MutantPrompt: Prompt Optimization via Mutation Under a Budget on Modest-sized LMs

Arijit Nag
IIT Kharagpur
arijitnag@iitkgp.ac.in

Animesh Mukherjee
IIT Kharagpur
animeshm@cse.iitkgp.ac.in

Niloy Ganguly
IIT Kharagpur
niloy@cse.iitkgp.ac.in

Soumen Chakrabarti
IIT Bombay
soumen@cse.iitb.ac.in

Abstract

Prompts serve as a critical instruction interface to unlock the diverse capabilities of Large Language Models (LLMs), thus directly influencing the quality of their outputs. While prompt engineering has shown great promise, identifying optimal prompts remains a significant challenge, particularly for low-resource languages, which often face higher computational costs due to increased token generation and limited gold standard task data. In response, we propose **MutantPrompt**: a framework that leverages multi-armed bandit algorithms to efficiently identify optimal prompts tailored to low-resource languages. By framing prompt selection as an exploration-exploitation problem under a fixed computational budget, the framework dynamically balances exploring new prompts with exploiting known high-performing ones. We demonstrate the framework's effectiveness across multiple low-resource Indic language tasks, including classification, question-answering and causal reasoning using three small parameter-size LLMs. The results highlight the cost efficiency of the search method in finding optimal prompts and resulting performance improvements. Our codes are publicly available at https://github.com/NLPatCNERG/MutantPrompt.

1 Introduction

Large Language Models (LLMs) (OpenAI et al., 2023; Touvron et al., 2023; Dubey et al., 2024; Chowdhery et al., 2022) have revolutionized natural language processing (NLP), enabling state-of-the-art performance across various tasks, from text generation to complex reasoning. However, LLMs' effectiveness heavily depends on the quality of the prompts used to interact with them. Prompt engineering—the practice of crafting effective input instructions to elicit high-quality outputs—has gained considerable attention as a means to optimize LLM performance. Despite its potential, identifying the most effective prompts remains a challenging problem, particularly in low-resource language (LRL) settings, perhaps widening the performance gap between LRLs and HRLs (high-resource languages) (Hendy et al., 2023; Jiao et al., 2023; Bang et al., 2023; Toraman et al., 2023; Fujii et al., 2023).

Low-resource languages face unique obstacles when interacting with LLMs. These languages often have limited annotated datasets, making it difficult to fine-tune models effectively. In addition, due to differences in tokenization (Muller et al., 2021; Rust et al., 2021; Ahia et al., 2023; Petrov et al., 2023) and linguistic structures, they frequently incur higher computational costs (Ahia et al., 2023; Petrov et al., 2023; Nag et al., 2024) as LLMs generate more tokens to process input and produce output with slow inference (Petrov et al., 2023; Hofmann et al., 2022). A naïve approach of finding optimal prompts using a trial-and-error method can be costly in such settings, as the higher number of token generations results in a significantly increased budget requirement. Furthermore, compared to HRL tasks, LRL tasks have less available gold standard data (Ulmer et al., 2022; Ko et al., 2021; Zhou et al., 2022), necessitating prompt optimization methods that rely less on such data to be effective.

This paper introduces **MutantPrompt**, a novel framework that addresses these challenges by leveraging multi-armed bandit (MAB) algorithms to optimize prompt selection for LRLs. By framing prompt selection as an exploration-exploitation problem, the framework strategically balances the discovery of new prompts with the utilization of previously identified high-performing ones under a fixed computational budget. This approach enables efficient search and selection of effective prompts without exhaustive manual tuning or expensive computational resources.

We evaluate **MutantPrompt** across multiple

Findings of the Association for Computational Linguistics: ACL 2025, pages 22082–22092

NLP tasks in 11 low-resource Indic languages, including classification, question-answering, and causal reasoning using three recent small-size (2b-7b parameters) LLMs, namely Llama-3.2 (Dubey et al., 2024), Mistral-3 (Jiang et al., 2023) and Gemma-2 (Gemma Team et al., 2024). Our experimental results demonstrate the cost efficiency of the search methodology in identifying optimal prompts while improving task performance. The findings underscore the importance of intelligent prompt selection mechanisms in promoting equitable and efficient LLM applications, ensuring that LRL users benefit from the advancement of NLP.

2 Related work

Continuous prompt-based methods (Li and Liang, 2021; Liu et al., 2023; Zhang et al., 2022a), which tune the parameters of specific input tokens, have demonstrated effectiveness but is costly and less interpretable (Lester et al., 2021), while discrete prompt optimization, using task-specific instructions or keywords, is more interpretable and user-friendly (Schick and Schütze, 2021; Liu et al., 2021). Various methods for automatic discrete prompt search and generation have been explored (Zhang et al., 2022b; Shin et al., 2020; Shi et al., 2022; Wallace et al., 2021; Deng et al., 2022; Chen et al., 2023; Yang et al., 2024; Lin et al., 2024). However, many rely on model gradients or token probabilities, which may not suit all models. Others select the best prompts through enumeration, re-sampling, or iterative edits (Zhou et al., 2023; Jiang et al., 2020; Zhang et al., 2022b; Prasad et al., 2023; Sclar et al., 2024). While effective, these methods can be computationally expensive or converge to suboptimal solutions. Recently, EvoPrompt (Guo et al., 2024) demonstrates the effectiveness of genetic and evolutionary algorithms to give optimal prompts without the need for model parameter tuning and is doing better than recent pseudo-gradient (Pryzant et al., 2023) and model token probability-based methods (Zhou et al., 2023). While their method provides valuable insights, it relies on extensive inference budgets due to longer instructions and full development data for scoring prompts, which can be costly. In contrast, **MutantPrompt** uses a budget-constrained MAB approach, exploiting and mutating the best-performing prompt using simple instructions and a dynamic mutation rate while using a small subset of data for scoring. We compare **MutantPrompt** with EvoPrompt to demonstrate performance and cost improvements.

