge, incorporating instructions into the prompt in a way the SLM can process and follow effectively. To address this, we employ an approach inspired by metaheuristic algorithms, where instructions are gradually incorporated into the prompt to steer the SLM toward the desired output. Our proposed prompt optimisation method is illustrated in Fig. 1 and is referred to as the Feedback-Integrated Prompt Optimiser (FIPO). The following subsections provide a detailed explanation of the FIPO methodology.

Feedback generation

To address the first challenge of generating meaningful feedback for the SLM, we implement a structured feedback-generation process. We utilise two operators, the Problem Formulation Operator and the Programming Operator to generate feedback. These operators are agentic workflows[49] powered by underlying LLMs. Their primary role is to analyse the problem formulations produced by the target SLM and generate relevant feedback. The target SLM refers to the specific Small Language Model for which we aim to optimise a prompt using our FIPO method. To facilitate feedback generation, we use a separate dataset referred to as the feedback dataset. For each instance in this dataset, the target SLM generates a problem formulation, which is then solved using a standard solver to determine the correctness of the resulting solution. The evaluation results consist of the generated problem formulation, the solver's output logs during execution, and the final solution status. These results serve as the basis for feedback generation by the operators.

- The problem formulation operator reviews the evaluation results from an optimisation perspective. It identifies issues in defining variables, constraints, and objective functions. These issues are then translated into problem formulation-specific feedback.
- The programming operator, on the other hand, evaluates the results from a programming perspective. It detects syntax errors, runtime issues, and problems with output formatting. These are converted into programming-specific feedback.

Feedback from both operators is collected into a shared repository called the feedback pool, which is later used to incorporate instructions into the prompt. This feedback generation process is conducted iteratively during FIPO's execution, allowing the feedback pool to continuously evolve with new insights based on how the target SLM interacts with the optimised prompt.

Prompt optimisation

The prompt optimisation process in FIPO draws inspiration from traditional local search[54]. At each iteration, FIPO randomly selects a piece of feedback from the feedback pool and attempts to enhance the current best-performing prompt by incorporating that feedback. If the newly generated prompt fails to outperform the current best, the feedback is discarded, and another candidate from the pool is considered. FIPO incorporates a selected piece of feedback as an instruction to the current best-performing prompt. This integration is performed using a Large Language Model (LLM), which rewrites the prompt in a way that meaningfully incorporates the intended improvements from the feedback. Although invoking an LLM introduces additional computational overhead, it enables a more meaningful integration of feedback, which accelerates convergence and ultimately enhances the overall efficiency of the approach. The LLM operates with a temperature of one, encouraging exploratory variations during prompt transformation. Each newly generated prompt is then evaluated against a separate dataset, referred to as the training dataset, to assess its effectiveness. During the evaluation, the target SLM generates the problem formulations, representing the stage where the majority of computational effort is concentrated. In practice, there is a considerable possibility that a newly generated prompt may underperform relative to the current best-performing prompt when evaluated with the target SLM. To avoid this, FIPO makes multiple rounds of feedback incorporation with the selected piece of feedback. This iterative prompt optimisation process of FIPO periodically updates the feedback pool with the current behaviour of the SLM with the best-performing prompt, incorporates feedback piece by piece, and generates different versions of incorporated prompts for a selected piece of feedback. These key components make FIPO powerful and gradually evolve the prompt to provide necessary instructions to the SLM.

