
PromptLab: Fine-Tuned Language Models for Interactive Prompt Optimization and Quality Scoring

Brandon Li

University of Illinois
Urbana-Champaign
Champaign, IL
bcli3@illinois.edu

Nusrat Chowdhury

University of Illinois
Urbana-Champaign
Champaign, IL
nusratc2@illinois.edu

Mia Onodera

University of Illinois
Urbana-Champaign
Champaign, IL
mco7@illinois.edu

Sean Liao

University of Illinois
Urbana-Champaign
Champaign, IL
seanyl2@illinois.edu

Abstract

The growing adoption of large language models (LLMs) across research, industry, and everyday use has exposed a persistent challenge: the quality of model outputs is highly sensitive to how user inputs, or prompts, are formulated. Despite this sensitivity, most users lack formal training in prompt engineering, and no widely accessible tool exists to help them evaluate and improve their prompts in real time.

This report presents PromptLab¹, a web-based application that addresses this gap by providing automated prompt evaluation, optimization, and generation. PromptLab uses fine-tuned versions of SmoLLM2-360M-Instruct [1], a compact but capable instruction-tuned language model, to rewrite weak user prompts into structured, effective alternatives and to score prompts across five interpretability dimensions: helpfulness, correctness, coherence, complexity, and verbosity. Explanatory feedback is generated by a Gemini API integration when enabled, providing users with actionable per-dimension guidance. The system is delivered through a React-based chat interface backed by a Flask API server, making it accessible to both technical and non-technical users.

Author Keywords

prompt optimization; language models; fine-tuning; HCI

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

Copyright held by the owner/author(s).

CS568, Spring 2026, UIUC

ACM 978-1-4503-6819-3/20/04.

<https://doi.org/10.1145/3334480.XXXXXXX>

¹<https://github.com/onoderamia/CS568>

CCS Concepts

•**Human-centered computing** → **Human computer interaction (HCI)**; •**Computing methodologies** → *Natural language processing*; Machine learning;

Introduction

Large language models have become central tools in domains ranging from software engineering to scientific writing and education. Despite their capabilities, their practical utility is fundamentally constrained by a well-documented challenge: model output quality is highly sensitive to the structure and specificity of the input prompt [9]. Poorly formed prompts routinely produce outputs that fail to satisfy user intent even when the underlying model is highly capable [11], a phenomenon sometimes called the *prompt gap*.

Prompt engineering has emerged as a recognized discipline for addressing this gap [12]. Techniques such as chain-of-thought prompting [14] and few-shot exemplars [2] have been shown to substantially improve model performance. Despite this, prompt engineering remains largely inaccessible: it requires familiarity with model internals, iterative experimentation, and domain-specific knowledge that most users lack.

We address this need by building PromptLab, an end-to-end web application for prompt evaluation, optimization, and generation. Our system makes the following contributions:

- A prompt optimization model fine-tuned on the BPO dataset [16] that rewrites weak prompts into clearer, more actionable alternatives.
- A prompt scoring model fine-tuned on HelpSteer2 [13] that rates prompts across five dimensions: helpful-

ness, correctness, coherence, complexity, and verbosity.

- A prompt generation module that converts a short task description into a structured, ready-to-use prompt.
- A full-stack React and Flask interface with optional Gemini API [4] explanations and interactive sentence-level refinement, making the system accessible to non-technical users.

