AI-Based Prompt Optimizer, Evaluator & Multi-LLM Comparison Platform

Mrs. Radhika S N
Anvita R, Archana R, Inchara S N, Arundhathi C V
Department of Computer Science and Engineering JNN College of Engineering, Shivamogga, Karnataka, India
radhikasn@jnnce.ac.in

***Abstract*—** Writing effective prompts for large language models (LLMs) is still mostly manual trial and error, and judging how different providers answer the same prompt usually means pasting it into several chat windows with no shared, repeatable scoring. This paper describes a full-stack platform that automates both tasks. It first scores a prompt on ten deterministic, rule-based quality dimensions. It then rewrites the prompt over a configurable number of rounds using an LLM-driven strategy, re-scoring after each round. The final prompt is sent concurrently to OpenAI, Google Gemini, Anthropic Claude and Groq, with each provider isolated so that one failure cannot block the rest. Responses are evaluated using reference-based lexical metrics (ROUGE-L, BLEU, TF-IDF cosine similarity), multi-run consistency, a latency- and token-based efficiency score, and an LLM-as-a-judge component, merged into a weighted composite score. Every experiment, including per-round history and per-provider outputs, is stored in a relational database and shown on an interactive analytics dashboard with CSV/JSON export. The backend uses FastAPI and Python, the frontend uses React and TypeScript, and an automated test suite exercises the analyzer, evaluation metrics and REST API without needing live credentials. The result is a reproducible, explainable and extensible framework for comparing prompt-engineering strategies and LLM providers on equal terms.

***Keywords*—** *Prompt Engineering; Large Language Models; Prompt Optimization; Multi-LLM Benchmarking; LLM-as-a-Judge; ROUGE; BLEU; FastAPI; React*

I. INTRODUCTION

How well a large language model (LLM) performs a task depends on the model itself and, to a large extent, on the wording of the prompt that instructs it [1]. In practice, prompt engineering remains a largely informal activity: practitioners rewrite a prompt, judge by eye whether the output seems better, and repeat, without a consistent method or a saved record of what changed and why.

A closely related problem is choosing a provider. Many commercial LLM services now offer broadly similar abilities at different prices, speeds and quality levels, so teams often paste one prompt into several separate interfaces and compare the answers informally. The two problems reinforce each other: a prompt tuned for one provider's instruction-following style may perform worse on another, so prompt quality and provider choice cannot be tuned in isolation.

Prior work shows that an LLM can propose better instructions itself, as in Automatic Prompt Engineer (APE) [1] and iterative black-box optimizers [2]. It also shows that a strong LLM acting as a judge can approximate human preferences at far lower cost than human evaluation [3]. Classical metrics such as BLEU [4] and ROUGE [5] remain cheap, reference-based signals when a gold answer exists. However, few openly documented systems combine deterministic prompt analysis, multi-round LLM optimization, concurrent multi-provider generation and configurable multi-metric evaluation in one persisted, reproducible platform. This paper presents such a system.

The platform follows a linear pipeline: PROMPT → ANALYZE → OPTIMIZE → GENERATE → EVALUATE → COMPARE → INSIGHTS. A rule-based analyzer first rates the raw prompt on ten explainable dimensions (clarity, specificity, context, task definition, constraints, output format, ambiguity control, role definition, examples and instruction structure) without calling any external model, so a baseline score is always available at zero API cost. An LLM-driven optimizer then rewrites the prompt over a set number of rounds, applying only the strategies that address the weaknesses the analyzer found, and re-scores each round with the same analyzer so that progress is comparable. The resulting prompt goes in parallel to any mix of OpenAI, Google Gemini, Anthropic Claude and Groq-hosted models; a per-provider error boundary ensures that a failure, rate limit or authentication error is reported for that provider alone. Each response is scored with a configurable-weight composite of lexical similarity, multi-run consistency, efficiency and a judge score, and every experiment is saved for later comparison, export and trend analysis.

