CoolPrompt: Automatic Prompt Optimization Framework for Large Language Models

Nikita Kulin, Viktor Zhuravlev, Artur Khairullin, Alena Sitkina, Sergey Muravyov
ITMO University
Saint-Petersburg, Russia
{242106,334857,368983,367537}@niuitmo.ru, smuravyov@itmo.ru

***Abstract*—The effectiveness of Large Language Models (LLMs) is highly dependent on the design of input prompts. Manual prompt engineering requires a domain expertise and prompting techniques knowledge that leads to a complex, time-consuming, subjective, and often suboptimal process. We introduce CoolPrompt as a novel framework for automatic prompt optimization. It provides a complete zero-configuration workflow, which includes automatic task and metric selection, also splits the input dataset or generates synthetic data when annotations are missing, and final feedback collection of prompt optimization results. Our framework provides three new prompt optimization algorithms ReflectivePrompt and DistillPrompt that have demonstrated effectiveness compared to similar optimization algorithms, and a flexible meta-prompting approach called HyPE for rapid optimization. Competitive and experimental results demonstrate the effectiveness of CoolPrompt over other solutions.**

I. Introduction

Large Language Models (LLMs) such as GPT-4 [1], Claude AI, DeepSeek [2], Grok, and LLaMA [3] have revolutionized artificial intelligence transitioning from task-specific solutions to general-purpose foundation models [4], [5] and driving their rapid adoption across research and industry [6]. They have exhibited unprecedented effectiveness due to their remarkable performance in natural language understanding [7], text generation [8], [9], code generation [10], [11], and reasoning [12]. Meanwhile their operational efficacy is fundamentally mediated by the quality of prompt design [13], where prompts serve as computational directives from human to model [14].

Prompt engineering is the practice of designing input instructions to elicit desired model behavior and has emerged as a critical and rapidly evolving discipline [15], [16]. The process involves creating prompts, which can include questions, instructions, or templates that use the embedded knowledge of the model to maximize performance on a task. Unlike traditional fine-tuning, prompt engineering does not require modifying the model's weights; instead, it leverages the model as a fixed generalist 'language computer'. Prompt engineering methods range from simple input templates such as few-shot techniques [17] to advanced strategies such as Chain-of-Thought prompting [18], Self-Discover [19], Tree-of-Thoughts [20], ReAct [21] and etc. [15] Moreover, recent advances have enabled LLMs to self-generate and iteratively refine their own prompts through in-context learning and reinforcement signals, automating aspects of the prompt design process [22].

Manual prompt engineering remains fraught with challenges that limit its potential, performance, scalability, and accessibility.

1) Designing high-performance prompts typically requires extensive trial-and-error, deep domain expertise, and prompting techniques knowledge; therefore the process is often time-consuming and nonsystematic.
2) Although current LLMs are trained in human-generated text data, the effectiveness of prompt generation is also influenced by factors such as input and output format [23], placement of few-shot examples [24], the use of key trigger words and tokens [25], [26], and the elimination of redundant tokens and words [27]. Consequently, these factors reduce the relative importance of semantic clarity in human-oriented content and narratives, thus slowing down the process of manual prompt design.
3) Prompt effectiveness often exhibits poor transferability across tasks, datasets, and even different LLM architectures [28]–[30], undermining reproducibility and scalability and requiring additional time and resources to refine prompts.

One of the most profound advances enabled by LLMs is the development of automatic prompt optimization (autoprompting), the use of algorithms and optimization strategies to automate the design, selection, and refinement of prompts supplied to language models [31]. Autoprompting leverages methods such as llm-based and planning approaches [30], [32], reinforcement learning [33]–[35], evolutionary algorithms [27], [36], [37], and meta-optimization [36], [38], [39] to optimize prompts. Studies show that automatic prompt optimization can achieve higher efficiency, consistency and scalability even with manual prompting by experts [30]; it reduces human workload while improving the robustness between tasks and generalization of prompt strategies.

Despite these advances, current autoprompting methods still have several drawbacks. First, many current autoprompting implementations are tailored to proprietary LLMs, making it difficult to use custom or open-source models for specific tasks, which reduces the democratization of LLM selection and usage [30], [37]. Second, the stage of prompt engineering remains costly, as there is no intuition or universal methods and strategies in selecting prompting and autoprompting methods, and the complexity of evaluating the problem for

specific data increases the barrier to prompt engineering [15], [31]. Third, rapid efficacy system evaluation requires comprehensive evaluation methodologies incorporating specialized testing frameworks, domain-adapted performance metrics, and statistically significant experimental designs, collectively imposing substantial computational and temporal resource requirements [40]. Furthermore, conventional prompt engineering approaches exhibit limited generalizability in various task domains and applications [40].

To address these fundamental limitations, we introduce **CoolPrompt**, a comprehensive automatic prompt optimization framework that serves as an alternative to manual prompt design, providing a complete workflow from task definition to prompt evaluation. This framework offers a quick start to prompt optimization with zero expertise and minimal prompt engineering requirements. **CoolPrompt** includes automatic task and metric selection for task assessment, splitting the input dataset or generating synthetic data when annotations are missing, and final feedback collection of prompt optimization results. Our framework includes two new innovative autoprompting algorithms: ReflectivePrompt [41] and DistillPrompt [42] that have demonstrated effectiveness compared to similar solutions and a flexible meta-prompting approach called HyPE for rapid optimization.