Algorithm 1 MutantPrompt.

Inputs:
- Seed prompt p_{seed}, Instruction p_{instr}, Budget B
- Max mutation rate R_{max}, Min mutation rate R_{min}
- Decay rate R_{decay}, Decay step D_{step}
- Dev dataset $\mathcal{D}_{dev}$, Batch size b

1: $\mathcal{P} \leftarrow \{p_{seed}\}$ /* Prompt pool */
2: $\mathcal{I} \leftarrow \frac{B}{b}$ /* Total iterations */
3: $Count_{succ} \leftarrow \{\}$ /* Success counts, empty dictionary */
4: $Count_{fail} \leftarrow \{\}$ /* Failure counts, empty dictionary */
5: **for** $i \in [0, \mathcal{I})$ **do**
6: **for** $p_i \in \mathcal{P}$ **do**
7: $\Theta[i] \leftarrow \beta_i(Count_{succ}[i]+1, Count_{fail}[i]+1)$ /* Sample prompt's success probability from Beta distribution */
8: $p_{curr} \leftarrow \arg\max_{p \in \mathcal{P}} \Theta[p]$
9: $batch_{succ}, batch_{fail} = evaluate\_prompt(p_{curr}, \mathcal{D}_{dev}, b)$ /* Evaluate the prompt on a batch of dev set and return the success and failure count */
10: $Count_{succ}[p_{curr}] += batch\_succ$
11: $Count_{fail}[p_{curr}] += batch\_fail$
12: $R_{curr} = \max(R_{min}, R_{max} \cdot R_{decay}^{\lfloor \frac{i}{D_{step}} \rfloor})$ /* Update mutation rate */
13: $Prob_{mut} \sim \mathcal{U}(0,1)$ /* Generate random probability of mutation from uniform distribution */
14: **if** $Prob_{mut} \leq R_{curr}$ **then**
15: $p_{best} = \arg\max_{p \in \mathcal{P}} \frac{Count_{succ}[p]}{(Count_{succ}[p]+Count_{fail}[p])}$
16: $p_{new} = refine\_prompt(p_{instr}, p_{best})$ /* Generate new prompt from p_{best} */
17: $\mathcal{P} \leftarrow \mathcal{P} \cup p_{new}$
18: $Count_{succ}[p_{new}] = 0$
19: $Count_{fail}[p_{new}] = 0$
20: $p_{final} \leftarrow \arg\max_{p \in \mathcal{P}} \frac{(Count_{succ}[p])}{(Count_{succ}[p]+Count_{fail}[p])}$
21: **return** p_{final} /* Best performing prompt */

3 Proposed method: MutantPrompt

In this work, given a task and a budget, we frame the search for an optimal prompt as a Multi-Armed Bandit (MAB) problem where each arm represents a prompt (we use Thompson Sampling to implement MAB). In addition, we want to extend the prompt pool by mutating the best-performing prompt after each iteration. We describe **MutantPrompt** in Algorithm 1. At the start of each iteration, we sample the probability of success of each prompt p_i denoted by Θ_i, from a *beta* distribution with the parameter α_i and β_i as the number of successes and a number of failures correspondingly till now for the prompt (line 7). Initially, they are set to zero, making it a uniform distribution. Next, we take the prompt with the highest success probability (line 8) and explore it by evaluating it on a small test set, and update the success and failure count of the prompt (lines 9–11). Subsequently, we probabilistically mutate the prompt