Related Work

Prompt engineering has been studied both as a human challenge and as a target for automation, yet existing systems address only parts of the problem that PromptLab approaches holistically. Zamfirescu-Pereira et al. [17] established through a CHI study that non-AI experts struggle to design effective prompts even when given structured tools, identifying opportunistic rather than systematic behavior as a core failure mode. Their BotDesigner prototype surfaced the need for scaffolded prompt feedback but required users to evaluate prompts themselves against conversation logs. PromptLab addresses this gap: rather than exposing the evaluation burden to the user, our HelpSteer2-fine-tuned scoring model automatically rates any submitted prompt across five interpretability dimensions and returns per-dimension scores with optional Gemini-generated explanations, removing the need for user expertise in assessing prompt quality. On the algorithmic side, frameworks like DSPy [8] and EvoPrompt [5] have demonstrated powerful automated prompt optimization pipelines. DSPy abstracts prompts into modular program components and uses “teleprompters” to optimize entire pipelines against a metric, while EvoPrompt applies evolutionary algorithms (genetic and differential evolution) to iteratively mutate and select better prompt candidates, outperforming manually

engineered prompts by up to 25% on BIG-Bench Hard tasks. Both systems, however, are designed for developers building LLM pipelines, require programmatic interfaces, and depend on labeled datasets or evaluation metrics defined in advance. PromptLab instead serves end users without requiring such infrastructure, optimizing prompts on demand through a BPO LoRA adapter fine-tuned specifically for prompt rewriting.

More recent systems have moved toward accessibility but retain dependencies that limit their reach. Promptmatix [10] from Salesforce AI Research generates structured prompts from natural language task descriptions without manual tuning, using meta-prompt-based optimization and DSPy-powered compilation with cost-aware objectives. While its goal of accessibility aligns with PromptLab's, Promptmatix still relies on large proprietary LLM APIs for its optimization loop, incurring per-query costs and introducing latency. PromptLab avoids this by fine-tuning SmolLM2-360M-Instruct [1] with LoRA [6] locally, achieving optimization inference in 6.38 seconds on CPU with no external API dependency for its core rewriting functionality. GREATER-PROMPT [7] provides a unified open-source toolkit for prompt optimization and explicitly critiques existing tools, including commercial offerings like Jina PromptPerfect, for their reliance on proprietary models and limited usability for non-experts. PromptLab shares this open-source, low-cost motivation while going further by integrating prompt scoring and generation into a single conversational interface, rather than exposing separate optimization algorithms that still require user configuration.

Where PromptLab most distinctly departs from prior work is in its support for interactive, sentence-level prompt refinement within a chat interface. Prompt-with-Me [15] represents the closest comparable approach to in-context prompt

assistance, embedding automated quality feedback and duplicate detection directly into a developer IDE workflow. However, it is scoped to software engineering contexts and does not support rewriting or generation. PromptLab's refinement loop allows users to select individual sentences in an optimized or generated prompt and apply targeted actions (elaborating a selected clause for more specificity, condensing it for brevity, or regenerating the entire prompt with structural variation) through a React chat UI backed by Flask-hosted local inference.

The technical foundations enabling PromptLab's design address limitations common across prior work. Parameter-efficient fine-tuning via LoRA [6] makes it practical to adapt compact models like SmolLM2-360M-Instruct [1] to specific tasks on consumer hardware, enabling the fast local inference that API-dependent systems like Promptmatix [10] cannot offer. Ranking-based supervision [3], as used in training our scoring model on HelpSteer2 [13], grounds our multi-dimensional quality scores in human judgments of relative quality, providing the kind of interpretable, per-dimension feedback that tools like DSPy [8] and EvoPrompt [5] forgo in favor of single aggregate metrics. Together, these foundations allow PromptLab to close the gap identified by Zamfirescu-Pereira et al. [17]: a locally-deployable, interactive system that gives non-expert users structured, actionable feedback without requiring programming knowledge, API access, or pre-labeled evaluation data.

Approach

PromptLab's frontend is a React application built with Vite. It runs in the browser and talks to a local Flask backend that hosts all large-language-model inference. The backbone model for every adapter is [SmolLM2-360M-Instruct](#) from Hugging Face.

We fine-tuned two complementary LoRA adapters on top of that base: one for prompt optimization and one for multi-dimensional quality scoring. Optimization and refinement of checked prompts use the BPO LoRA; scoring uses the HelpSteer rater LoRA (with a fallback adapter path if the primary rater checkpoint is absent). Optional feedback text for the score cards can be generated with the Google Gemini API [4] when enabled via environment configuration. Otherwise the UI falls back to deterministic template explanations.