CoolPrompt allows machine learning and prompt engineers, researchers, and practitioners to take advantage of state-of-the-art prompt-based optimization without requiring deep knowledge of the inner workings of LLMs or optimization algorithms [43]. Beyond its technical contributions, this work addresses the main challenges of ensuring accessibility when deploying LLMs, removing expert barriers, and offering intuitive interfaces. This standardization is key to democratizing and accelerating the adoption of LLMs in industries and research areas where operational engineering knowledge is limited but the potential for application is high.

Our key contributions are as follows:

1) We present a zero-configuration framework that advances a wide range of LLMs and automates the full prompt optimization pipeline as an alternative manual prompting design, from task definition to automatic prompt evaluation and feedback.
2) We propose synthetic data generation that eliminates data bottlenecks in prompt optimization.
3) We propose a wide LLM inference backends choice using LangChain to democratize access to variety of models and its optimizations.
4) Our framework-agnostic design supports several optimization strategies for short-term optimizations: meta-prompting approach HyPE, and for long-term optimizations: autoprompting algorithms ReflectivePrompt and DistillPrompt.
5) We conduct comprehensive evaluation across five task categories, showing consistent improvements over current approaches with reduced computational overhead.

The experimental studies show that CoolPrompt achieves competitive performance on different tasks such as mathematical reasoning, question answering, classification, summarization, and natural language understanding. Its cost-aware optimization further allows users to tailor performance-efficiency trade-offs, validating both its practical utility and generalizability.

II. Related Work

A. Prompting Techniques

Recent developments in prompt engineering have shown significant advances in prompt design techniques [13]. For example, Few-shot [17] prompting provides instructive examples to guide the model. Chain-of-Thought prompting [18] has proven effective by generating the model's reasoning process before arriving at the final answer. Building on this, more advanced reasoning prompt designs have emerged. Self-Discover [19] selects pre-existing reasoning chains, adapts them to the specific task, and applies them directly. Self-Consistency [19] samples multiple reasoning paths and implements them to produce the most consistent answer. Tree-of-Thoughts [20] and Graph-of-Thoughts [44] generate various decomposed reasoning variations, which are then evaluated and selected, thus increasing the depth of the exploration. ReAct [21] goes further by generating reasoning that translates into actions, while reflecting on previous steps; it is commonly integrated within Retrieval-Augmented Generation (RAG) [21] pipelines and agent-based systems.

Recent research has also investigated self-critique methods to minimize risks of hallucinations such as Chain-of-Verification [45] and Self-Refine [46], as well as agentic prompting frameworks that empower LLMs to operate autonomously with tool-use capabilities. In addition, multimodal prompting techniques have extended prompt engineering beyond text to include image [47], [48], audio [49], video [50], and segmentation prompting [51].

B. Automatic Prompting Algorithms

Currently, a variety of auto-prompting algorithms have been developed, based on different optimization methods. Specifically, EvoPrompt [52] and PromptBreeder [53] employ an evolutionary approach, where a large language model (LLM) serves as a selection, mutation, or recombination operator. PromptAgent [54] and StablePrompt [55] utilize Reinforcement Learning (RL), optimizing prompts using a reward model. Solutions such as iPrompt [56] and OPRO [57] are built on LLMs or foundation models (FM), leveraging meta-prompts to modify the optimization pipeline. The primary motivation for exploring autoprompting comes from research on the Automatic Prompt Engineer [30], where it was demonstrated that modern LLMs can handle prompt generation and optimization tasks comparable to or even better than human experts.

C. Prompt Optimization Libraries

Current prompt optimization solutions offer a variety of functionalities and optimization modules. AdalFlow [58] provides an auto-differentiable framework that supports both zero-

shot and few-shot prompt optimization, along with rapid construction of LLM, RAG, and Agent pipelines. PromptWizard [59] enables automatic prompt optimization through prompt refinement and synthetic data generation. PromptFoo [60] offers prompt evaluation capabilities combined with a testing of LLM pipelines. Prompt-Promptor [61] optimizes and assesses candidate prompts using LLM agents. DSPy [62] introduces a model for prompt expansion and optimization based on structured approaches. PromptoMatrix [63] showcases an end-to-end prompt optimization pipeline that employs multiple strategies and evaluates performance on synthetic data.

In addition, the ecosystem includes numerous other frameworks, such as Anthropic's prompt optimization tool, LangChain Prompt Canvas, and Google's AI Studio, each focused on specific aspects of prompt optimization and providing user-friendly interfaces.

III. CoolPrompt

A. Architecture Overview

CoolPrompt is a comprehensive framework built on a Python backend, featuring a complete pipeline for automated prompt optimization. The system is designed for both direct usage and seamless integration with other platforms and systems. The complete framework architecture and user workflow, spanning from query submission to result generation, are presented in Fig. 1.

The architecture comprises several core functional modules.