The objectives of this work are to: (i) score prompt quality with explainable, deterministic heuristics instead of an opaque model; (ii) improve prompts automatically over configurable rounds and named strategies while preserving the original task intent; (iii) benchmark several providers on an identical prompt, concurrently, with isolated failure handling and real (not simulated) latency, token and cost figures; (iv) evaluate responses with a mix of reference-based and reference-free metrics merged into one auditable composite score; and (v) persist every experiment so that strategies and providers can be compared empirically over time rather than anecdotally.

II. LITERATURE SURVEY

We reviewed recent work on automated prompt optimization and LLM evaluation to position the proposed system.

Zhou et al. [1] introduced APE, which treats an instruction as a program that an LLM searches for and selects to maximize a score function. Instructions found this way matched or beat human-written ones on most tested tasks, showing that an LLM

can act as a prompt-rewriting engine, although the search needs many candidate generations and a held-out scoring set. Yang et al. [2] extended this with OPRO, which frames prompt improvement as a black-box optimization loop: the LLM proposes new prompts conditioned on earlier prompts and their scores. This improves sample efficiency over undirected search but still needs many optimizer calls per task.

On the evaluation side, Papineni et al. [4] proposed BLEU, an n-gram precision measure with a brevity penalty that remains a cheap proxy for surface similarity to a reference. Lin [5] proposed the recall-oriented ROUGE family, of which ROUGE-L (longest common subsequence) is widely used for summarization and generation. Both are fast and reproducible, but they correlate only imperfectly with human views of semantic quality, especially for open-ended output. Zheng et al. [3] addressed this gap with LLM-as-a-judge, showing that a strong model such as GPT-4 agrees with human preferences about as often as humans agree with each other. They also documented systematic judge biases (position, verbosity and self-enhancement bias), which is why judge scores here are treated as one signal among several rather than as ground truth.

Related research uses LLMs inside search and design loops. Wei et al. [10] showed that chain-of-thought prompting improves reasoning, Romera-Paredes et al. [6] used program search with LLMs to make mathematical discoveries, and Liu et al. [7], van Stein and Bäck [8] and Wu et al. [9] applied LLMs to heuristic design, metaheuristic generation and algorithm selection. A survey on LLM-driven meta-optimizers [11] and applied studies of prompt design for materials-data and medical information extraction [12], [13] and for literature review [14] further show how prompt wording affects results across fields. Work on evolving deeper LLM reasoning [15] points the same way.

Across this literature, LLM-driven prompt rewriting, lexical reference-based metrics and LLM-as-a-judge evaluation have mostly been released as separate artifacts or single-purpose scripts. Few documented systems combine a free deterministic baseline scorer, multi-round optimization, concurrent failure-isolated benchmarking and a configurable-weight composite evaluation behind one web application with persisted history. That gap motivates this work.

III. PROPOSED METHODOLOGY

The platform is a seven-stage pipeline: prompt analysis, LLM-driven optimization, multi-provider generation, response evaluation, cross-provider comparison, experiment persistence and dashboard insight. It is implemented as a FastAPI backend and a React/TypeScript single-page frontend that talk over a versioned REST API. Every API call, score, token count and latency shown by the platform comes from a real measurement or provider response. When a value cannot be computed, for example the cost of a model missing from the local pricing table, the system reports it as unavailable instead of estimating it.

A. Prompt Quality Analysis

The analyzer is a deterministic, rule-based scorer that needs no external API call and therefore adds no latency or cost. For a raw prompt it computes ten independent scores on a 0–100 scale: clarity (penalizes vague filler words and overlong sentences), specificity (rewards concrete numbers and named entities), context (background markers or adequate length), task definition (a clear leading action verb), constraints (explicit limits on length, tone or scope), output format (explicit requests such as JSON, tables or bullet lists), ambiguity control, role definition (a persona assigned to the model), examples (illustrative input/output pairs) and instruction structure (numbered or bulleted multi-step instructions). Each dimension also produces a short, human-readable strength, weakness or suggestion. The optimizer reuses these notes unchanged, so rewriting targets the weaknesses actually detected instead of applying generic advice.