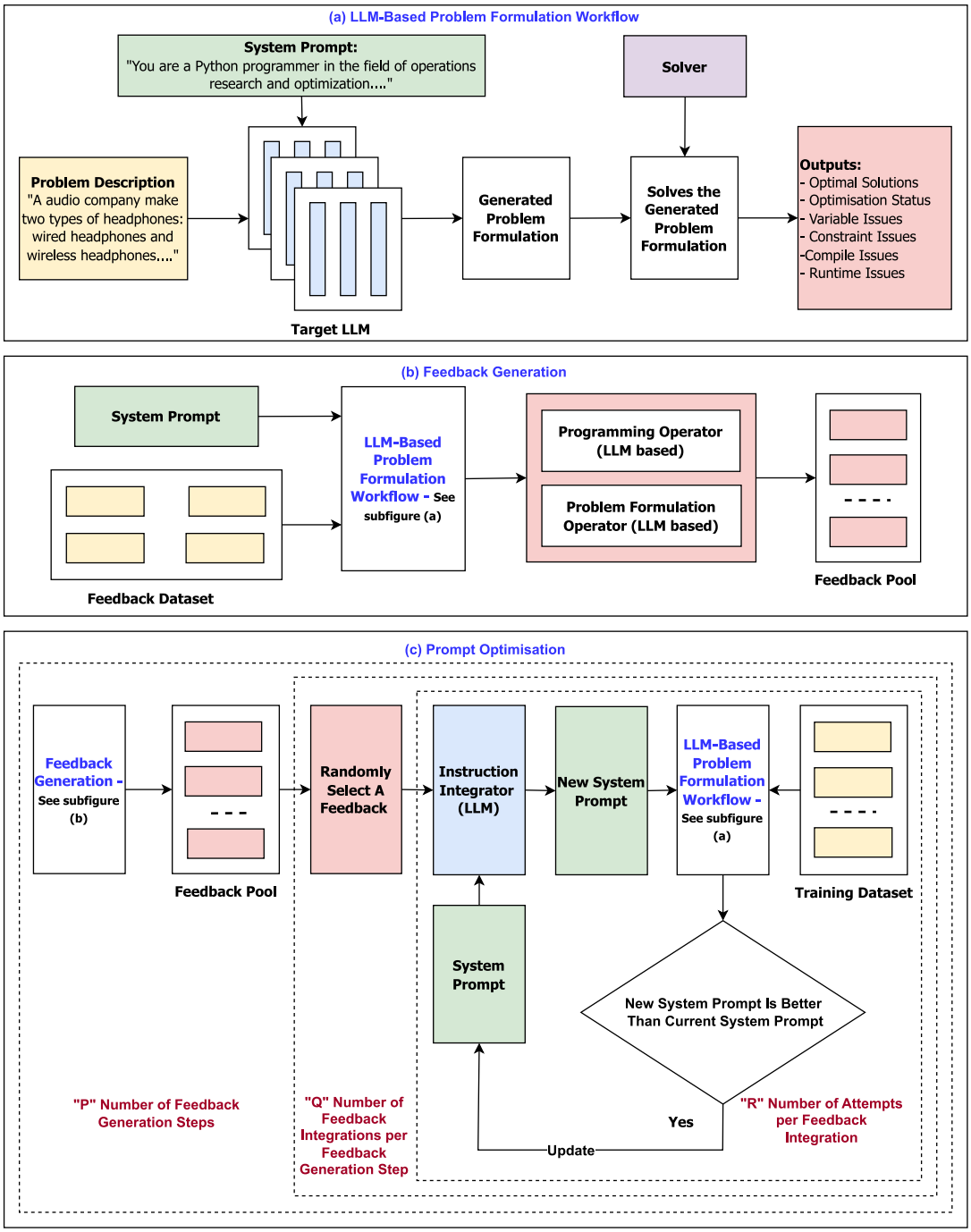

Fig. 1. Overview of Feedback-Integrated Prompt Optimiser (FIPO), illustrated in three interconnected sub-diagrams. (**a**) LLM-Based Problem Formulation Workflow, the system prompt and problem description are processed by the target LLM, which generates problem formulations that are evaluated by a solver, producing outputs such as solution status and variable or constraint issues. (**b**) Feedback Generation, two LLM-based operators (the Problem Formulation Operator and the Programming Operator) analyse the solver outputs and generate feedback, which is collected into a shared feedback pool. (**c**) Prompt Optimisation, explores $P \times Q \times R$ prompts, where P is the number of feedback generation rounds, Q is the number of feedback integrated from the pool per round, and R is the number of retries attempted per selected feedback to refine the prompt.

Implementation details

FIPO is an iterative algorithm designed to automatically refine prompts for a target language model T by leveraging a separate optimiser language model O. The core intuition is to simulate a structured, automated feedback loop where O acts as a critic and editor, diagnosing failures in T's performance under a given prompt

and proposing improvements. The algorithm takes as input an initial prompt I, the target and optimiser models (T and O), a training dataset D_T for evaluation, a feedback dataset D_F for feedback generation, and parameters P, Q, and R controlling the iteration depth. The FIPO procedure (detailed in Algorithm 1 in Supplementary Information (SI)) proceeds over P major iterations. Each iteration consists of three distinct phases: feedback generation, feedback integration, and selective promotion. In the first phase, the current prompt is evaluated by T on the feedback dataset D_F. The generated problem formulations, the solver's output logs during executions, and the final solution statuses are aggregated into the optimiser model O, which is tasked with generating two types of natural-language feedback via specialised operators: one critiques the problem formulation, and the other critiques the programming. The outputs form a diverse feedback pool. In the second phase, the algorithm engages in exploratory integration. For Q steps, a single feedback statement is randomly sampled from the pool. This feedback, along with the current best prompt, is provided to O, which now acts as an instruction integrator to generate a revised prompt candidate. This integration is repeated R times per sampled feedback with a non-zero temperature to encourage diversity. The final phase is a rigorous evaluation and selection step. Each candidate prompt is assessed by measuring T's accuracy on the training dataset D_T. The algorithm maintains a running best prompt and accuracy, promoting a new candidate only if it demonstrates better performance.

Experiments

Overview

We conduct extensive experiments to evaluate the effectiveness of the FIPO framework. For this purpose, we use the LPWP dataset[20], which contains 287 linear programming problems along with their corresponding output solutions and code templates for problem formulations. We randomly partition the dataset into three subsets in a reproducible manner: 8% for training, 17% for feedback generation, and the remaining 75% for testing. This partitioning strategy is intentionally structured to diverge from conventional machine learning practices. A small training subset is allocated to initialise prompts through local search, thereby reducing the substantial computational cost associated with larger training sets. The feedback subset is dedicated to guiding iterative refinement, enabling the method to extract meaningful insights from limited examples in a cost-effective manner. Finally, a large testing set is employed to maximise the number of problem instances reserved for evaluation, ensuring both robustness and representativeness. This design choice is consistent with prior prompt optimisation studies, such as OPRO[15] and EVOPROMPT[16], which similarly adopt small training samples to preserve fairness and efficiency. As target language models (T), we use Qwen-2.5 and Phi-3.5-mini-instruct, which have approximately seven billion and four billion parameters, respectively. For the optimiser language model (O), we employ GPT-4o-mini, accessed via the OpenAI API, selected for its cost-effectiveness. All target models are hosted on an NVIDIA-Tesla-V100 GPU cluster equipped with four GPUs and 382 GB of system memory. For the initial prompt (I), we utilise the Standard and Chain-of-Thought (CoT) prompts introduced in[18] (refer to SI, Supplementary Data S4 and S5 for full prompt templates). To solve the problem formulations generated by the models, we use the Gurobi solver as the standard backend. We use accuracy as the evaluation metric, defined as the percentage of problem formulations that yield the correct ground truth solution after execution:

$$\text{Accuracy} = \frac{\text{Number of Problem Formulations That Successfully Produce the Ground Truth Solution}}{\text{Total Number of Generated Problem Formulations}} \times 100\%. \quad (1)$$

Baselines

We benchmark FIPO against two commercial LLMs, GPT-3.5-turbo and GPT-4o-mini using both Standard and Chain-of-Thought (CoT) prompts. Table 1 presents the baseline performance of these models on the LPWP dataset. GPT-3.5-turbo achieves the highest accuracy, attaining 53% with the Standard prompt and 50% with the CoT prompt. In contrast, GPT-4o-mini exhibits significantly lower performance, reaching only 20% and 9% accuracy with the Standard and CoT prompts, respectively. These results highlight that even larger language models are sensitive to prompt design, with performance varying notably between models. The substantial degradation in GPT-4o-mini's performance further emphasises the critical role of prompt optimisation and model adaptation when working with complex optimisation tasks.

Overall performance

Figure 2 illustrates the training performance behaviour of FIPO using the Standard prompt with the Qwen-2.5 model. The training dataset accuracy exhibits a rapid increase within the first 100 steps, reaching approximately 75%, and subsequently stabilises with minor incremental gains. In parallel, the feedback dataset accuracy follows

Prompt	Dataset	Model	Test dataset accuracy %
Standard	LPWP	GPT-3.5-turbo	53%
Standard	LPWP	GPT-4o-mini	20%
CoT	LPWP	GPT-3.5-turbo	50%
CoT	LPWP	GPT-4o-mini	9%

Table 1. Test accuracy of larger language models under different prompt strategies on the LPWP dataset. This table presents the performance of GPT-3.5-turbo and GPT-4o-mini using two prompting methods: Standard and Chain-of-Thought (CoT).