Task↓ | Lang→	Method	as	bn	gu	hi	kn	ml	mr	or	pa	ta	te	avg.
IndicSentiment	p_{seed}**(Llama-3.2)**	57.60	55.60	71.80	54.40	39.50	53.80	38.30	50.30	59.80	34.70	29.10	49.54
	p_{final}**(Llama-3.2)**	85.10	89.60	89.40	92.00	87.20	87.20	90.20	76.20	90.40	92.30	91.90	**88.32**
	p_{seed}**(Gemma-2)**	89.00	95.30	93.50	96.70	92.10	93.70	94.50	65.20	90.30	95.20	93.20	90.79
	p_{final}**(Gemma-2)**	89.80	95.50	94.80	96.90	92.50	93.30	95.20	65.40	91.10	95.50	93.20	**91.20**
	p_{seed}**(Mistral-3)**	64.70	80.80	61.50	86.70	61.60	55.80	65.10	56.50	56.00	63.50	53.90	64.19
	p_{final}**(Mistral-3)**	75.90	86.40	73.00	91.80	77.50	64.60	78.10	67.30	62.90	73.30	65.20	**74.18**
IndicXNLI	p_{seed}**(Llama-3.2)**	33.90	34.60	31.55	46.80	11.35	34.85	42.45	33.55	28.55	27.90	30.60	32.37
	p_{final}**(Llama-3.2)**	36.35	41.50	36.75	46.80	43.10	45.30	45.55	37.65	43.40	47.20	37.40	**41.91**
	p_{seed}**(Gemma-2)**	45.35	52.35	49.45	58.85	49.50	49.50	51.35	38.10	48.55	52.05	50.40	49.59
	p_{final}**(Gemma-2)**	52.40	58.80	58.35	64.85	57.00	56.25	57.75	41.65	56.65	59.35	57.70	**56.43**
	p_{seed}**(Mistral-3)**	35.05	40.50	36.20	45.45	36.15	35.05	36.85	33.45	35.80	38.55	34.75	37.07
	p_{final}**(Mistral-3)**	43.00	49.95	46.10	55.30	45.65	45.25	44.70	44.05	49.30	44.85	39.25	**46.13**
IndicXParaphrase	p_{seed}**(Llama-3.2)**	33.45	31.00	28.45	36.95	8.50	26.10	30.45	49.95	46.40	-	23.20	31.45
	p_{final}**(Llama-3.2)**	61.85	73.60	52.55	49.95	60.20	67.20	66.35	49.30	59.81	-	63.45	**60.43**
	p_{seed}**(Gemma-2)**	60.00	80.50	55.20	83.70	65.45	68.85	72.05	49.75	53.40	-	69.80	65.87
	p_{final}**(Gemma-2)**	69.45	91.95	71.50	94.50	73.60	76.30	84.40	50.15	56.16	-	75.10	**74.31**
	p_{seed}**(Mistral-3)**	63.40	81.90	51.35	92.50	65.90	63.05	72.50	50.60	52.25	-	54.70	64.82
	p_{final}**(Mistral-3)**	62.55	86.10	53.55	93.40	67.80	66.75	74.55	50.70	53.35	-	56.10	**66.49**
IndicCOPA	p_{seed}**(Llama-3.2)**	54.20	65.20	55.36	72.16	57.80	55.60	57.02	49.00	58.40	60.80	61.20	58.79
	p_{final}**(Llama-3.2)**	57.20	67.80	59.60	72.16	62.20	57.40	59.91	49.00	65.00	62.80	59.40	**61.13**
	p_{seed}**(Gemma-2)**	58.20	62.00	58.71	70.60	58.00	57.80	57.46	49.20	58.60	59.60	60.00	59.11
	p_{final}**(Gemma-2)**	60.60	65.00	62.50	69.27	58.00	60.80	60.36	51.20	60.00	62.80	64.00	**61.32**
	p_{seed}**(Mistral-3)**	55.00	64.60	54.02	73.05	57.60	58.60	57.02	49.40	57.60	61.80	58.80	58.86
	p_{final}**(Mistral-3)**	55.00	63.40	57.37	73.27	59.60	61.00	63.03	53.00	61.40	60.80	67.60	**61.41**
CSQA	p_{seed}**(Llama-3.2)**	29.66	33.05	13.90	38.57	32.68	26.65	33.50	19.13	32.48	31.55	31.75	29.36
	p_{final}**(Llama-3.2)**	35.50	36.50	32.30	47.52	41.04	33.20	41.10	31.68	39.49	33.35	33.15	**36.80**
	p_{seed}**(Gemma-2)**	32.15	34.95	19.70	38.72	37.29	27.50	33.00	26.55	21.97	31.40	31.90	30.47
	p_{final}**(Gemma-2)**	34.28	38.30	58.30	42.47	37.39	32.25	37.05	28.39	26.23	35.15	31.30	**36.46**
	p_{seed}**(Mistral-3)**	26.46	24.35	14.90	19.76	21.72	25.05	21.05	25.48	22.22	20.50	23.15	22.24
	p_{final}**(Mistral-3)**	26.16	24.35	36.50	22.06	27.83	26.30	23.00	28.69	26.93	25.80	24.15	**26.52**

Table 1: Performance comparison of seed prompt (p_{seed}) and the optimal prompt (p_{final}) generated by **MutantPrompt** for different tasks, languages and LLMs. On the right, we report the average performance across languages for a particular task and LLM. Best performances are marked in **Bold** and underlined.

with the best performance and add it to the prompt pool (line 14-19) (Appendix A Figure 1 shows the instruction p_{instr} used for this). By mutating the best prompt, we expect another reasonably effective prompt to be added to the pool, making the prompt pool rich. We also exponentially reduce the mutation rate (line 12) as we move forward before it saturates to the minimum mutation rate. There are two motivations for doing this – first, as we start with a single prompt, it is essential to mutate more frequently in the beginning to increase the prompt pool. Second, if we keep mutating at a higher rate throughout iterations, the prompt pool will explode, not converging the beta distributions as exploration outweighs exploitation. After the entire budget is consumed, we select the prompt with the best success ratio as the optimal prompt (line 20).