1) **PromptTuner** is a primary interface class for parameter configuration and optimization pipeline execution.
2) **Evaluator** is a module for assessing prompt performance in datasets, incorporating multiple metrics for both classification and generation tasks.
3) **PromptOptimizer** is a versatile optimization module that supports short-term adaptations through prompt engineering techniques with meta-prompts and long-term automatic prompt optimization algorithms.
4) **PromptAssistant** is an LLM-based component with predefined meta-prompts to generate self-improvement feedback for users.
5) **Synthetic Data Generator** is an auxiliary module for synthetic data generation when no input dataset is provided.
6) **Task Detector** is an automated task classification component for scenarios without explicit user-defined task specifications.

B. Optimization Workflow

Figure 1 illustrates the sequential workflow steps of the CoolPrompt framework, which will be described in detail. During step 1, the user provides a start prompt for optimization initialization. Additionally, user may explicitly specify the following task objectives: task type (classification or generation), evaluation metric, PromptOptimizer configuration, LLM selection supported via LangChain integration for target LLM, denoted in Figure1 as LLM, and assistant LLM (denoted in Figure 1 as LLM-2), problem description, labeled dataset with target variables, and train-test split ratio.

Step 2 processes user configurations through automated fallback mechanisms when parameters are unspecified. The system employs: Task Detector for automatic task classification when task type is undefined; predefined task-specific metrics when evaluation criteria are unspecified; default configured LLM interfaces when no model is selected; LLM-generated problem descriptions when absent; Synthetic Data Generator module for dataset creation when unavailable; HyPE as the default PromptOptimizer; and predefined split ratios for dataset partitioning.

Steps 3-4 perform dataset validation, triggering synthetic dataset generation with corresponding labels via the Synthetic Data Generator when required, followed by train-test sampling. Step 5 evaluates the initial prompt on the training subset. Step 6 executes prompt optimization through the selected PromptOptimizer method. Step 7 assesses the final optimized prompt on test set.

The comparative evaluation between the initial and optimized prompts occurs during Steps 8-9. Step 10 employs the PromptAssistant component to generate self-improvement feedback by analyzing start and final prompt performance. The workflow concludes at Step 11 with delivery of pipeline results: comprehensive train-test evaluations with metrics and actionable prompt refinement feedback.

C. Prompt Optimization Methods

1) HyPE: HyPE (Hypothetical Prompt Enhancer) is a rapid meta-prompting approach for adaptive prompt enhancement that asks a large language model to generate a hypothetical instructive prompt which solves the same underlying task as the user's query. The design intentionally avoids multi-round prompt search or ensembles of hand-crafted transformation rules: instead, HyPE exploits the model's internal knowledge of effective prompting patterns to produce an immediately usable reformulation in one extra forward pass, giving lightweight, task-adaptive prompt optimization with minimal engineering.

The idea of HyPE is motivated by the HyDE method [64], which synthesizes a hypothetical document to improve the retrieval process. HyPE applies the same idea to prompt formulation rather than retrieval. Building on this concept, HyPE stands out from previous prompt optimization techniques. Methods like chain-of-thought prompting require task-specific exemplars, while automated strategies often depend on multi-step LLM calls or rule-based transformations, incurring substantial computational or engineering overhead. In contrast, HyPE's single-step generation leverages the model's inherent, pretrained understanding of instructional language.

HyPE relies on a single meta-prompt to guide its entire prompt optimization process, so the meta-prompt's quality is critical to the quality of generated prompts. Our current approach was to design a meta-prompt that directly reflects the method's purposes (see Fig. 2)