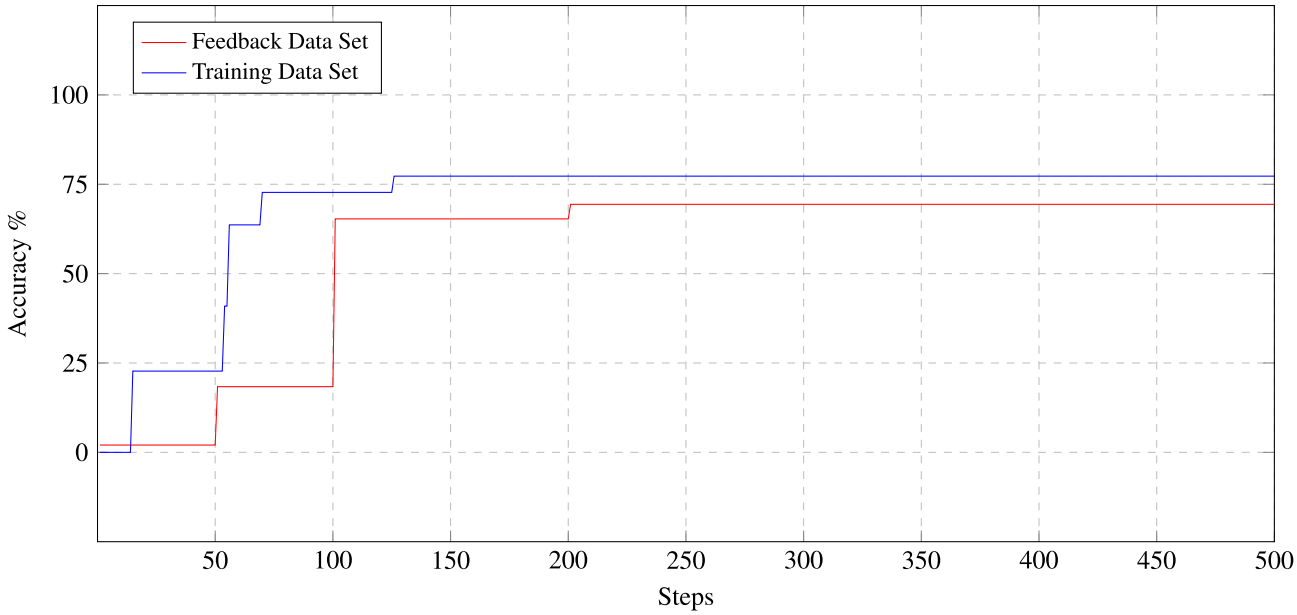

Fig. 2. Training performance progression for the Standard prompt using Qwen-2.5. The plot shows accuracy over optimisation steps for both the training dataset (blue) and the feedback dataset (red). Accuracy is defined as the percentage of problem formulations that produce correct ground truth solutions. The curve illustrates how FIPO iteratively improves prompt performance through feedback integration.

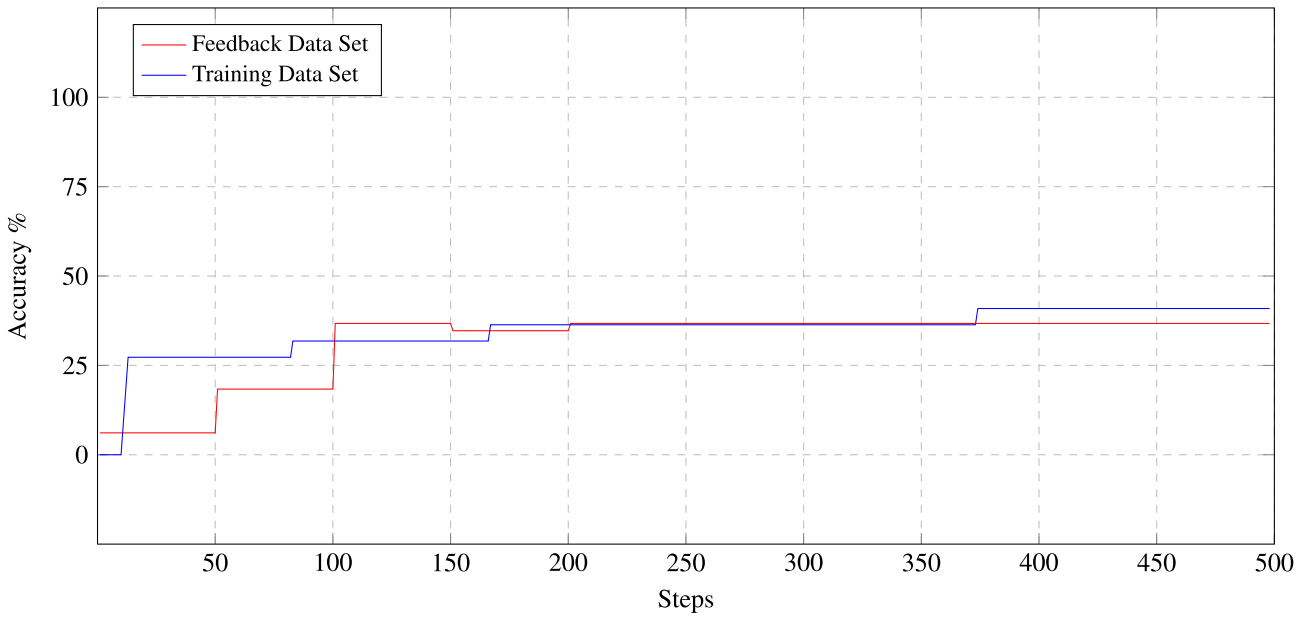

Fig. 3. Training performance progression for the CoT prompt using Qwen-2.5. The graph illustrates accuracy across optimisation steps for both the training dataset (blue) and the feedback dataset (red). Accuracy is measured as the percentage of problem formulations that correctly yield the ground truth solution. Compared to the Standard prompt, the CoT prompt shows slower improvement and lower final accuracy.

a similar but slightly delayed pattern, plateauing around 65%. This early sharp rise indicates that FIPO is highly effective at quickly improving prompt quality during the initial stages of optimisation. The convergence observed after approximately 200 steps suggests that the framework successfully integrates the most beneficial feedback relatively early in the training process.