Datasets

The BPO adapter is trained on the [Better Prompt Optimization](#) corpus [16] (original user prompts paired with human- or expert-written optimized prompts). The rater adapter is trained on [Nvidia's HelpSteer](#) dataset, which provides human ratings of assistant responses along five dimensions: helpfulness, correctness, coherence, complexity, verbosity. Each dimension is then mapped to our 1–5 UI scale. Training scripts live under the backend/colab directory and use Unsloth, PEFT, and supervised fine-tuning (SFT) with LoRA rather than full model fine-tuning, which keeps compute and storage requirements modest while specializing behavior.

Training SmolLM Models

We use the same base checkpoint for both fine-tuning tasks: SmolLM2-360M-Instruct, loaded in 4-bit for Colab training via Unsloth. LoRA is applied to the standard attention and MLP projection modules with rank 16, which is a typical trade-off between capacity and stability for small models.

For prompt optimization, we load the BPO dataset and format each example as a ChatML-style multi-turn string: a fixed system message instructing the model to rewrite prompts for clarity and specificity, a user turn containing the raw prompt, and an assistant turn containing the tar-

get optimized prompt. We train with the Hugging Face TRL SFTTrainer on the text field, with hyperparameters chosen for short runs on a single GPU (on the order of a few hundred steps, small batch with gradient accumulation, and a linear learning-rate schedule in our BPO Colab script). The exported LoRA weights are stored as `smollm_bpo_lora` and loaded at inference with PEFT on top of the same base model used in training.

For dimension scoring, we load the HelpSteer dataset. Each row provides a user prompt, an assistant response, and numeric labels for the five dimensions. Native HelpSteer labels are on a 0–4 scale; we shift and clip them to 1–5 for consistency with the UI. The training objective is to predict a single compact JSON object with exactly those five keys and integer values. The model sees a system message that defines the rater role and key definitions, then a user message that wraps the prompt and response in simple XML-like tags. Training again uses SFT on formatted sequences. The resulting adapter is stored as `smollm_helpsteer_rater_lora`, with `smollm_ranked_lora` available as a fallback when the primary adapter path is missing in a given checkout.

At runtime the Flask server loads the base causal LM for optimization (BPO adapter) and a separate stack for scoring (base plus rater LoRA, and a base-only generation path for auxiliary text such as the draft reply used in scoring). This separation avoids feeding explanation or draft generation through the rater weights, which would bias outputs toward JSON-like text.

Frontend and Backend Integration

The React app is served by the Vite development server. The browser never loads the PyTorch models; all inference happens on the machine running Flask, by default on port 5001.

Check Prompt mode issues parallel requests for scoring and optimization: one POST carries the user's prompt to `score_dims`, another to optimize. The backend returns JSON for dimension scores, an overall average, per-dimension explanation strings, an optional draft assistant reply used inside the scoring pipeline, and the optimized prompt text. Generate Prompt mode sends the user's short idea to `generate_task_prompt` and displays the generated prompt plus supporting UI text. Both flows can optionally open interactive refinement of the displayed prompt.

Interactive Refinement

After an optimized prompt or a generated prompt is shown, the UI splits the text into sentences using the same punctuation-based splitting rule as the server. The user selects exactly one sentence. Two actions apply only to that sentence, and one action rewrites the whole prompt.

The 'Elaborate' button replaces the selected sentence with a longer instruction that adds prompt-engineering detail only (output format, scope, constraints, audience, what to include or exclude). The model is explicitly forbidden from answering the underlying topic. The frontend POSTs to `/api/refine_optimized` (Check tab) or `/api/refine_generated` (Generate tab) with mode set to sentence, the sentence index, action elaborate, the full current text, and the original user prompt or idea. The backend runs one BPO generation, cleans the new span, drops any near-duplicate of unchanged sentences, splices the result back at that index, and returns the full updated string. Only the displayed prompt field in the chat message is updated; scores and feedback cards are not recomputed.