In summary, starting with a single seed prompt, the algorithm iteratively evaluates its performance and refines the prompt pool. At each iteration, Thompson Sampling is used to estimate the success probabilities of all prompts in the pool, selecting the most promising one for evaluation. The success and failure counts of the selected prompt are updated based on its performance. To expand the pool, the algorithm probabilistically mutates the best-performing prompt into a new one using a dynamic mutation rate that decreases over time, ensuring a balance between exploration (discovering new prompts) and exploitation (exploring existing prompts to build confidence). This iterative process continues until the computational budget is exhausted, after which the prompt with the highest success rate is selected as the optimal solution. Details of algorithm parameters are in Appendix B (Table 5).

4 Experiment and results

To check the effectiveness of **MutantPrompt**, we experiment with five datasets, of which four are from IndicXTREME (Doddapaneni et al., 2023) – IndicSentiment (sentiment classification), IndicXNLI (natural language inference), IndicCOPA (commonsense causal reasoning), IndicXParaphrase (paraphrase detection) and one from IndicGLUE (Kakwani et al., 2020) – CSQA (multiple-choice QA) covering 11 Indian languages. We use three recent small parameter size LLMs (2B-7B parameters) to show the technique's effectiveness in resource-constrained setup. The LLMs are Mistral-3(7B), Llama-3.2(3B) and Gemma-2(2B). For performance metrics, we use exact match accuracy for all the tasks. Details

Task↓ | Lang→	Method	as	bn	gu	hi	kn	ml	mr	or	pa	ta	te	avg.
IndicSentiment	**EvoPrompt**	84.50	83.80	87.84	80.88	84.30	83.60	86.20	45.65	84.10	81.50	89.10	81.04
	MutantPrompt	85.10	89.60	89.40	92.00	87.20	87.20	90.20	76.20	90.40	92.30	91.90	**88.32**
IndicXNLI	**EvoPrompt**	35.10	35.70	33.60	47.60	37.05	39.15	42.60	36.70	35.85	30.25	32.25	36.90
	MutantPrompt	36.35	41.50	36.75	46.80	43.10	45.30	45.55	37.65	43.40	47.20	37.40	**41.91**
IndicXParaphrase	**EvoPrompt**	56.80	65.95	52.40	75.20	46.35	51.25	59.70	57.95	54.00	-	57.00	57.66
	MutantPrompt	61.85	73.60	52.55	49.95	60.20	67.20	66.35	49.30	59.81	-	63.45	**60.43**
IndicCOPA	**EvoPrompt**	52.40	66.40	60.49	72.61	56.00	59.20	59.69	47.60	60.80	62.40	57.60	59.56
	MutantPrompt	57.20	67.80	59.60	72.16	62.20	57.40	59.91	49.00	65.00	62.80	59.40	**61.13**
CSQA	**EvoPrompt**	33.72	35.75	24.20	38.87	33.13	35.55	32.25	30.76	33.63	36.30	33.75	33.45
	MutantPrompt	35.50	36.50	32.30	47.52	41.04	33.20	41.10	31.68	39.49	33.35	33.15	**36.80**

Table 2: Performance comparison of **MutantPrompt** with **EvoPrompt** for the Lamma-3.2(3B) model. On the right, we report the average performance across languages for the tasks. Best performances are marked in **Bold** and underlined.

of model parameters are in Appendix B (Table 6).

Lang	$p^1_{seed} \rightarrow p^1_{final}$	$p^2_{seed} \rightarrow p^2_{final}$	$p^3_{seed} \rightarrow p^3_{final}$
as	35.35→35.09	32.50→36.82	29.56→35.50
bn	36.70→36.45	34.55→37.25	33.05→36.50
gu	21.50→33.60	17.45→25.35	13.90→32.30
hi	44.42→42.42	40.82→42.02	38.57→47.52
kn	37.57→37.12	35.37→38.62	32.68→41.04
ml	32.35→31.00	29.10→29.65	26.65→33.20
mr	37.90→36.10	35.25→36.70	33.50→41.10
or	26.86→27.77	22.11→28.23	19.13→31.68
pa	37.82→39.37	34.82→34.72	32.48→39.49
ta	35.95→36.30	33.60→34.20	31.55→33.35
te	35.00→33.20	33.90→35.15	31.75→33.15
avg.	34.67→35.31	31.77→34.43	29.35→36.80

Table 3: Performance comparison between different seed prompt (p^i_{seed}) to their final optimal prompt (p^i_{final}) for the CSQA dataset using Llama-3.2(3B). The last row shows the average performance across languages for p^i_{seed} and p^i_{final}. Here, the performances of p^i_{seed} degrade as i increases, but p^i_{final} remains in a similar range, showing the robustness of **MutantPrompt**.

4.1 Generated prompt is better than seed

In Table 1, we compare the performance of p_{seed} with that of p_{final} for different tasks and languages. We observe that p_{final} always performs better than p_{seed}, irrespective of the LLMs used. The performance gain is as high as 90% for Llama-3.2 and 25% for Gemma-2 and Mistral-3, which shows the effectiveness of our approach. The huge gain for Llama-3.2 is possibly due to its ability to respond better to optimally designed prompts. Overall, the generation of a more elaborate prompt can be one of the reasons behind the performance improvement (details are in Figure 2 of the Appendix).