Figure 3 shows the training performance behaviour of FIPO using the Chain-of-Thought (CoT) prompt with the Qwen-2.5 model. Compared to the Standard prompt setting, the improvement here is more gradual and less pronounced. The training dataset accuracy rises steadily within the first 100 steps, reaching approximately

Prompt	Dataset	Model	Feedback-integrated prompt optimiser	Test dataset accuracy %
Standard	LPWP	Phi-3.5-mini-instruct	✗	6%
			✓	28%
Standard	LPWP	Qwen-2.5	✗	3%
			✓	54%
CoT	LPWP	Phi-3.5-mini-instruct	✗	7%
			✓	28%
CoT	LPWP	Qwen-2.5	✗	3%
			✓	47%

Table 2. Impact of the Feedback-Integrated Prompt Optimiser (FIPO) on model performance. The table compares test dataset accuracy of two language models Phi-3.5-mini-instruct and Qwen-2.5 on the LPWP dataset using Standard and Chain-of-Thought (CoT) prompts. The results include both the original performance of the language models and their performance with the application of FIPO. Across all configurations, the application of FIPO consistently improves accuracy, demonstrating its effectiveness.

Configuration	Number of feedback generation steps (P)	Number of Feedback integrations per feedback generation step (Q)	Number of attempts per feedback integration (R)	Total number of steps	Test dataset accuracy %
Baseline	10	10	5	500	54%
Single feedback generation step	1	10	50	500	37%
	1	100	5	500	44%
Single feedback integration per feedback generation step	10	1	50	500	33%
	100	1	5	500	47%
Single attempt per feedback integration	10	50	1	500	27%
	50	10	1	500	44%

Table 3. Effect of varying FIPO parameters on model performance. This table presents the test dataset accuracy for different configurations of the FIPO framework by varying the number of feedback generation steps (P), feedback integrations per generation step (Q), and attempts per feedback integration (R), while keeping the total number of optimisation steps constant at 500. The baseline configuration (P=10, Q=10, R=5) achieves the highest accuracy. Reducing any of these parameters generally leads to lower performance, highlighting the importance of multiple iterations and diverse feedback integration in improving prompt quality.

40%, and plateaus with only minor gains thereafter. The feedback dataset accuracy closely follows the training accuracy, suggesting good generalisation, but the overall performance remains relatively low. Unlike the Standard prompt setting, where early sharp improvements were observed, the CoT setting exhibits smaller, incremental accuracy increases throughout training, reflecting the additional complexity introduced by the CoT prompting structure. These results indicate that while FIPO can still optimise CoT-based prompts, the learning dynamics are slower, and the achievable accuracy is lower compared to Standard prompts. SLMs with enhanced reasoning capabilities may have the potential to overcome this accuracy barrier. However, addressing this aspect is beyond the scope of the current research.

Table 2 presents the performance of the proposed Feedback-Integrated Prompt Optimiser (FIPO) across different models and prompt styles on the LPWP dataset. Without FIPO, the baseline test accuracy remains very low for both models. Phi-3.5-mini-instruct achieves 6% with the Standard prompt and 7% with the CoT prompt, while Qwen-2.5 achieves only 3% in both cases. However, when FIPO is applied, a substantial improvement is observed. For the Standard prompt, the test accuracy increases from 6 to 28% for Phi-3.5-mini-instruct and from 3 to 54% for Qwen-2.5. Similarly, for the CoT prompt, FIPO boosts the test accuracy from 7 to 28% for Phi-3.5-mini-instruct and from 3 to 47% for Qwen-2.5. These results demonstrate the significant effectiveness of FIPO in enhancing prompt quality and model performance, particularly when using SLMs. Notably, the improvements are more pronounced with Qwen-2.5, indicating that FIPO can be particularly impactful for slightly larger SLMs with more representational capacity. Overall, the results validate the importance of iterative feedback integration for improving automated problem formulation tasks.

Ablation study

Table 3 presents a comparison of different framework parameter configurations and their impact on test dataset accuracy. The baseline configuration, consisting of 10 feedback generation steps (P), 10 feedback integrations per step (Q), and 5 attempts per integration (R), achieves the highest test accuracy of 54%. Reducing the number of feedback generation steps to 1 while keeping $Q = 10$ and $R = 50$ results in a noticeable drop in performance to 37%, and using $Q = 100$ with $R = 5$ yields a slightly higher accuracy of 44%. Similarly, when only a single

Parameter setting (P, Q, R)	Availability of problem formulation operator	Availability of programming operator	Total number of steps	Test dataset accuracy %
(10, 10, 5)	✓	✓	500	54%
(10, 10, 5)	✗	✓	500	31%
(10, 10, 5)	✓	✗	500	37%

Table 4. Impact of feedback generation operators on model performance. This table shows the effect of enabling or disabling the Problem Formulation Operator and Programming Operator during feedback generation, under a fixed parameter setting (P=10, Q=10, R=5) and 500 total optimisation steps. The highest test dataset accuracy is achieved when both operators are active. Removing either operator results in a notable performance drop, highlighting the complementary role of both operator types in generating effective feedback.

Parameter setting (P, Q, R)	Batch size	Total number of steps	Test dataset accuracy %
(10, 10, 5)	1	500	54%
(10, 10, 5)	10	500	32%

Table 5. Effect of batch size on framework performance. This table compares the test dataset accuracy for two configurations with different batch sizes under the same total number of optimisation steps (500). A smaller batch size of 1 yields significantly higher accuracy than a batch size of 10, suggesting that fine-grained, sequential feedback integration is more effective than batching in the FIPO framework.

feedback integration is allowed per generation step ($Q = 1$), the accuracy decreases to 33% and 47% depending on the number of feedback generation steps. Limiting the number of attempts per feedback integration ($R = 1$) further reduces performance, with accuracies falling to 27% and 44%. These results highlight the importance of maintaining a balanced and diverse feedback incorporation process. In particular, multiple feedback generation steps and multiple integration attempts are critical to achieving high-quality prompt optimisation, as they allow broader exploration and more opportunities to refine the prompt effectively. The observed performance trends validate the design choices of FIPO and highlight the importance of maintaining a sufficiently iterative and exploratory framework to optimise prompts for complex tasks such as automated problem formulation.