The 'More concise' button shortens the selected sentence: the backend asks the BPO model for a compressed version with fewer words, rejects outputs that are not shorter than the input when possible, and may fall back to a sim-

ple word-count trim. The same splice-and-return path and "patch only the prompt text" behavior apply on the frontend.

Guidelines for Optimized Prompt Generation

Small instruction-tuned models often leak meta-text (i.e. "Here is an optimized version. . .") or answer the user's task instead of rewriting the instruction. We address this with layered constraints and post-processing.

System prompts for optimization and for "idea to prompt" generation spell out non-negotiable rules: output only the final prompt text; do not answer or execute the task; preserve intent and task form (question vs directive); cap length; require concrete constraints and an explicit output-format hint where appropriate; forbid leading labels such as "Rewritten:" or conversational openers such as "Here's"; forbid wrapping the entire rewrite in quotation marks. A dedicated post-processing routine strips repeated "Here is . . .:" prefixes, peels a leading quoted span when the model wraps the rewrite, and truncates trailing commentary phrases the model sometimes appends after the real rewrite.

Primary optimization may fail lightweight validation (empty output, unchanged text, meta prefixes, answer-like openings, or extreme length). In that case, the server triggers a stricter second-pass prompt that again forbids answering and demands only the rewritten prompt in two to three sentences.

Evaluation

We evaluate PromptLab across four dimensions: prompt optimization quality, qualitative comparison with larger LLMs, system latency, and model training convergence.

PromptLab Output Analysis

Table 1 presents the five input prompts evaluated in this study along with their corresponding PromptLab optimized

outputs. Figure 1 shows word count and quality score distributions across these prompts. Optimized prompts are consistently longer than the originals, reflecting added specificity and context. Quality scores across all five HelpSteer dimensions (helpfulness, correctness, coherence, complexity, verbosity) converge around a mean of 3 across prompts, likely bounded by the scoring model’s training distribution and parameter capacity, constraining the range of scores it can meaningfully distinguish.

Qualitative Comparison with Larger Models

To assess output quality, we selected three representative prompts spanning different task types and ran each through PromptLab’s BPO LoRA optimizer, GPT-4o, and Claude. Table 2 presents the rewrites side by side. PromptLab produced outputs that were competitive in structure and clarity compared to larger models, though it occasionally lacked the depth and specificity of GPT-4o and Claude. This gap is expected given the significant difference in model scale; Best Prompt Optimization (BPO) fine-tuning partially bridges this by directing the model’s capacity toward prompt rewriting specifically rather than broad generalization.

System Latency

Figure 3 shows per-endpoint latency measured on CPU inference (Apple M1, single request). The Optimize endpoint completes in 6.38s using only the BPO LoRA adapter. Score Dims takes 19.01s total, split between 11.25s for SmoLM2 scoring and 7.76s for the Gemini API feedback call. Generate Prompt takes 13.16s. The higher latency of Score Dims and Generate Prompt relative to Optimize reflects their larger token throughput demands. On GPU (e.g., Tesla T4), these latencies would reduce to approximately 1–2s per request.

Training Convergence

Both LoRA adapters were trained for 200 steps on a Tesla T4 GPU using the Unsloth framework with 4-bit quantization. Figure 2 shows training loss curves for the BPO adapter (prompt optimization, final loss 1.20) and the Ranked adapter (quality scoring). Both models converge smoothly within 200 steps, demonstrating that LoRA fine-tuning is sample-efficient even with a 360M parameter base model. The BPO adapter trained on 13,895 examples from the BPO dataset; the Ranked adapter trained on 10,000 examples from the HelpSteer-ranked dataset.