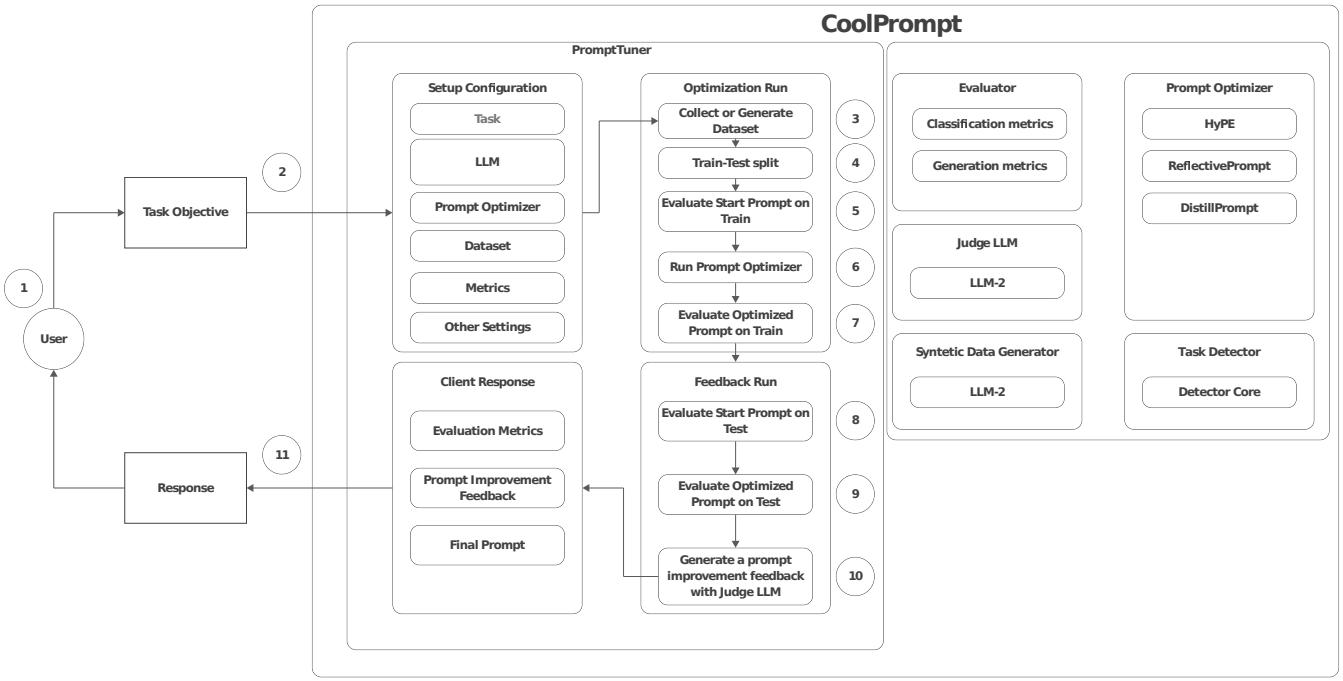

Fig. 1. CoolPrompt System Architecture and User Workflow

This intrinsic efficiency yields prompts that are both more generalizable and precise than the original query, effectively distilling the task's core requirements into an optimal instruction for the model itself, without the need for external search or exemplars.

2) ReflectivePrompt: ReflectivePrompt is an evolutionary-based prompt optimization method, which is built on the idea of Reflective Evolution [65]. Using the concepts of textual gradient [66] and self-reflection [67], it provides remarkable results in different areas of autoprompting tasks. All the data required to run the method: a dataset, a description of the target problem, and an initial user prompt. The remaining individuals of the first population are created by producing diverse paraphrases of the user prompt.

ReflectivePrompt implements two evolutionary operators: crossover and elitist mutation. They both leverage short-term and long-term reflections to improve their effectiveness. Crossover creates a new prompt from two parent individuals using generated short-term reflection. Initially, a set of parent pairs is sampled according to these rules:

1) Each prompt is selected with probability proportional to its fitness score.
2) One prompt can be selected in multiple parent pairs.
3) Within each pair, one prompt must have a strictly higher score than the other.

The difference in fitness scores is required to determine the superior and inferior prompts in each parent pair, since the short-term reflection is focused on identifying qualities and dissimilarities that yield higher-scoring prompts. Short-term reflection consists of the model generating reflective analyses and hints that are then used to achieve better crossover offspring. In summary, the set of short-term reflections constitutes an analytics of the individuals in the current population.

The elitist mutation operator generates new individuals using the best prompt in the current population and long-term reflection. This operator enables local search in the area of the present optimum. Long-term reflection is updated in each epoch based on its prior version and all short-term reflections produced in that generation. It contains a distilled summary of the model reasoning about the current population and accumulates knowledge across all epochs of evolution by incorporating its previous state.

3) DistillPrompt: DistillPrompt is a gradient-free automatic prompt optimization algorithm based on iterative prompt distillation. The method employs prompt compression, semantic reformulation, and dynamic example integration to enhance prompt effectiveness across diverse NLP tasks. This method is based on the idea of the Tree-of-Thoughts prompting technique. At each epoch, DistillPrompt uses the best prompt from the previous iteration according to the target metric. For the first epoch, the initial user prompt is employed.

The pipeline is as follows: for the first epoch, the initial user prompt is employed. Then, several variations of the initial prompt are generated. These diverse modifications are used to analyze the search space from different perspectives. The generated variations explore the search area in mostly blind and inefficient way, and in order to cope with this, the second step incorporates knowledge embedding. The objective is to specialize the prompt for the task while preserving its original formulation as much as possible. To achieve this, several examples are randomly sampled from the training dataset and

You are an expert prompt engineer. Your only task is to generate a hypothetical instructive prompt that would help a large language model effectively answer the following query. The prompt must solve the same underlying task as the original query while being more effective.
HARD CONSTRAINTS
1. LANGUAGE:
- Output MUST be in the EXACT SAME LANGUAGE as the query.
2. CONTENT:
- Output ONLY the hypothetical instructive prompt - do NOT answer the original query directly.
- The hypothetical prompt must solve the same task as the oiginal query provided by user.
- If the original query contains any code snippets, you must include it in final prompt.
3. TECHNICAL PRESERVATION:
- Code blocks must be preserved with original syntax and formatting.
- Variables, placeholders ({{var}}), and technical terms kept unchanged.
- Markdown and special formatting replicated precisely.
YOUR OUTPUT FORMAT
[PROMPT_START]<your hypothetical instructive prompt here>[PROMPT_END]
INPUT
User's query: {QUERY}
Problem description: {PROBLEM_DESCRIPTION}
OUTPUT
Hypothetical Instructive Prompt:

Fig. 2. Final meta-prompt for HyPE

provided to the model to extract some key principles and ideas that are necessary to solve these training examples. The created concepts are then embedded in the prompt.

However, there is a risk of the model "overfitting" to the given examples and embedding the provided questions and labels itself rather than generalizing for the whole task. To mitigate this, the next step involves instruction compression. The LLM reformulates each prompt into a small number of sentences, preserving the core content of both the original formulations and the embedded task-solving principles.