B. LLM-Driven Prompt Optimization

The optimizer runs a configurable number of rounds (1–10). Round zero scores the original prompt to set a baseline. In each later round the current prompt is re-analyzed and a rewrite instruction is built from only the strategies the user selected (clarity, specificity, context, role assignment, constraints, output formatting, few-shot examples, ambiguity reduction and/or instruction structuring) plus the weaknesses most recently found. The instruction also tells the model to keep the original task intent and topic and to return only the rewritten prompt text, and it is sent to a user-selected LLM provider. The returned prompt is re-scored by the same deterministic analyzer from Section III-A, so improvement is measured on one consistent scale rather than by the rewriting model's own, possibly inflated, self-assessment. The full per-round history (prompt text, composite score, per-dimension scores and applied strategies) is kept so that a side-by-side diff can be inspected round by round.

C. Concurrent Multi-Provider Generation

The optimized prompt, or the original if optimization is skipped, can be sent to any subset of four providers (OpenAI, Google Gemini, Anthropic Claude and Groq) through a common provider-agnostic client interface. All selected providers run concurrently using asynchronous I/O instead of one after another. Each call sits inside its own error boundary: an authentication failure, rate limit or network error is caught, classified and reported as that provider's status without cancelling the other in-flight calls. The platform can also run the same prompt several times per provider to collect a small sample for consistency scoring. Input and output token counts and latency are read directly from each provider's response. Estimated cost is computed only when the resolved model appears in a maintained per-provider pricing table; otherwise it is shown as unavailable.

D. Response Evaluation

Each response is scored on up to four independent axes, combined into a configurable-weight composite score (weights are validated on the server to total 100%):

Correctness / similarity: when a reference answer is supplied, the response is compared with it using ROUGE-L (F-measure over the longest common subsequence) [5], BLEU with modified n-gram precision and a brevity penalty [4], and TF-IDF cosine similarity.

Consistency: when several runs of the same provider and prompt exist, pairwise similarity between the sampled responses estimates output stability.

Efficiency: a bounded score derived from observed latency and total token count, favoring faster and more token-efficient responses without penalizing correctness.

LLM-as-a-judge: a structured-JSON call to a user-selected judge model [3] rates the response on task-relevant criteria, optionally using the same reference answer as the lexical metrics.

An axis that cannot be computed (for example, no reference answer, or only one run) is left out of the weighted sum instead of being counted as zero, so the composite always reflects only the evidence available for that response.

E. Cross-Provider Comparison and Persistence

Because generation (Section III-C) and evaluation (Section III-D) are applied the same way to every selected provider, the platform can show a like-for-like table of response text, latency, token usage, estimated cost and composite score per provider. Every experiment, including the original prompt, optimization history, per-provider outputs and evaluation results, is stored in a relational database (SQLite via SQLAlchemy) with a normalized schema of experiments, optimization rounds, provider outputs and evaluations. This supports later retrieval, CSV/JSON export and deletion.

F. Monitoring Dashboard and System Architecture

A dashboard built with React, TypeScript and Recharts shows live provider connectivity, experiment volume over time, average composite scores and per-provider latency and cost trends drawn from the stored history. Users can therefore see which optimization strategies and providers work best for their workload without reading raw database records.

The backend is written in Python, with FastAPI for the REST layer, Pydantic for request/response validation and SQLAlchemy over SQLite for persistence. Each provider has its own client module built on that provider's official SDK, all unified behind one interface and created by a single provider-selecting factory. The factory also offers a clearly labeled mock provider so that the analyzer, optimizer, evaluator and API can be tested end to end without network access or credentials; the platform never substitutes mock output for a real call while in real mode. The frontend is a React/TypeScript single-page application built with Vite and styled with Tailwind CSS, with dedicated pages for the optimizer, multi-provider comparison, standalone evaluation, experiment history, analytics and settings, communicating with the backend through a typed fetch wrapper. Because the pipeline relies on documented REST endpoints and standard SDKs, the design is modular, testable and extensible, with automated PDF reporting and ablation-style experiments as future work.