Conclusion

PromptLab shows that small, fine-tuned language models can support practical prompt improvement workflows through optimization, scoring, and guided prompt generation. By combining LoRA-adapted SmoLM2 models with a lightweight React and Flask interface, the system provides users with interactive feedback on prompt quality and produces more structured prompts from either weak drafts or vague ideas.

A potential next step would be to improve both model quality and usability by expanding training data, improving scoring calibration, and reducing inference latency. Future versions could support richer prompt rubrics, stronger sentence-level refinement, better evaluation against human judgments, and more robust generalization across domains. With these improvements, PromptLab could become a more reliable assistant for both everyday users and developers who want to design effective prompts at scale.

REFERENCES

- [1] H. et al. Allal. 2025. SmoLLM2: A Family of Efficient Small Language Models. arXiv preprint arXiv:2502.02737. (2025).
<https://arxiv.org/abs/2502.02737>
- [2] T. B. et al. Brown. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33.
<https://arxiv.org/abs/2005.14165>
- [3] C. et al. Burges. 2005. Learning to Rank Using Gradient Descent. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. DOI:<http://dx.doi.org/10.1145/1102351.1102363>
- [4] Google DeepMind. 2023. Gemini: A Family of Highly Capable Multimodal Models. arXiv preprint arXiv:2312.11805. (2023).
<https://arxiv.org/abs/2312.11805>
- [5] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
<https://arxiv.org/abs/2309.08532>
- [6] E. J. et al. Hu. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *Proceedings of ICLR*.
<https://arxiv.org/abs/2106.09685>
- [7] Subhajit et al. Khanna. 2025. GREATERPROMPT: A Unified, Customizable, and High-Performing Open-Source Toolkit for Prompt Optimization. arXiv preprint arXiv:2504.03975. (2025).
<https://arxiv.org/abs/2504.03975>
- [8] Omar et al. Khattab. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. arXiv preprint arXiv:2310.03714. (2023).
<https://arxiv.org/abs/2310.03714>
- [9] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *Comput. Surveys* 55, 9 (2023), 1–35. DOI:<http://dx.doi.org/10.1145/3560815>
- [10] Rithesh Murthy, Ming Zhu, Liangwei Yang, Jieliu Qiu, Juntao Tan, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025. Promptomatrix: An Automatic Prompt Optimization Framework for Large Language Models. arXiv preprint arXiv:2507.14241. (2025).
<https://arxiv.org/abs/2507.14241>
- [11] L. et al. Ouyang. 2022. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35.
<https://arxiv.org/abs/2203.02155>
- [12] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of EMNLP*.
<https://arxiv.org/abs/2010.15980>
- [13] Z. et al. Wang. 2024. HelpSteer2: Open-Source Dataset for Training Top-Performing Reward Models. arXiv preprint arXiv:2406.08673. (2024).
<https://arxiv.org/abs/2406.08673>
- [14] J. et al. Wei. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35. <https://arxiv.org/abs/2201.11903>

- [15] Marvin et al. Wessel. 2025. Prompt-with-Me: In-IDE Structured Prompt Management for LLM-Driven Software Engineering. arXiv preprint arXiv:2509.17096. (2025).
<https://arxiv.org/abs/2509.17096>
- [16] Y. et al. Yang. 2023. Large Language Models as Optimizers. arXiv preprint arXiv:2309.03409. (2023).
<https://arxiv.org/abs/2309.03409> BPO Dataset.
- [17] J. D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. DOI:<http://dx.doi.org/10.1145/3544548.3581388>