Since the examples from training data were sampled independently and randomly for each candidate, the resulting insights may vary. Thus, the natural progression is to merge the compressed candidates into a single distilled prompt, accumulating the collective ideas. The final stage generates the variations of the aggregated prompt, similar to the very first step, and then the new best prompt is selected from these newly created candidates.

D. Key Features

1) Interaction With LLMs: CoolPrompt supports comprehensive LLM integration, ranging from locally deployed open-source models to proprietary API-based solutions. For standardized LLM interfacing, we implemented LangChain due to its provider-agnostic architecture that abstracts model-specific implementations, optimization techniques, and API variations. This design constitutes a critical framework component that democratizes LLM selection for end-users while eliminating the need for custom interface adaptation, a notable limitation present in alternative prompt optimization libraries.

2) Prompt Improvement Feedback: Beyond prompt optimization capabilities, CoolPrompt enhances methodological transparency by providing users with constructive feedback containing actionable suggestions and composition insights. This functionality is implemented through the PromptAssistant module, which performs a comparative analysis between initial and optimized prompt versions. PromptAssistant generates an interpretation of prompt optimization results, thereby contributing to the development of users' technical proficiency in prompt engineering and to exploration of "efficient prompt pattern".

3) Synthetic Data Generator: Modern LLMs have demonstrated remarkable efficacy in the resolution of instructional tasks, allowing the generation of synthetic data complete with target annotations. CoolPrompt takes advantage of this capability to address critical bottlenecks in prompt evaluation. The generation process comprises the following steps: First, a problem description which was parsed from an initial prompt is included into a standardized meta-prompt with instructions to provide dataset samples in input-output formats. Second, received samples are expanded by generating hypothetical edge cases using another meta-prompt. Finally, PromptAssistant validates a dataset by selecting the most relevant samples. In order to control the sample size, we use structured output formatting.

4) Task Detector: Task Detector is a specialized component or module designed to work in conjunction with LLMs. Its primary function is to analyze the input of a user prompt and automatically identify the intent and the specific type of task the user wants the LLM to perform.

Instead of manual setup by user, LLM, which could be as target LLM, as PromptAssistant, identifies a task problem that uses for specifying a target metric.

IV. Experimental Evaluation

A. Experimental Setup

A comparison was conducted among automatic prompt optimization frameworks: CoolPrompt, Promptomatix, AdalFlow, and Promptify, with a manual zero-shot prompt used as the baseline and as a start prompt for optimization processes. The experiment was performed on five benchmark datasets:

1) **Question Answering**: SQuAD_2
2) **Mathematical Reasoning**: GSM8K
3) **Text Generation**: CommonGen

4) **Classification**: AG News
5) **Summarization**: XSum

The metrics were selected according to the dataset tasks: BERTScore for **SQuAD**, **CommonGen** and **XSum**; EM (Exact Match) for **GSM8K**; F1 Macro for **AG News**.

For each task in the evaluation experiment, we provided a training and validation data split of 30 samples from the original dataset with a train split ratio of 0.2. We chose the usage of original dataset because not every automatic optimization framework is capable for the generating syntetic data.

We evaluated each method in the comparison in 3 runs in order to obtain more objective results due to probabilistic LLM output generation, where Table I presents the average results across all runs. Prompt optimization methods DistillPrompt and ReflectivePrompt were run with a number of epochs of 5 and for the second method the prompt population size of 10. LLM run with the following generation parameters: temperature was 0.7 and maximum number of new tokens was 3500.

Temperature controls the randomness of the generated text, where low temperatures produce deterministic text and high temperatures foster greater creativity and diversity. We set temperature to 0.7 to ensure a variety of LLM responses and optimization runs.

Maximum number of new tokens constrains the limitation a size of generated tokens. In order to avoid overly restricting the model, we set this limit to 3500 tokens.

B. Results

Table I presents the average results across all runs. CoolPrompt demonstrated competitive metric scores across all datasets, outperforming the manual zero-shot prompt and surpassing other algorithms on average.

C. Competitive Analysis

Table II presents a comprehensive feature comparison between CoolPrompt and other frameworks, which includes the main key features for automatic prompt optimization.

V. Discussion

According results presented in Table I, CoolPrompt demonstrated competitive performance efficiency on the majority of benchmark tasks. The model's responses on the GSM8K and AG News datasets were evaluated based on exact match accuracy against the ground truth labels. On these datasets, the results obtained using CoolPrompt are comparable to those of other frameworks. For generative tasks, the prompts generated by CoolPrompt yielded superior results compared to other libraries.

The competitive analysis in Table II indicates that the most similar framework is Promptomatix, which is distinguished by its more limited options for selecting custom models compared to CoolPrompt. It is also worth mentioning the implemented criterion for the automatic selection of evaluation metrics. The current version of CoolPrompt supports the selection of various metrics, depending on previously chosen or automatically detected task type, with F1 and BertScore selected by default as the most representative metrics for providing a balanced assessment between model responses and target outputs.

VI. Future Work

This section outlines the current state and future development prospects of the CoolPrompt, designed for automatic prompt optimization in NLP tasks. CoolPrompt has several limitations and directions for further research to deal with them.