Table 4 presents the effect of different feedback generation operators on the framework's performance. The full configuration, where both the Problem Formulation Operator and the Programming Operator are active, achieves the highest test dataset accuracy of 54%. When only the Programming Operator is available, the test accuracy drops significantly to 31%, whereas having only the Problem Formulation Operator results in a moderate performance of 37%. These results demonstrate that the combination of feedback from both problem formulation and programming perspectives is crucial for maximising prompt optimisation performance. The notable decrease in accuracy when either operator is removed highlights the complementary roles they play. The Problem Formulation Operator ensures the generated problem is conceptually valid, while the Programming Operator ensures technical correctness and implementation feasibility. Overall, the findings emphasise that leveraging diverse and specialised feedback sources is essential for achieving robust and high-quality prompt improvements in automated problem formulation.

Table 5 analyses the effect of batch size, defined as the number of feedback pieces integrated per optimisation step, on the framework's performance. When the batch size is set to 1, the framework achieves the highest test dataset accuracy of 54%. In contrast, increasing the batch size to 10 results in a significant drop in accuracy to 32%, even though the total number of optimisation steps remains constant at 500. These results indicate that integrating a single piece of feedback at a time is more effective for refining the prompt compared to integrating multiple pieces simultaneously. Fine-grained, incremental updates allow the framework to better assess the individual impact of each feedback piece, reducing the risk of conflicting instructions and preserving the stability of prompt evolution. Conversely, larger batch sizes may introduce competing or redundant modifications in a single step, leading to suboptimal convergence. Overall, the results emphasise that smaller batch sizes corresponding to sequential feedback integration are crucial for achieving high-quality prompt optimisation in complex problem formulation tasks.

Comparison with prompt optimisation methods

Figure 4 compares the training performance behaviour of four different optimisation strategies: Local Search, OPRO[15], EVOPROMPT[16], and FIPO. Unlike FIPO, which integrates feedback to guide prompt refinement, the Local Search approach generates new prompts randomly at each iteration without incorporating any feedback. A new prompt is adopted only if it outperforms the current best prompt. This process is repeated for 500 iterations. For EVOPROMPT, we adopt the genetic algorithm-based implementation with a population size of ten, evolving over 50 generations, resulting in a total of 500 prompt optimisations. OPRO is executed using its standard implementation for 500 iterations. To ensure a fair comparison, FIPO is also run for 500 iterations. This consistent iteration budget across all methods enables a direct comparison of their training dynamics and optimisation efficiency. Although FIPO shows slower initial progress compared to other methods, it consistently achieves higher final accuracy, surpassing 70% and stabilising after the first 100 steps. EVOPROMPT also

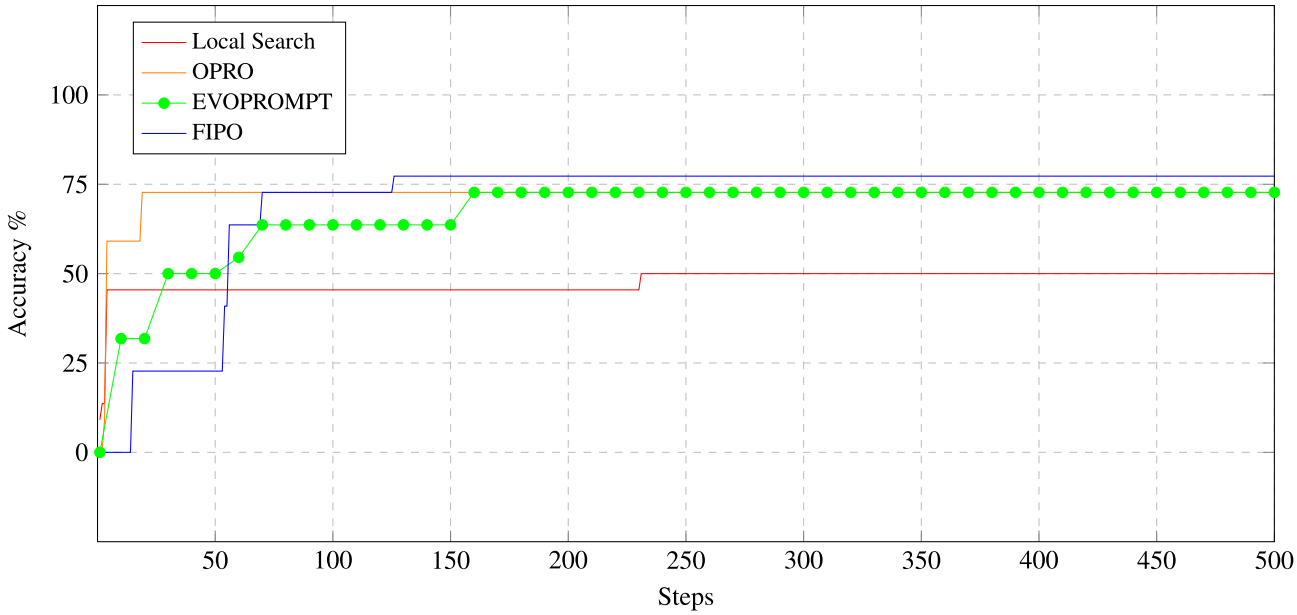

Fig. 4. Comparison of training performance across different prompt optimisation methods. The plot shows accuracy over 500 optimisation steps for four approaches: Local Search, OPRO, EVOPROMPT, and the proposed FIPO framework. FIPO achieves the highest and most stable accuracy, demonstrating better convergence.

Application	Prompt	Dataset	Model	Total number of steps	Test dataset accuracy %
FIPO	Standard	LPWP	Qwen-2.5	500	54%
EVOPROMPT	Standard	LPWP	Qwen-2.5	500	27%
OPRO	Standard	LPWP	Qwen-2.5	500	29%
Local Search	Standard	LPWP	Qwen-2.5	500	24%

Table 6. Benchmarking FIPO against existing prompt optimisation frameworks. This table compares the performance of FIPO with EVOPROMPT, OPRO, and Local Search on the LPWP dataset using the Standard prompt and Qwen-2.5 model. All methods are evaluated under identical conditions with 500 optimisation steps. FIPO achieves a significantly higher test accuracy, highlighting its effectiveness.

shows strong early-stage performance, reaching a plateau around 70% accuracy, but exhibits less fine-grained improvement compared to FIPO. In contrast, OPRO converges the fastest, achieving moderate early gains. However, OPRO quickly plateaus at a lower accuracy level compared to FIPO without significant improvement over time. Local Search demonstrates the slowest and least effective learning curve, plateauing at approximately 50% accuracy with minimal improvements beyond 250 steps.