Appendix

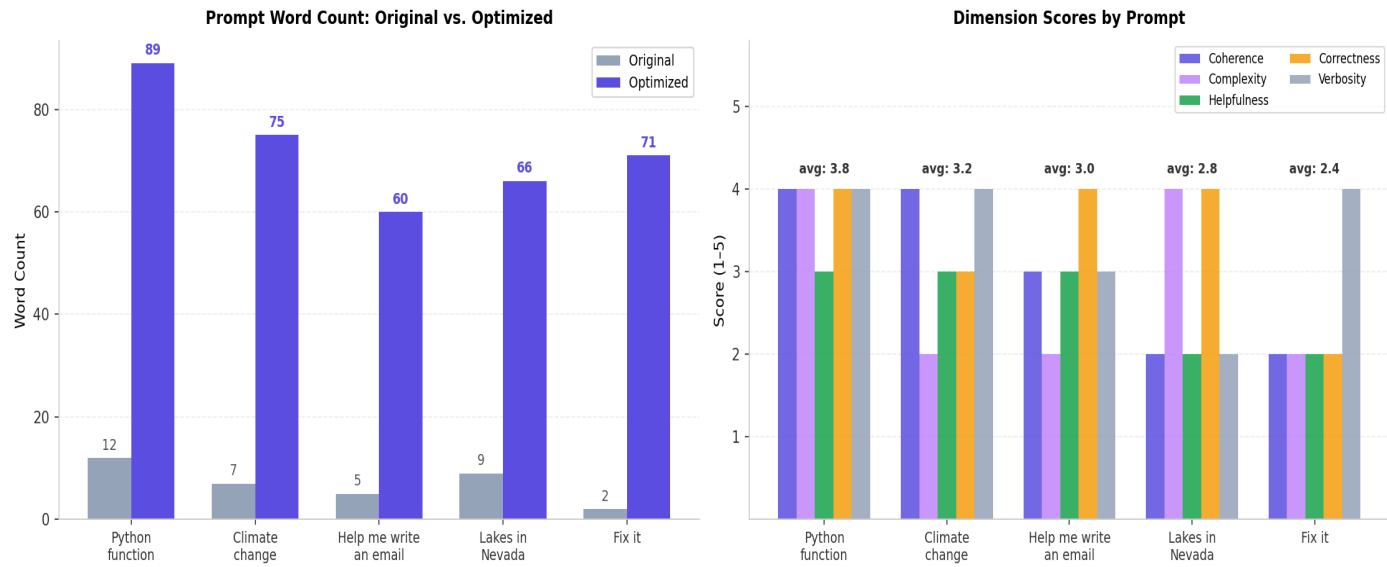


Figure 1: Left: Word count before and after prompt optimization across test prompts. Right: Mean HelpSteer dimension scores (helpfulness, correctness, coherence, complexity, verbosity) for original vs. optimized prompts.

LoRA Fine-Tuning — Training Loss Curves (SmoLLM2-360M)