LLM Type. Instructive LLMs are considered the most suitable for the library's operation. The use of non-instructive models is considered inefficient due to their low responsiveness to instructions. Reasoning models present a distinct challenge due to increased computational costs, adversely affecting optimization speed. Their integration requires additional experimental research aimed at maximizing optimization efficiency and minimizing redundant generation.

Advanced Validation. CoolPrompt implements a standard set of evaluation metrics: for classification tasks (accuracy, recall, precision, F1); for text generation tasks (BLEU, ROUGE, METEOR, BERTScore). However, to adequately assess language model responses against specific criteria such as relevance, completeness, and toxicity, the implementation of specialized metrics that demonstrate high correlation with human evaluation (e.g., similar to the G-Eval metric [68]) is necessary.

Generalization to Other Modalities. The current library implementation is limited to the textual modality. Nonetheless, prompt engineering methods are applied in multimodal tasks, including Image Retrieval, Visual Question Answering (Visual QA), and Image Captioning. A promising direction for development is the research and creation of automatic prompt engineering algorithms adapted for multimodal domains.

Speed vs Quality Balance. To ensure high response speed, meta-prompting methods are employed; however, these are inferior in optimization effectiveness compared to slower, iterative algorithms. Finding the optimal compromise between operational speed and the quality of generated prompts is a non-trivial task requiring dedicated research to develop a balanced solution.

Synthetic Data Quality. The reliability and relevance of synthetic data generated during the optimization process are critically dependent on two factors: the competence of the assistant model and the precision of the task formulation. When operating within highly specialized domains (e.g., medicine, jurisprudence, biology), the model must utilize the corresponding terminology. Failure to meet this condition can lead to errors in data labeling, factual inaccuracies (hallucinations), or the generation of trivial examples lacking useful information.

VII. Conclusion

In this work, we have shown CoolPrompt a zero-configuration framework that automates the full prompt optimization pipeline as an alternative manual prompting design,

TABLE I. AUTOMATIC PROMPT OPTIMIZATION LIBRARIES PERFORMANCE ACROSS TASKS

Dataset	Metric	Manual 0-shot Prompt	Promptify	AdalFlow	Promptomatix	CoolPrompt
SQuAD_2	BertScore	0.875	0.905	0.920	0.918	**0.934**
GSM8K	EM	0.527	0.615	**0.753**	0.728	0.732
CommonGen	BertScore	0.871	0.885	0.904	0.902	**0.913**
AG News	F1	0.705	0.841	0.722	**0.858**	**0.858**
XSum	BertScore	0.823	0.233	0.841	0.857	**0.872**

TABLE II. FEATURE COMPARISON BETWEEN AUTOMATIC OPTIMIZATION LIBRARIES

Framework	Auto Data	Auto Task	Custom Model Usage	Zero Config	Feedback
Promptify	×	×	×	×	×
AdalFlow	×	×	×	×	×
Promptomatix	✓	✓	×	✓	✓
CoolPrompt	✓	✓	✓	✓	✓

demonstrating competitive effectiveness across diverse tasks. CoolPrompt represents a significant advancement in the field of automatic prompt optimization. We proposed the end-to-end prompt optimization pipeline with following key features as a wide LLM inference backends, syntetic data generation, task prediction, a bundle of optimization algorithms, and systematic prompt evaluation. Our evaluation results demonstrate competitive effectiveness across various tasks.

The principles embedded within automation, efficiency, and accessibility it as a key tool for the next generation of LLM-based applications. CoolPrompt plays a particularly important role in the context of the continuous evolution of language models, fostering broader participation in AI development and accelerating the adoption of these technologies across various subject domains and user communities by removing barriers in prompt design.

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, and et al., *Gpt-4 technical report*, *https://arXiv.org/abs/2303.08774*.

[2] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *CoRR*, 2024.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, and et al., *Llama: open and efficient foundation language models*, *https://arXiv.org/abs/2302.13971*.

[4] M. N. Vivekananda, P. A. Shidlyali, and V. V. Malgi, *Advancing artificial intelligence: insights into the applications and challenges of large language models*. IEEE, 2025.

[5] S. Yang, O. Nachum, Y. Du, J. Wei, P. Abbeel, and D. Schuurmans, *Foundation models for decision making: problems, methods, and opportunities*, *https://arXiv.org/abs/2303.04129*.

[6] Z. Zhao, W. Fan, J. Li, Y. Liu, X. Mei, Y. Wang, Z. Wen, F. Wang, X. Zhao, J. Tang, and et al., *Recommender systems in the era of large language models (llms)*. IEEE, 2024, vol. 36, no. 11.

[7] N. Karanikolas, E. Manga, N. Samaridi, E. Tousidou, and M. Vassilakopoulos, *Large language models versus natural language understanding and generation*. PCI, 2023.

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, and et al., "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[9] G. Bao, Y. Zhao, Z. Teng, L. Yang, and Y. Zhang, *Fast-DetectGPT: efficient zero-shot detection of machine-generated text via conditional probability curvature*. ICLR, 2023.

[10] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, and et al., *Evaluating large language models trained on code*, *https://arXiv.org/abs/2107.03374*.