Table 6 compares the testing performance of FIPO against existing prompt optimisation frameworks, including EVOPROMPT, OPRO, and Local Search using the LPWP dataset and the Qwen-2.5 model. FIPO achieves the highest test dataset accuracy at 54%, significantly outperforming EVOPROMPT (27%), OPRO (29%), and Local Search (24%). The results demonstrate the effectiveness of FIPO's feedback-integrated optimisation approach in producing more accurate and robust prompts compared to evolutionary strategies (EVOPROMPT), reinforcement learning-based techniques (OPRO), and random search methods (Local Search). The substantial performance gap highlights the importance of structured feedback integration and incremental prompt refinement. Overall, these findings validate FIPO as a highly competitive and scalable solution for automated prompt optimisation in tasks such as automated problem formulation.

Qualitative analysis

Table 7 presents a qualitative analysis of various prompts and their influence on test dataset accuracy for the Qwen-2.5 model. The Standard prompt, which provides minimal instruction and limited context, yields a low test accuracy of only 3%. A prompt generated using random sampling combined with local search improves performance moderately, achieving 24% accuracy. In contrast, the highest accuracy of 54% is obtained using the prompt optimised through FIPO, which incorporates explicit guidance on modelling structure, example-driven formatting, and clearly defined instructions. Notably, a manually revised version of the FIPO-optimised prompt, where incorrect examples were corrected, achieved an intermediate accuracy of 36%. This result underscores the high sensitivity of SLMs to prompts and highlights the necessity of prompt optimisation. From a qualitative

			Prompt characteristics			
Prompt	**Overview**	**Test dataset accuracy %**	**Clear and precise**	**Contextual information**	**Provide examples**	**Specify output format**
You are a Python programmer in the field of operations research and optimisation. Your proficiency in utilising third-party libraries such as Gurobi is essential. In addition to your expertise in Gurobi, it would be great if you could also provide some background in related libraries or tools, like NumPy, SciPy, or PuLP. You are given a specific problem. Your aim is to develop an efficient Python program that addresses the given problem while ensuring that the objective function accurately reflects the quantities to be minimised or maximised based on the problem requirements, including the correct coefficients for each variable. Additionally, you must ensure that decision variables are defined as non-negative integers to accurately represent the quantities of molars and canines being filled in the optimisation problem. Now the origin problem is as follow: $\{problem\}$ Please rewrite the following Python code block to match the specified format, incorporating the following instruction: "Generate a Python code block that formulates and solves an optimisation problem using either the Gurobi or Pulp library, ensuring to define the objective function, constraints, and decision variables clearly based on the given problem description, with decision variables as non-negative integers." Ensure the code block includes a function definition, argument descriptions, return value description, and a placeholder for implementation. The function should accept two integer parameters for bike and car capacities and should return an integer objective value. Here is a starter code: $\{code\_example\}$ Please provide a single Python code block that solves the optimisation problem using Gurobi, enclosed within triple backticks.	Developed using FIPO for Qwen-2.5	54%	High-Guide precise modelling	High-Provide concrete context	High-Strong example reference, though some examples contain inaccuracies	High-Very specific output format
You are a Python programmer in the field of operations research and optimization. Your proficiency in utilizing third-party libraries such as Gurobi is essential. In addition to your expertise in Gurobi, it would be great if you could also provide some background in related libraries or tools, like NumPy, SciPy, or PuLP. You are given a specific problem. Your aim is to develop an efficient Python program that addresses the given problem while ensuring that the objective function accurately reflects the quantities to be minimized or maximized based on the problem requirements, including the correct coefficients for each variable. Additionally, you must ensure that the decision variables are defined as non-negative integers to accurately represent the quantities involved in the optimization problem, such as the number of bikes and cars, ensuring they reflect real-world countable items. Now the origin problem is as follow: $\{problem\}$ Please rewrite the following Python code block to match the specified format, incorporating the following instruction: "Generate a Python code block that formulates and solves an optimization problem using either the Gurobi or Pulp library, ensuring to define the objective function, constraints, and decision variables clearly based on the given problem description, with decision variables as non-negative integers." Ensure the code block includes a function definition, argument descriptions, return value description, and a placeholder for implementation. The function should accept relevant input parameters and return a value representing the computed objective. Here is a starter code: $\{code\_example\}$ Please provide a single Python code block that solves the optimization problem using Gurobi, enclosed within triple backticks.	FIPO-Developed Prompt with Manually Corrected Examples-Qwen-2.5	36%	High - Guide precise modelling	High - Provide concrete context	High - Strong example reference	High-Very specific output format
You are a highly skilled Python programmer with a specialisation in operations research and optimisation. Your expertise in advanced libraries, particularly Gurobi, equips you to adeptly solve intricate optimisation problems. Additionally, your knowledge of complementary libraries such as NumPy, SciPy, and PuLP further strengthens your problem-solving abilities. Your task is to design a robust and efficient Python program that effectively addresses the following optimisation challenge: $\{problem\}$ You'll have access to starter code that serves as a framework for your implementation: $\{code\_example\}$ Please ensure to submit only the polished Python code that successfully resolves the outlined problem.	Developed using local search for Qwen-2.5	24%	Medium-Lack of specific guidance	Medium-Lack of tailored context	Low-Lack of specific guidance on formatting & structure	Medium-Lack how to format or what to exclude or include
You are a Python programmer in the field of operations research and optimisation. Your proficiency in utilising third-party libraries such as Gurobi is essential. In addition to your expertise in Gurobi, it would be great if you could also provide some background in related libraries or tools, like NumPy, SciPy, or PuLP. You are given a specific problem. You aim to develop an efficient Python program that addresses the given problem. Now the origin problem is as follow: $\{problem\}$ Give your Python code directly. Here is a starter code: $\{code\_example\}$	Standard prompt tested using Qwen-2.5	3%	Low-Vague instructions	Low-Only mention libraries	Low-Vaguely mention starter code	Low-No format instructions

Table 7. Qualitative analysis of prompts.

perspective, prompts differ notably across several characteristics: clarity and precision, provision of contextual information, inclusion of example formatting, and specificity of instructions. The Standard prompt scores low across all dimensions, offering vague guidance and limited context. In contrast, the FIPO-optimised prompt delivers high clarity, precise instructions, structured context, and illustrative examples even though they are incorrect, significantly aiding the model's ability to formulate problems correctly. These results demonstrate that well-structured prompts, enriched with domain-specific guidance are crucial for maximising SLM performance in complex tasks like automated problem formulation. Furthermore, the substantial performance gap between

the baseline and the FIPO-optimised prompt highlights the importance of systematic prompt refinement processes.