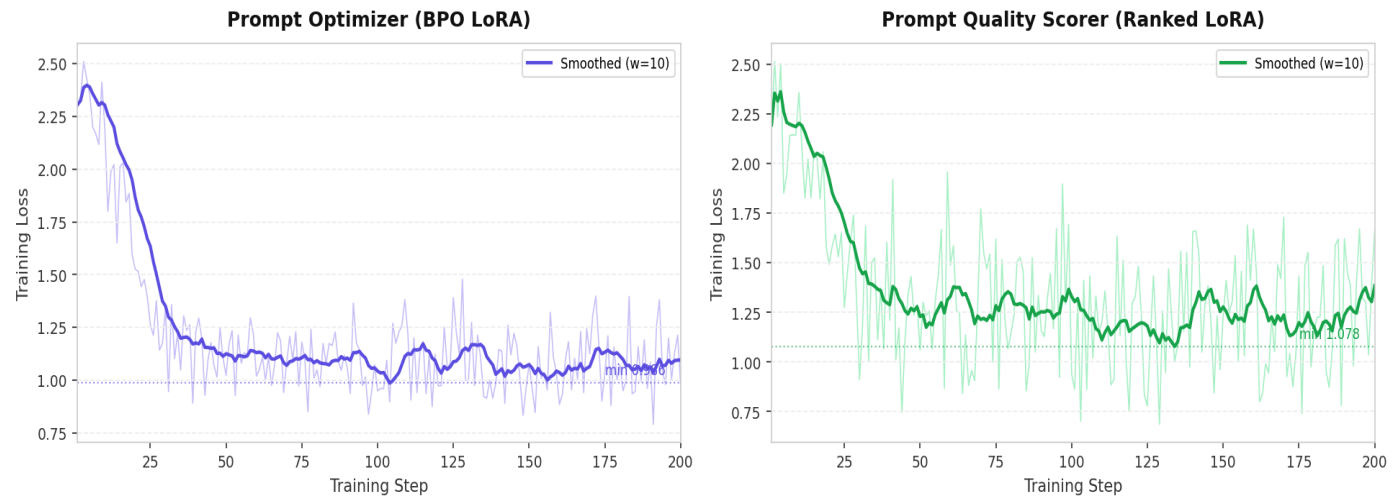


Figure 2: Smoothed training loss curves ($w=10$) for BPO LoRA (purple) and Ranked LoRA (green) adapters over 200 training steps.