4.2 Our method beats recent baseline

In Table 2, we compare the performance of the final optimal prompt generated (for our case it is p_{final}) by **MutantPrompt** and the recent prompt generation technique EvoPrompt starting from same seed prompt (p_{seed}) for all the tasks using the Llama-3.2 model (results with other LLMs are in Appendix Table 8 & 7). Here, we see that most of the time, **MutantPrompt** shows better performance than EvoPrompt. Although there are cases where **MutantPrompt** lags behind or gives similar results compared to EvoPrompt, **MutantPrompt** proves to be more cost-effective (detailed discussion in Section 4.4).

Task	Method	$Token_{instr}$	$\lvert p_{new} \rvert$
IndicSentiment	**EvoPrompt**	991	100
	MutantPrompt	263	129
IndicXNLI	**EvoPrompt**	1011	100
	MutantPrompt	293	130
IndicXParaphrase	**EvoPrompt**	983	100
	MutantPrompt	275	116
IndicCOPA	**EvoPrompt**	1039	100
	MutantPrompt	332	114
CSQA	**EvoPrompt**	1019	100
	MutantPrompt	303	142

Table 4: Here, we compare the token length of the instruction prompts ($Token_{instr}$) between **EvoPrompt** and **MutantPrompt** for different tasks using Llama-3.2(3B) (other LLM results are in Appendix Table 9 & 10). We also compare the average number of new prompts (p_{new}) explored by both techniques.

4.3 Robustness of generated prompt

In Table 3, we check the robustness of the generated prompt p_{final} for **MutantPrompt**. Here, we take the task as CSQA, Llama-3.2 as the LLM, and use three different seed prompts denoted by p^1_{seed}, p^2_{seed} and p^3_{seed}. We see the seed prompts perform differently and, on average, follow the $p^1_{seed} > p^2_{seed} > p^3_{seed}$. But if we check the final prompt performances generated from each of these three seed prompts, they all give performance improvement but, more importantly, show similar performance. It shows **MutantPrompt** is not very dependent on the starting seed prompts, which makes it very practical in real-world use cases where the user does not need to worry much about designing the seed prompt.

4.4 Cost-effective and explores more

In Table 4, we analyze the token count (a proxy for incurred cost) of the instruction prompt that di-

rects the LLM to generate a new prompt for **MutantPrompt** alongside EvoPrompt. We find that the EvoPrompt instruction prompt results in 3X-4X more tokens than **MutantPrompt**, making it more cost-effective. Also, we compare the total number of new prompts (p_{new}) explored between the two techniques, given the same budget (here, we use 4000 as the total budget). It shows **MutantPrompt** explores 15%-40% more prompts throughout its journey, making it more probable to find the optimal prompt.

5 Conclusion

This paper presents a framework leveraging multi-armed bandit algorithms to generate and search for optimal prompts for low-resource language tasks. By framing the prompt selection process as an exploration-exploitation problem under a fixed computational budget, the proposed approach efficiently balances the discovery of new prompts with the exploitation of known high-performing ones. This method addresses the unique challenges faced by low-resource language tasks, including limited gold standard data, high computational costs due to increased token generation, and the need to optimize prompts without overwhelming smaller LLMs with overly complex instructions. Our experimental evaluations on various NLP tasks in low-resource Indic languages demonstrate the effectiveness and cost-efficiency of this approach. Moving forward, we want to explore the proposed framework for more complex tasks and other LRLs, fostering more inclusive and efficient LLM querying.

6 Limitations

While our proposed framework using the multi-armed bandit for prompt optimization demonstrates effectiveness in low-resource language settings, there are several limitations that we want to work on in future.

Generalization across languages: While the framework shows promise for low-resource Indic languages, its generalization to other low-resource languages with vastly different linguistic structures remains untested. Further evaluations are needed to understand its effectiveness across diverse language families.

Dependency on LLM quality: The effectiveness of the optimized prompts is inherently tied to the capabilities and limitations of the underlying LLM. In low-resource settings, smaller or less powerful models may still struggle with nuanced tasks, even with optimized prompts.

Scalability to complex tasks: The framework has been evaluated on classification, question-answering, and causal reasoning tasks. However, its scalability and effectiveness on more complex NLP tasks, such as generative tasks or tasks requiring multi-step reasoning, remain to be explored.

Exploration-exploitation trade-off: Although the framework dynamically balances exploration and exploitation using simple heuristics, finding the optimal balance is challenging, especially in low-resource settings where computational budgets are limited. An inappropriate balance may lead to either premature convergence to suboptimal prompts or excessive exploration, wasting computational resources. We need to devise a concrete approach that can decide the balance based on task quality and the available development dataset.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David R. Mortensen, Noah A. Smith, and Yulia Tsvetkov. 2023. Do all languages cost the same? tokenization in the era of commercial language models.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2023. Instructzero: Efficient instruction optimization for black-box large language models.

Aakanksha Chowdhery et al. 2022. Palm: Scaling language modeling with pathways.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning.

Sumanth Doddapaneni, Rahul Aralikatte, Gowtham Ramesh, Shreya Goyal, Mitesh M. Khapra, Anoop Kunchukuttan, and Pratyush Kumar. 2023. Towards leaving no indic language behind: Building monolingual corpora, benchmark and models for indic languages.