Conclusion

In this research, we introduced FIPO, a novel Feedback-Integrated Prompt Optimiser designed to enhance automated problem formulation capabilities for Small Language Models (SLMs). While SLMs offer a computationally efficient alternative to larger models, their limited generalisation abilities make them particularly sensitive to prompt quality. To address this challenge, we developed a metaheuristic-based local search approach that systematically refines prompts through iterative feedback integration. FIPO leverages agentic workflows to generate diverse feedback from both problem formulation and programming perspectives, enabling a structured and targeted optimisation process. Through comprehensive experiments on the LPWP dataset, we observed that FIPO consistently improves the problem formulation accuracy of SLMs compared with baseline prompt engineering approaches. Furthermore, our results show that FIPO delivers consistent improvements over state-of-the-art prompt optimisation methods, including OPRO and EVOPROMPT, highlighting the effectiveness of structured feedback integration and incremental prompt evolution. Ablation studies further validated the importance of key design elements, such as multiple rounds of feedback incorporation and fine-grained feedback integration, in achieving robust optimisation outcomes. Overall, our findings establish feedback-driven, metaheuristic prompt optimisation as a promising strategy for enabling cost-efficient SLMs for automated problem formulation. In future, we plan to enhance FIPO by integrating reasoning-enhanced language models to automate even more complex large optimisation datasets and to further investigate the underlying causes of prompt sensitivity in SLMs within the context of problem formulation.

Data Availability

The implementation of the FIPO framework will be publicly released upon acceptance of the paper. The data that support the findings of this study are available at the following link: https://github.com/pivithuruthejanamarasinghe/FIPO-LPWP.

Received: 6 June 2025; Accepted: 4 November 2025
Published online: 24 November 2025

References

1. Pochet, Y., & Wolsey, L. A. *Production Planning by Mixed Integer Programming*, vol. 149 (Springer, 2006).
2. Toth, P., & Vigo, D. *The Vehicle Routing Problem* (SIAM, 2002).
3. Coffman Jr, E. G., Garey, M. R., & Johnson, D. S. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.* **7**(1), 1–17 (1978).
4. Kłosowski, G., Edward, K., & Gol, A. integer linear programming in optimization of waste after cutting in the furniture manufacturing. In *Intelligent Systems in Production Engineering and Maintenance–ISPEM 2017: Proceedings of the First International Conference on Intelligent Systems in Production Engineering and Maintenance ISPEM 2017 1*, 260–270 (Springer, 2018).
5. Bellman, R. A Markovian decision process. *J. Math. Mech.* **6**(5), 679–684 (1957).
6. Lawler, E. L. & Wood, D. E. Branch-and-bound methods: A survey. *Oper. Res.* **14**(4), 699–719 (1966).
7. Luke, S. *Essentials of Metaheuristics*. Lulu, 2nd edn. (2013). http://cs.gmu.edu/∼sean/book/metaheuristics/.
8. Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual (2024). https://www.gurobi.com.
9. Google OR-Tools. CP-SAT Solver (2023). https://developers.google.com/optimization/cp/cp_solver.
10. IBM ILOG Cplex. V12. 1: User's Manual for CPLEX. *Int. Bus. Mach. Corp*. **46**(53), 157 (2009).
11. Astorga, N., Liu, T., Xiao, Y., & van der Schaar, M. Autoformulation of mathematical optimization models using LLMs. arXiv preprint arXiv:2411.01679 (2024).
12. Raza, M., Jahangir, Z., Riaz, M. B., Saeed, M. J. & Sattar, M. A. Industrial applications of large language models. *Sci. Rep.* **15**(1), 13755 (2025).
13. AhmadiTeshnizi, A., Gao, W., & Udell, M. OptiMUS: Optimization modeling using MIP solvers and large language models. arXiv preprint arXiv:2310.06116 (2023).
14. Jiang, X., Wu, Y., Zhang, C., &, Zhang, Y. DROC: Elevating large language models for complex vehicle routing via decomposed retrieval of constraints. In *13th International Conference on Learning Representations, ICLR 2025*. OpenReview. net. (2025).
15. Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., & Chen, X. Large language models as optimizers. arXiv preprint arXiv:2309.03409 (2023).
16. Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., Liu, G., Bian, J., Yang, Y. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. arXiv preprint arXiv:2309.08532 (2023).
17. Wang, T., Yu, W.-Y., She, R., Yang, W., Chen, T., Zhang, J. Leveraging large language models for solving rare MIP challenges. arXiv preprint arXiv:2409.04464 (2024).
18. Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., et al. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations* (2023).
19. Yang, X., Song, L., Zhang, Y., & Bian, J. HEURAGENIX: A multi-agent LLM-based paradigm for adaptive heuristic evolution and selection in combinatorial optimization (2024). https://openreview.net/forum?id=xxSK3ZNAhh.
20. Ramamonjison, R., Li, H., Yu, T. T., He, S., Rengan, V., Banitalebi-Dehkordi, A., Zhou, Z., & Zhang, Y. Augmenting operations research with auto-formulation of optimization models from problem descriptions. arXiv preprint arXiv:2209.15565 (2022).
21. Yang, Z., Huang, Y., Shi, W., Feng, L., Song, L., Wang, Y., Liang, X., & Tang, J. OPTIBENCH MEETS RESOCRATIC: Measure and Improve LLMs for Optimization Modeling. arXiv preprint arXiv:2407.09887 (2024).
22. AhmadiTeshnizi, A., Gao, W., Brunborg, H., Talaei, S., Lawless, C., & Udell, M. OptiMUS-0.3: Using large language models to model and solve optimization problems at scale. arXiv preprint arXiv:2407.19633 (2024).
23. Yansen, Z., Qingcan, K., Wing, Y.Y., Hailei, G., Xiaojin, F., Xiongwei, H., Tao, Z., Chen, M. Decision Information Meets Large Language Models: The Future of Explainable Operations Research. arXiv preprint arXiv:2502.09994 (2025).
24. Barbierato, E., Gatti, A., Incremona, A. & Pozzi, A. The ontological and ethical evolution of machine learning. *IEEE Access* Breaking Away From AI (2025).
25. Yupeng Chang, X. et al. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.* **15**(3), 1–45 (2024).
26. Ashish, V., Noam, S., Niki, P., Jakob, U., Llion, J., Aidan, N.G., Łukasz, K., & Illia, P. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **30** (2017).