[11] K. Zhang, J. Li, G. Li, X. Shi, and Z. Jin, *CodeAgent: enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024. [Online]. Available: https://aclanthology.org/2024.acl-long.737/

[12] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, and et al., "Solving quantitative reasoning problems with language models," *Advances in neural information processing systems*, vol. 35, pp. 3843–3857, 2022.

[13] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing," *ACM computing surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[14] S. Kadavath, T. Conerly, A. Askell, T. Henighan, D. Drain, E. Perez, N. Schiefer, Z. Hatfield-Dodds, N. DasSarma, E. Tran-Johnson *et al.*, "Language models (mostly) know what they know," *CoRR*, 2022.

[15] S. Vatsal and H. Dubey, *A survey of prompt engineering methods in large language models for different NLP tasks*, *https://arXiv.org/abs/2407.12994*.

[16] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff *et al.*, "The prompt report: a systematic survey of prompting techniques," *CoRR*, 2024.

[17] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: a survey on few-shot learning," *ACM computing surveys (csur)*, vol. 53, no. 3, pp. 1–34, 2020.

[18] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, and et al., "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[19] P. Zhou, J. Pujara, X. Ren, X. Chen, H.-T. Cheng, Q. V. Le, E. Chi, D. Zhou, S. Mishra, and H. S. Zheng, "Self-discover: large language models self-compose reasoning structures," *Advances in Neural Information Processing Systems*, vol. 37, pp. 126 032–126 058, 2024.

[20] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: deliberate problem solving with large language models," *Advances in neural information processing systems*, vol. 36, pp. 11 809–11 822, 2023.

[21] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, *React: synergizing reasoning and acting in language models*. ICLR, 2023.

[22] Z. Li, Y. Du, J. Hu, X. Wan, and A. Gao, *Self-instructed derived prompt generation meets in-context learning: unlocking new potential of black-box LLMs*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar,

Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025. [Online]. Available: https://aclanthology.org/2025.acl-long.92/

[23] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, *Rethinking the role of demonstrations: what makes in-context learning work?*, *https://arXiv.org/abs/2202.12837*.

[24] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, *Fantastically ordered prompts and where to find them: overcoming few-shot prompt order sensitivity*. ACL, 2022.

[25] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma, *An explanation of in-context learning as implicit bayesian inference*. ICLR, 2022.

[26] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh, *AutoPrompt: eliciting knowledge from language models with automatically generated prompts*. EMNLP, 2020.

[27] J. Wang, Z. Hu, and L. Bing, *Evolving prompts in-context: an open-ended, self-replicating perspective*. ICML, 2025.

[28] Y. Zhang, Y. Dong, S. Zhang, T. Min, H. Su, and J. Zhu, *Exploring the transferability of visual prompting for multimodal large language models*. IEEE, 2024.

[29] Y. Su, X. Wang, Y. Qin, C.-M. Chan, Y. Lin, H. Wang, K. Wen, Z. Liu, P. Li, J. Li, L. Hou, M. Sun, and J. Zhou, *On transferability of prompt tuning for natural language processing*. Association for Computational Linguistics, 2022. [Online]. Available: http://dx.doi.org/10.18653/v1/2022.naacl-main.290

[30] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, *Large language models are human-level prompt engineers*. NIPS, 2022.

[31] W. Li, X. Wang, W. Li, and B. Jin, "A survey of automatic prompt engineering: an optimization perspective," *arXiv e-prints*, pp. arXiv–2502, 2025.

[32] S. Yang, Y. Wu, Y. Gao, Z. Zhou, B. Zhu, X. Sun, J.-G. Lou, Z. Ding, A. Hu, Y. Fang *et al.*, *AMPO: automatic multi-branched prompt optimization*. Association for Computational Linguistics, 2024.

[33] X. Wang, C. Li, Z. Wang, F. Bai, H. Luo, J. Zhang, N. Jojic, E. Xing, and Z. Hu, *PromptAgent: strategic planning with language models enables expert-level prompt optimization*. ICLR, 2023.

[34] M. Deng, J. Wang, C.-P. Hsieh, Y. Wang, H. Guo, T. Shu, M. Song, E. Xing, and Z. Hu, *RLPrompt: optimizing discrete text prompts with reinforcement learning*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022. [Online]. Available: https://aclanthology.org/2022.emnlp-main.222/

[35] T. Zhang, X. Wang, D. Zhou, D. Schuurmans, and J. E. Gonzalez, *TEMPERA: test-time prompt editing via reinforcement learning*. ICLR, 2023.

[36] C. Singh, J. X. Morris, J. Aneja, A. M. Rush, and J. Gao, *Explaining patterns in data with language models via interpretable autoprompting*, *https://arXiv.org/abs/2210.01848*.

[37] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, *EvoPrompt: connecting LLMs with evolutionary algorithms yields powerful prompt optimizers*, *https://arXiv.org/abs/2309.08532*.

[38] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, *Large language models as optimizers*. ICLR, 2023.

[39] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng, *Automatic prompt optimization with "gradient descent" and beam search*. EMNLP, 2023.

[40] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, and et al., "A survey on evaluation of large language models," *ACM transactions on intelligent systems and technology*, vol. 15, no. 3, pp. 1–45, 2024.

[41] V. N. Zhuravlev, A. R. Khairullin, E. A. Dyagin, A. N. Sitkina, and N. I. Kulin, "Reflectiveprompt: reflective evolution in autoprompting algorithms," in press, *https://arxiv.org/abs/2508.18870*.