Abhimanyu Dubey et al. 2024. The llama 3 herd of models.

Takuro Fujii, Koki Shibata, Atsuki Yamaguchi, Terufumi Morishita, and Yasuhiro Sogawa. 2023. How do different tokenizers perform on downstream tasks in scriptio continua languages?: A case study in Japanese. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 39–49, Toronto, Canada. Association for Computational Linguistics.

Gemma Team et al. 2024. Gemma 2: Improving open language models at a practical size.

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.

Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. 2023. How good are gpt models at machine translation? a comprehensive evaluation.

Valentin Hofmann, Hinrich Schütze, and Janet Pierrehumbert. 2022. An embarrassingly simple method to mitigate undesirable properties of pretrained language model tokenizers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know?

Wenxiang Jiao, Wenxuan Wang, Jen tse Huang, Xing Wang, Shuming Shi, and Zhaopeng Tu. 2023. Is chatgpt a good translator? yes with gpt-4 as the engine.

Divyanshu Kakwani, Anoop Kunchukuttan, Satish Golla, Gokul N.C., Avik Bhattacharyya, Mitesh M. Khapra, and Pratyush Kumar. 2020. IndicNLPSuite: Monolingual Corpora, Evaluation Benchmarks and Pre-trained Multilingual Language Models for Indian Languages. In *Findings of EMNLP*.

Wei-Jen Ko, Ahmed El-Kishky, Adithya Renduchintala, Vishrav Chaudhary, Naman Goyal, Francisco Guzmán, Pascale Fung, Philipp Koehn, and Mona Diab. 2021. Adapting high-resource NMT models to translate low-resource related languages without parallel data. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 802–812, Online. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. 2024. Use your instinct: Instruction optimization for llms using neural bandits coupled with transformers.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. Gpt understands, too.

Benjamin Muller, Antonios Anastasopoulos, Benoît Sagot, and Djamé Seddah. 2021. When being unseen from mBERT is just the beginning: Handling new languages with multilingual language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 448–462, Online. Association for Computational Linguistics.

Arijit Nag, Animesh Mukherjee, Niloy Ganguly, and Soumen Chakrabarti. 2024. Cost-performance optimization for processing low-resource language tasks using commercial llms.

OpenAI et al. 2023. Gpt-4 technical report.

Aleksandar Petrov, Emanuele La Malfa, Philip H. S. Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the*

59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 3118–3135, Online. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze questions for few shot text classification and natural language inference.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting.

Weijia Shi, Xiaochuang Han, Hila Gonen, Ari Holtzman, Yulia Tsvetkov, and Luke Zettlemoyer. 2022. Toward human readable prompt tuning: Kubrick's the shining is a good movie, and a good prompt too?

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts.

Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. 2023. Impact of tokenization on language models: An analysis for turkish. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 22(4).

Hugo Touvron et al. 2023. LLaMA 2: Open foundation and fine-tuned chat models.

Dennis Ulmer, Jes Frellsen, and Christian Hardmeier. 2022. Exploring predictive uncertainty and calibration in NLP: A study on the impact of method & data scarcity. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2707–2735, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2021. Universal adversarial triggers for attacking and analyzing nlp.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers.

Ningyu Zhang, Luoqiu Li, Xiang Chen, Shumin Deng, Zhen Bi, Chuanqi Tan, Fei Huang, and Huajun Chen. 2022a. Differentiable prompt makes pre-trained language models better few-shot learners.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. 2022b. Tempera: Test-time prompting via reinforcement learning.

Ran Zhou, Xin Li, Ruidan He, Lidong Bing, Erik Cambria, Luo Si, and Chunyan Miao. 2022. MELM: Data augmentation with masked entity language modeling for low-resource NER. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2251–2262, Dublin, Ireland. Association for Computational Linguistics.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers.

MutantPrompt: Prompt Optimization via Mutation Under a Budget on Modest-sized LMs
(Appendix)

A Supplementary results

Below is a description of a task and a sample prompt to query a LLM system to perform the tasks. You need to generate a new prompt to solve the task. Please use the given sample prompt as a reference for designing the new propmt, keeping the underlying theme intact try to improve it by adding variation in phrasing, structure, or style. Like the given sample prompt that assumes inputs are already available in the prompt, the new prompt also should not ask for user inputs(avoid word/phrase like as e.g., 'provide a', 'provide the') or keep placeholders for user inputs and should be phrased in the second person (e.g., 'you', 'your') when asking for correct alternative. The prompt should explicitly mention that only the correct option should be the output. Write the prompt in such a way that a user will use this generated prompt to query a LLM system to solve the task. Output the new prompt directly without starting with phrases like 'Here's is a new prompt...'.

Task description: "A paragraph contains a word denoted by '<MASK>'. The task is to guess the correct alternatives to replace the '<MASK>' word out of the four words given as options."

Sample prompt: "In the given text, you'll find a blank represented by '<MASK>'. From the provided choices, select the word that, when inserted, would render the sentence grammatically correct and coherent within the given context. Your response should solely contain the appropriate word.