27. Josh, A., Steven, A., Sandhini, A., Lama, A., Ilge, A., Florencia, L.A., Diogo, A., Janko, A., Sam, A., Shyamal, A., et al. GPT-4 Technical Report. arXiv preprint arXiv:2303.08774 (2023).
28. Anthropic. Claude, 2025. https://claude.ai/. Large language model.
29. Gemini Team et al. Gemini: A Family of Highly Capable Multimodal Models. arXiv preprint arXiv:2312.11805 (2023).
30. Hugo, T., et al. LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971 (2023).
31. Zhiling, Z., et al. Large language models for reticular chemistry. *Nat. Rev. Mater.* 1–13 (2025).
32. Duzhen, Z., Yahan, Y., Jiahua, D., Chenxing, L., Dan, S., Chenhui, C., & Dong, Y. MM-LLMs: Recent Advances in Multimodal Large Language Models. arXiv preprint arXiv:2401.13601 (2024).
33. Van Nguyen, C., et al. A Survey of Small Language Models. arXiv preprint arXiv:2410.20011 (2024).
34. Paulius, M., et al. Mixed Precision Training. In *International Conference on Learning Representations* (2018). https://openreview.net/forum?id=r1gs9JgRZ.
35. Hong, L., Zhiyuan, L., David, L. W.H., Percy, L., & Tengyu, M. Sophia: A scalable stochastic second-order optimizer for language model pre-training. In *The Twelfth International Conference on Learning Representations*. (2024). https://openreview.net/forum?id=3xHDeA8Noi.
36. Bohang, Z., Jikai, J., Cong, F., & Liwei, W. Improved analysis of clipping algorithms for non-convex optimization. arXiv preprint arXiv:2010.02519 (2020).
37. Angelos, K., Apoorv, V., Nikolaos, P., & François, F. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 5156–5165 (PMLR, 2020).
38. Elias, F. & Dan, A. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, 10323–10337 (PMLR, 2023).
39. Elias, F., Saleh, A., Torsten, H., & Dan, A. GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers. arXiv preprint arXiv:2210.17323 (2022).
40. Inar, T. & Jean-Loup, T. Baby Llama: knowledge distillation from an ensemble of teachers trained on a small dataset with no performance penalty. arXiv preprint arXiv:2308.02019 (2023).
41. Pranab, S., Ayush, K.S., Sriparna, S., Vinija, J., Samrat, M., & Aman, C. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. arXiv preprint arXiv:2402.07927 (2024).
42. Yuxin, W., Neel, J., John, K., Micah, G., Jonas, G., & Tom, G Hard prompts made easy: gradient-based discrete optimization for prompt tuning and discovery. *Adv. Neural Inf. Process. Syst.* **36**, (2024).
43. Zhang, Z., Liu, S. & Cheng, J. Exploring prompts in few-shot cross-linguistic topic classification scenarios. *Appl. Sci.* **13**(17), 9944 (2023).
44. Ruotian, M., Xiaolei, W., Xin, Z., Jian, L., Nan, D., Tao, G., Qi, Z., & Xuanjing, H. Are Large Language Models Good Prompt Optimizers? arXiv preprint arXiv:2402.02101 (2024).
45. Yongchao, Z., Andrei, I.M., Ziwen, H., Keiran, P., Silviu, P., Harris, C., & Jimmy, B. Large Language Models Are Human-Level Prompt Engineers. arXiv preprint arXiv:2211.01910 (2022).
46. Reid, P., Dan, I., Jerry, L., Yin, T.L., Chenguang, Z., & Michael, Z. Automatic Prompt Optimization with Gradient Descent and Beam Search. arXiv preprint arXiv:2305.03495 (2023).
47. Xinyuan, W., Chenxi, L., Zhen, W., Fan, B., Haotian, L., Jiayou, Z., Nebojsa, J., Eric, P.X., & Zhiting, H. PROMPTAGENT: Strategic Planning with Language Models Enables Expert-Level Prompt Optimization. arXiv preprint arXiv:2310.16427 (2023).
48. Hong, S., Xue, L., Yinchuan, X., Youkow, H., Qi, C., Min, W., Jian, J., & Denis, C. AutoHint: Automatic Prompt Optimization with Hint Generation. arXiv preprint arXiv:2307.07415 (2023).
49. Jiayi, Z., et al. AFLOW: Automating Agentic Workflow Generation. arXiv preprint arXiv:2410.10762 (2024).
50. Sirui, H., Yizhang, L., Bang, L., Bangbang, L., Binhao, W., Ceyao, Z., Chenxing, W., Danyang, L., Jiaqi, C., Jiayi, Z., et al. DATA INTERPRETER: An LLM Agent for Data Science. arXiv preprint arXiv:2402.18679 (2024).
51. Zelong, L., et al. AutoFlow: Automated Workflow Generation for Large Language Model Agents. arXiv preprint arXiv:2407.12821 (2024).
52. Thomas, S. Local Search Algorithms for Combinatorial Problems. *Darmstadt University of Technology PhD Thesis*, 20 (1998).
53. Blum, C. & Roli, A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Comput. Surv. CSUR* **35**(3), 268–308 (2003).
54. Yves, C., Antoon, W.J.K., & Pesch, E.J. Local search in combinatorial optimization. In *Artificial Neural Networks: An Introduction to ANN Theory and Practice*, 157–174 (2005).
55. David, S. J., Christos, H.P., & Mihalis, Y. How easy is local search? *J. Comput. Syst. Sci.* **37**(1), 79–100 (1988).

Author contributions

All authors contributed to this research. P.T.A. designed the methodology, led the development of the framework, conducted all experiments, and wrote the manuscript. Authors S.N., Y.S., and D.A. provided supervision throughout the research and critically reviewed the research artefacts and manuscript drafts.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-025-27495-8.

Correspondence and requests for materials should be addressed to P.T.A.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

© The Author(s) 2025