[42] ——, "Automatic prompt optimization with prompt distillation," unpublished, *https://arxiv.org/abs/2508.18992*.

[43] O. Baclic, M. Tunis, K. Young, C. Doan, H. Swerdfeger, and J. Schonfeld, "Challenges and opportunities for public health made possible by advances in natural language processing," *Canada Communicable Disease Report*, vol. 46, no. 6, p. 161, 2020.

[44] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, and et al., *Graph of thoughts: solving elaborate problems with large language models*. AAAI, 2024, vol. 38, no. 16.

[45] S. Dhuliawala, M. Komeili, J. Xu, R. Raileanu, X. Li, A. Celikyilmaz, and J. Weston, *Chain-of-Verification reduces hallucination in large language models*, 2023.

[46] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, and et al., "Self-refine: iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 534–46 594, 2023.

[47] S. Hakimov and D. Schlangen, *Images in language space: exploring the suitability of large language models for vision & language tasks*. ACL, 2023.

[48] J. Oppenlaender, "A taxonomy of prompt modifiers for text-to-image generation," *Behaviour & Information Technology*, vol. 43, no. 15, pp. 3763–3776, 2024.

[49] S. Wang, C.-H. Yang, J. Wu, and C. Zhang, *Can whisper perform speech-based in-context learning?* IEEE, 2024.

[50] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, and et al., "Video generation models as world simulators," *OpenAI Blog*, vol. 1, no. 8, p. 1, 2024.

[51] L. Tang, P.-T. Jiang, H. Xiao, and B. Li, "Towards training-free open-world segmentation via image prompt foundation models," *International Journal of Computer Vision*, vol. 133, no. 1, pp. 1–15, 2025.

[52] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers," *CoRR*, 2023.

[53] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel, "Promptbreeder: self-referential self-improvement via prompt evolution," *arXiv e-prints*, pp. arXiv–2309, 2023.

[54] X. Wang, C. Li, Z. Wang, F. Bai, H. Luo, J. Zhang, N. Jojic, E. P. Xing, and Z. Hu, "Promptagent: strategic planning with language models enables expert-level prompt optimization," *CoRR*, 2023.

[55] M. Kwon, G. Kim, J. Kim, H. Lee, and J. Kim, *StablePrompt: automatic prompt tuning using reinforcement learning for large language models*. Association for Computational Linguistics (ACL), 2024.

[56] C. Singh, J. X. Morris, J. Aneja, A. M. Rush, and J. Gao, "Explaining patterns in data with language models via interpretable autoprompting," *arXiv e-prints*, pp. arXiv–2210, 2022.

[57] J. Hong, N. Lee, and J. Thorne, *ORPO: monolithic preference optimization without reference model*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024. [Online]. Available: https://aclanthology.org/2024.emnlp-main.626/

[58] L. Yin and Z. Wang, "Llm-autodiff: auto-differentiate any llm workflow," *arXiv e-prints*, pp. arXiv–2501, 2025.

[59] E. Agarwal, J. Singh, V. Dani, R. Magazine, T. Ganu, and A. Nambi, *PromptWizard: task-aware prompt optimization framework*, *https://arxiv.org/abs/2405.18369*.

[60] Promptfoo, *Promptfoo: LLM evals & red teaming*, *https://github.com/promptfoo/promptfoo*.

[61] J. Shen, J. J. Dudley, J. Zheng, B. Byrne, and P. O. Kristensson, "Promptor: a conversational and autonomous prompt generation agent for intelligent text entry techniques," *CoRR*, 2023.

[62] K. Opsahl-Ong, M. J. Ryan, J. Purtell, D. Broman, C. Potts, M. Zaharia, and O. Khattab, *Optimizing instructions and demonstrations for multi-stage language model programs*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024. [Online]. Available: https://aclanthology.org/2024.emnlp-main.525/

[63] R. Murthy, M. Zhu, L. Yang, J. Qiu, J. Tan, S. Heinecke, C. Xiong, S. Savarese, and H. Wang, *Promptomatix: an automatic prompt optimization framework for large language models*, *https://arxiv.org/abs/2507.14241*.

[64] L. Gao, X. Ma, J. Lin, and J. Callan, "Precise zero-shot dense retrieval without relevance labels," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 1762–1777.

[65] H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, and G. Song, "Reevo: large language models as hyper-heuristics with reflective evolution," *Advances in neural information processing systems*, vol. 37, pp. 43 571–43 608, 2024.

[66] Y. Li, X. Hu, X. Qu, L. Li, and Y. Cheng, *Test-time preference optimization: on-the-fly alignment via iterative textual feedback*. ICML, 2025. [Online]. Available: https://openreview.net/forum?id=ArifAHrEVD

[67] L. Zhao, Y. Wang, Q. Liu, M. Wang, W. Chen, Z. Sheng, and S. Wang, *Evaluating large language models through role-guide and*

self-reflection: a comparative study. ICLR, 2025. [Online]. Available: https://openreview.net/forum?id=E36NHwe7Zc

[68] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, *G-Eval: NLG evaluation using GPT-4 with better human alignment*, *https://arxiv.org/abs/2303.16634*.