Example:
Text: I went to the <MASK> and bought a new book.
Options: store, library, market, park
Correct Answer: store
Paragraph: {paragraph}
Options: {option_a}, {option_b}, {option_c}, {option_d}"

New prompt:

Figure 1: Template for generating new prompt (p_{new}) in the $Refine\_prompt$ function described in Algorithm 1.

B Experimental settings

We run all the experiments on a single RTX A6000 GPU with 48GB RAM. To save GPU hour and cost, we run all the inference and prompt generation experiments one time (except for Table 3, where we do it with 3 different seeds), and to make them reproducible, we fix the LLM seed value to 42. To run **MutantPrompt**, it took around 2 hr for a task and language with a budget of 4000 on a single 48GB RTXA6000 GPU. For zero-shot inference, for each task, we select 2000 random instances for each language (For IndicSentiment and IndicXParaphrase, we use 500 data as data is limited). For EvoPrompt, we use a development dataset size of 50 for prompt optimization.

Parameter	Value
B	[400-4000]
R_{max}	1.0
R_{min}	0.1
R_{decay}	0.9
D_{step}	[1,5]
b	[3,5]

Table 5: Algorithm 1 parameter details.

Parameter	Value
LLM	LLama-3.2(3B), Gemma-2(2B), Mistral-3(7B)
LLM parameter size	2-7 Billion
LLM model type	Instruction tuned
LLM temperature	0.5 (for prompt generation), 0.01(for evaluation)
LLM top p	0.95
LLM max token length	512(for prompt generation), 64(for prediction)
Seed	42

Table 6: LLM parameter details.

Task↓ | Lang→	Method	as	bn	gu	hi	kn	ml	mr	or	pa	ta	te	avg.
IndicSentiment	**EvoPrompt**	88.90	95.70	93.80	93.60	93.30	93.80	93.90	72.10	90.20	95.20	92.60	91.19
	MutantPrompt	89.80	95.50	94.80	96.90	92.50	93.30	95.20	65.40	91.10	95.50	93.20	**91.20**
IndicXNLI	**EvoPrompt**	51.05	58.10	56.45	58.75	47.20	45.10	54.05	40.40	50.80	52.05	54.10	51.64
	MutantPrompt	52.40	58.80	58.35	64.85	57.00	56.25	57.75	41.65	56.65	59.35	57.70	**56.43**
IndicXParaphrase	**EvoPrompt**	68.40	80.50	72.95	92.75	69.95	81.45	78.95	53.20	53.10	-	73.85	72.51
	MutantPrompt	69.45	91.95	71.50	94.50	73.60	76.30	84.40	50.15	56.16	-	75.10	**74.31**
IndicCOPA	**EvoPrompt**	58.00	61.80	53.35	65.92	52.00	59.80	60.58	47.60	55.40	58.60	59.40	57.50
	MutantPrompt	60.6	65.00	62.50	69.27	58.00	60.80	60.36	51.20	60.00	62.80	64.00	**61.32**
CSQA	**EvoPrompt**	31.89	32.00	31.70	38.12	35.84	29.00	31.15	27.47	25.58	33.35	31.95	31.64
	MutantPrompt	34.28	38.30	58.30	42.47	37.39	32.25	37.05	28.39	26.23	35.15	31.30	**36.46**

Table 7: Performance comparison between **MutantPrompt** technique with **EvoPrompt** for the generated optimal prompt (p_{final}) on Gemma-2(2B) model. On the right, we report the average performance across languages for the tasks. Best performances are marked in **Bold** and underlined.

Task↓ | Lang→	Method	as	bn	gu	hi	kn	ml	mr	or	pa	ta	te	avg.
IndicSentiment	**EvoPrompt**	74.90	86.40	72.20	91.90	77.10	63.10	77.70	68.90	62.50	72.90	63.60	73.75
	MutantPrompt	75.90	86.40	73.00	91.80	77.50	64.60	78.10	67.30	62.90	73.30	65.20	**74.18**
IndicXNLI	**EvoPrompt**	42.10	50.15	37.15	52.55	42.80	41.85	48.20	43.90	42.75	47.00	40.30	44.43
	MutantPrompt	43.00	49.95	46.10	55.30	45.65	45.25	44.70	44.05	49.30	44.85	39.25	**46.13**
IndicXParaphrase	**EvoPrompt**	68.05	88.50	52.50	92.10	64.50	66.20	72.45	52.05	57.91	-	55.50	**66.98**
	MutantPrompt	62.55	86.10	53.55	93.40	67.80	66.75	74.55	50.70	53.35	-	56.10	66.49
IndicCOPA	**EvoPrompt**	45.40	50.20	49.11	55.68	49.60	48.80	53.45	49.60	40.40	51.40	49.60	49.39
	MutantPrompt	55.00	63.40	57.37	73.27	59.60	61.00	63.03	53.00	61.40	60.80	67.60	**61.41**
CSQA	**EvoPrompt**	29.91	26.75	23.55	32.67	25.63	31.70	26.35	30.07	20.32	29.25	23.35	**27.23**
	MutantPrompt	26.16	24.35	36.50	22.06	27.83	26.30	23.00	28.69	26.93	25.80	24.15	26.52

Table 8: Performance comparison between **MutantPrompt** technique with **EvoPrompt** for the generated optimal prompt (p_{final}) on Mistral-3(7B) model. On the right, we report the average performance across languages for the tasks. Best performances are marked in **Bold** and underlined.