IV. CONCLUSION

This paper presented the design of an AI-based prompt optimizer, evaluator and multi-LLM comparison platform. It combines a deterministic, explainable prompt-quality analyzer, a multi-round LLM-driven optimizer and concurrent, failure-isolated benchmarking across OpenAI, Google Gemini, Anthropic Claude and Groq. By scoring every response with a configurable-weight composite of reference-based lexical similarity, consistency, efficiency and LLM-as-a-judge results, and by saving every experiment for later comparison, the approach aims to replace ad hoc, unrecorded prompt trial and error with a reproducible and auditable workflow. It also provides a scalable base for future additions such as automated PDF reporting, ablation-study analysis and self-consistency majority voting in prompt-engineering tools.

ACKNOWLEDGMENT

The authors sincerely thank our guide, [GUIDE NAME – confirm], Assistant Professor, Department of Computer Science and Engineering, for constant guidance, insightful suggestions and continued support throughout this project. We are equally grateful to the Head of the Department of Computer Science and Engineering and the Principal of JNN College of Engineering, Shivamogga, for their steady support and encouragement. We also thank our families and friends, and gratefully acknowledge JNN College of Engineering and its Department of Computer Science and Engineering for the resources, technical knowledge and motivation that made this project possible.

REFERENCES

[1] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, “Large language models are human-level prompt engineers,” in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.

[2] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, “Large language models as optimizers,” in *Proc. ICLR*, 2024.

[3] L. Zheng, W.-L. Chiang, Y. Sheng, et al., “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[4] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: a method for automatic evaluation of machine translation,” in *Proc. 40th Annu. Meeting of the ACL*, 2002, pp. 311–318.

[5] C.-Y. Lin, “ROUGE: a package for automatic evaluation of summaries,” in *Text Summarization Branches Out*, ACL, 2004, pp. 74–81.

[6] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, et al., “Mathematical discoveries from program search with large language models,” *Nature*, vol. 625, pp. 468–475, 2024, doi: 10.1038/s41586-023-06924-6.

[7] F. Liu, X. Tong, M. Yuan, X. Lin, et al., “Evolution of heuristics: towards efficient automatic algorithm design using large language model,” in *Proc. 41st Int. Conf. Machine Learning (ICML)*, 2024.

[8] N. van Stein and T. Bäck, “LLaMEA: a large language model evolutionary algorithm for automatically generating metaheuristics,” *IEEE Trans. Evol. Comput.*, 2024, doi: 10.1109/TEVC.2024.3402686.

[9] X. Wu, Y. Zhong, J. Wu, F. Liu, and Q. Zhang, “AS-LLM: when algorithm selection meets large language model,” arXiv:2307.08784, 2023.

[10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24824–24837.

[11] Y. Zheng, L. Zhang, K. Li, R. Wang, W. Li, T. Zhang, Q. Zhang, and Y. Jin, “A survey on large language models driven meta-optimizers for automated intelligent optimization,” *Artif. Intell. Rev.*, vol. 59, no. 72, 2026, doi: 10.1007/s10462-025-11470-w.

[12] M. P. Polak and D. Morgan, “Extracting accurate materials data from research papers with conventional language models and prompt engineering,” *Nat. Commun.*, vol. 15, no. 1569, 2024, doi: 10.1038/s41467-024-45563-x.

[13] Y. Tang, et al., "Large language model in medical information extraction from titles and abstracts with prompt engineering strategies: a comparative study of GPT-3.5 and GPT-4," medRxiv, 2024, doi: 10.1101/2024.02.14.24304572.

[14] Y. Lee, J. H. Oh, D. Lee, M. Kang, and S. Lee, "Prompt engineering in ChatGPT for literature review: practical guide exemplified with studies on white phosphorus," *Sci. Rep.*, vol. 15, no. 15310, 2025, doi: 10.1038/s41598-025-99423-9.

[15] K. H. Lee, I. Fischer, Y. H. Wu, et al., "Evolving deeper LLM thinking," arXiv:2501.09891, 2025.