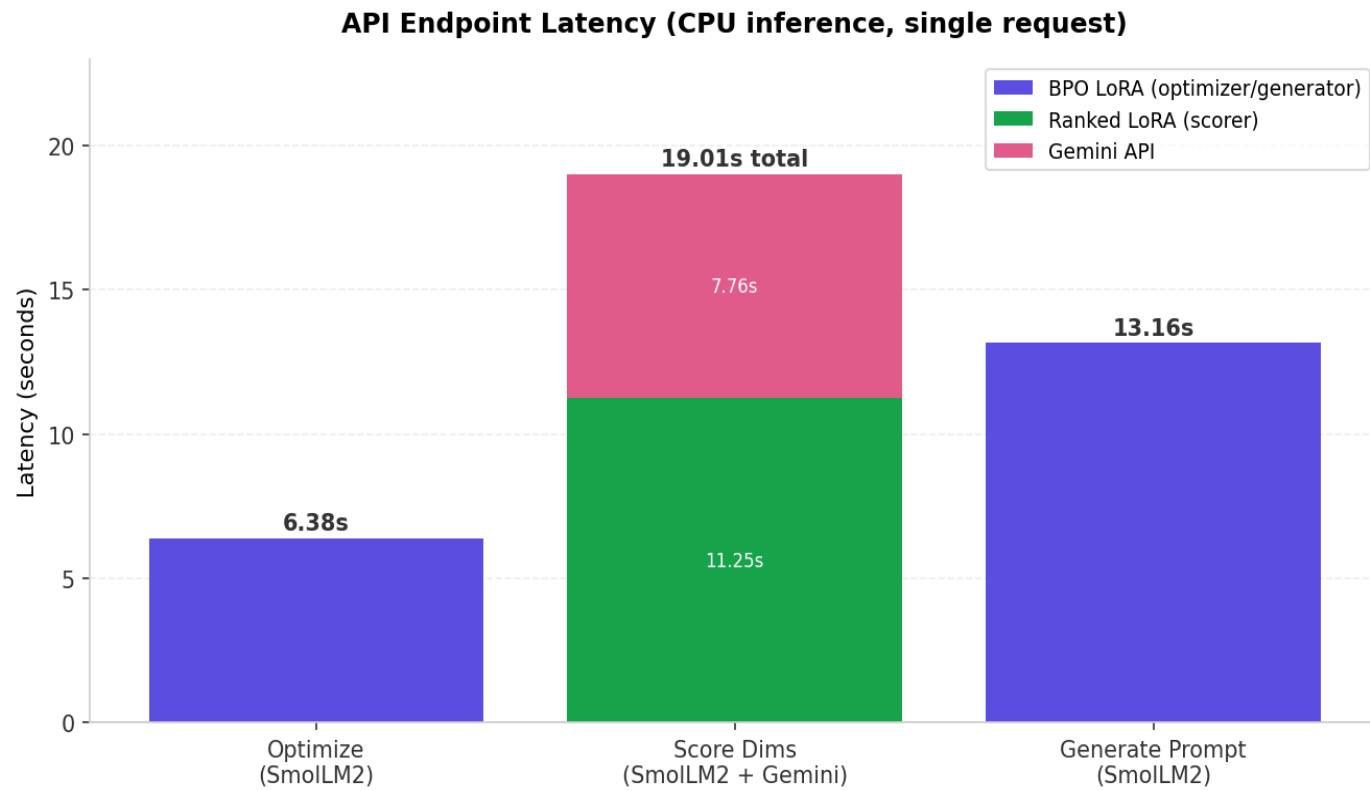


Figure 3: API endpoint latency breakdown. Score Dims shows stacked SmoLM2 (Ranked LoRA, green) and Gemini API (pink) components.

Input Prompt	PromptLab Output
Create a Python function that sorts a list of numbers and explain each step with comments.	Create a Python function named <code>sort_numbers</code> that takes a list of integers as input and returns a sorted version of the list. Explain each step using comments, including the following steps: 1. Initialize variables. 2. Iterate through the list. 3. Compare current and previous elements. 4. Swap if necessary. 5. Return the sorted list.
Describe the causes and effects of climate change.	Describe the causes and effects of climate change, including both natural factors and human activities that contribute to its rise. Explain how these factors interact and impact ecosystems, weather patterns, sea levels, and global temperatures. Provide examples of significant impacts on local communities, economies, and wildlife habitats. Offer suggestions for mitigating and adapting to climate change.
Help me write an email.	Write a professional email to [recipient] regarding [subject]. The email should be concise, clear, and respectful in tone. Include a greeting, body paragraph explaining the purpose, and a closing statement. Ensure the email is no longer than 200 words.
Write me a list of lakes in Nevada for fishing.	Write me a list of lakes in Nevada for fishing. Please include information about their location, size, depth, water temperature, and any notable features. Also provide details on fish caught per day and any regulations regarding lake usage.
Fix it.	Fix the following prompt by providing a detailed explanation of how to improve its structure and clarity while adhering to specific constraints such as specifying the type of feedback required, including both constructive criticism and suggestions for improvement, and ensuring that the response contains at least three bullet points highlighting key areas needing attention.

Table 1: PromptLab optimized outputs across all five representative input prompts.

Input Prompt	PromptLab	GPT-4o	Claude
Coding task.	Create a Python function named <code>sort_numbers</code> that takes a list of integers and returns a sorted version. Explain each step using comments including initialization, iteration, comparison, swapping, and return.	Write a Python function that takes a list of numbers and returns it sorted in ascending order. Include detailed in-line comments explaining what each line does and why it is necessary.	Write a Python function called <code>sort_numbers</code> that returns a new sorted list without modifying the original. State the algorithm used, include a docstring with params, return type, and time complexity, and explain the why behind key steps.
Climate change.	Describe the causes and effects of climate change, including natural factors and human activities. Explain how these factors impact ecosystems, weather patterns, sea levels, and temperatures. Provide examples of impacts on communities and wildlife, and offer mitigation suggestions.	Provide a concise, well-structured explanation of the primary human and natural causes of climate change, followed by a clear description of its major environmental, economic, and societal effects.	Write a factual explainer on climate change for an educated non-scientist. Cover human vs. natural causes, 3–4 major physical effects with timescales, and key societal impacts. Use plain language and flag anything still scientifically debated.
Fix it.	Fix the following prompt by explaining how to improve its structure and clarity, specifying the type of feedback required, including constructive criticism and suggestions, and ensuring at least three bullet points highlighting key areas needing attention.	Identify the specific issue in a piece of text, code, or explanation that I provide, correct it, and offer a clear improved version with a brief explanation of what was changed and why.	<i>Refused (too vague)</i>

Table 2: Qualitative comparison of prompt rewrites across PromptLab, GPT-4o, and Claude on three representative input prompts.