Task	Method	$Token_{instr}$	$\lvert p_{new} \rvert$
IndicSentiment	**EvoPrompt**	1185	100
	MutantPrompt	275	129
IndicXNLI	**EvoPrompt**	1138	100
	MutantPrompt	296	130
IndicXParaphrase	**EvoPrompt**	1091	100
	MutantPrompt	278	116
IndicCOPA	**EvoPrompt**	1234	100
	MutantPrompt	345	114
CSQA	**EvoPrompt**	1406	100
	MutantPrompt	316	142

Table 9: Here, we compare the token length of the instruction prompts ($Token_{instr}$) between **EvoPrompt** and **MutantPrompt** for different tasks using Gemma-2(2B). We also compare the average number of new prompts (p_{new}) explored by both techniques.

Task: CSQA

Seed prompt: "The paragraph below includes a gap indicated by <MASK>. From the given options, choose the word that accurately fills the gap. Answer with only the correct word:
Paragraph: {paragraph}
Options: {option_a}, {option_b}, {option_c}, {option_d}"

Generated prompt: "In the given text, there is a word signified by '<MASK>', and the objective is to select the appropriate word from the provided options that, when inserted, will result in a sentence that is both grammatically correct and logically sensible. Please identify the word that fits seamlessly into the sentence without any additional information:

Example:

Text: The child <MASK> the toy.
Options: broke, played, laughed, held
Correct answer: played

In this case, you are required to find the word that, when inserted into the text, creates a grammatically sound and logically consistent sentence. Provide the suitable word without any extra details:

Example:

Text: The bird <MASK> the nest.
Options: built, sang, flew, hatched
Correct answer: built
Paragraph: {paragraph}
Options: {option_a}, {option_b}, {option_c}, {option_d}"

Task: IndicXNLI

Seed prompt: "Identify the relation('neutral'/'contradiction'/'entailment') between Premise and Hypothesis. Output only the correct relation.
Premise: {premise}
Hypothesis: {hypothesis}"

Generated prompt: "Determine the nature ('Neutral', 'Contradiction', or 'Entailment') of the relationship that exists between the presented statement (Premise) and the following statement (Hypothesis). Your response should clearly illustrate the connection between the two statements.

For example, if the Premise is "The cat is on the mat" and the Hypothesis is "The cat is not on the mat", the relationship is 'Contradiction'.

Please provide the exact relationship for the given Premise and Hypothesis.
Premise: {premise}
Hypothesis: {hypothesis}"

Task: IndicXParaphrase

Seed prompt: "Identify if the Sentence2 is a paraphrase of Sentence1. Output only either 'yes' or 'no'.
Sentence1: {sentence1}
Sentence2: {sentence2}"

Generated prompt: "Analyze each pair of sentences and determine if Sentence2 is a paraphrase of Sentence1. For each pair, you should provide a clear "yes" or "no" response indicating whether Sentence2 is a paraphrase of Sentence1. Your response should only include the correct answer.
Sentence1: {sentence1}
Sentence2: {sentence2}"

Task: IndicCOPA

Seed prompt: "Given the Premise, Question and two choices('Choice1'/'Choice2'), return the correct choice that answer the question for the premise. Output only 'Choice1' or 'Choice2', no need to output the whole sentence.
Premise: {premise}
Choice1: {choice1}
Choice2: {choice2}
Question: {question}"

Generated prompt: "Given a scenario described by a premise and a question about either 'cause' or 'effect', and presented with two possible answers ('Choice1'/'Choice2'), kindly determine the response that accurately answers the question within the context of the premise. Simply provide 'Choice1' or 'Choice2' as your answer, without including the question or the premise in your response.
Premise: {premise}
Choice1: {choice1}
Choice2: {choice2}
Question: {question}"

Task: IndicSentiment

Seed prompt: "Given the Sentence, predict the correct sentiment('positive'/'negative') of the sentence. Output only the correct sentiment class.
Sentence: {sentence}"

Generated prompt: "Determine the sentiment of the given statement, categorizing it as 'positive' or 'negative'. Kindly provide the sentiment category as your response.

Example:
Statement: I absolutely love this movie!
Sentiment: positive
Sentence: {sentence}"

Figure 2: Sample seed prompt (p_{seed}) and generated optimal prompt (p_{final}) for all the tasks.

Task	Method	$Token_{instr}$	$\lvert p_{new} \rvert$
IndicSentiment	**EvoPrompt**	1187	100
	MutantPrompt	288	129
IndicXNLI	**EvoPrompt**	1277	100
	MutantPrompt	303	130
IndicXParaphrase	**EvoPrompt**	1217	100
	MutantPrompt	299	116
IndicCOPA	**EvoPrompt**	1334	100
	MutantPrompt	370	114
CSQA	**EvoPrompt**	1298	100
	MutantPrompt	322	142

Table 10: Here, we compare the token length of the instruction prompts ($Token_{instr}$) between **EvoPrompt** and **MutantPrompt** for different tasks using Mistral-3(7B). We also compare the average number of new prompts (p_{new}) explored by both techniques.