# Revisiting the Discrete System-Prompt Optimization in LLM

**Ing. Václav Diviš Ph.D., Ing. Marek Hrúz Ph.D., Ing. Bastian Spatz**

[1+2]UWB Pilsen, Faculty of Applied Sciences (FAV), Department of Cybernetics
Technická 8, 301 00 Plzeň, Czech Republic
[1]vincie@kky.zcu.cz , [2] mhruz@kky.zcu.cz
[3]ARRK Engineering GmbH
bastian.spatz@arrk-engineering.com

**Abstract**

This document presents a general overview and detailed analysis of an approach to optimize the system prompt in a large language model (LLM) via backpropagation. We point out the current challenge of prompt engineering to elicit the desired responses from various LLMs. We discuss how a given system prompt and query (user prompt) are processed through the model, how the loss is computed against a ground truth output, and how gradients are propagated back to update only the system prompt embeddings. We analyze the tensor shapes throughout the forward and backward passes for a specific Llama-3-8B model, including cases where the output length differs from the input length. We further propose a method to align the embeddings and map them back to the discrete (token) space, making the optimized system's prompts interpretable.

**Keywords:** System Prompt Optimization, Large Language Models, Gradient-based Embedding Update

## 1. Introduction

In LLM applications, the system prompt is crucial for steering the model's behavior. It is used during supervised fine-tuning (SFT) and reinforcement learning (RL). Unlike the user prompt, the system prompt is typically hidden from end users. Selecting an effective system prompt involves defining its role, format, and constraints, as well as iteratively testing and aligning it with the intended application goals. To improve this process, we propose an optimization method that refines the system prompt by backpropagating a token-level loss into its embeddings, using a question–answer training set that represents the target operational domain.

Our work follows the taxonomy of the Automatic Prompt Optimization (APO) formulation (Ramnath et al. 2025). Given an initial system prompt $P = (p_1, p_2, \cdots, p_N) \in \mathcal{V}^N$, where $\mathcal{V}$ is the vocabulary, and pairs of input–output text $\{(x_i, y_i)\}_{i=1}^{K}$ from the training set $\mathcal{D}_{\text{train}}$ for a model $M$, the goal is to search for an optimal system prompt $P_{\text{opt}}$ through the following optimization loop:

1. Process the input $x_i$ through the embedding layer, transformer blocks, and the LM head to generate logits.
2. Compute the per-token loss $\mathcal{L}$ (e.g., cross-entropy) with respect to the ground-truth output $y_i$.
3. Backpropagate the loss, updating only the system prompt embeddings while keeping the transformer parameters and query embeddings frozen.
4. Map the updated continuous prompt embeddings back to discrete tokens via nearest-neighbor search in the token embedding space.
5. Repeat until the loss $\mathcal{L}$ converges (i.e., reaches a plateau).

The optimization objective can be expressed as:

$$P_{\text{opt}} := \arg\min_{P \in \mathcal{V}^N} \mathbb{E}_{(x_i, y_i) \sim \mathcal{D}_{\text{train}}} [\mathcal{L}(M(P \oplus x_i), y_i)],$$

where $\oplus$ means concatenation.

## 2. Related Work

Prompt optimization has emerged as a key strategy to adapt the LLM performance without the need for full fine-tuning the corresponding model. Instead of designing and crafting effective prompts to guide the LLM toward useful outputs, which is a domain of Prompt engineering (Sahoo et al. 2024), the prompt optimization focuses on systematically refining and tuning prompts—using experimentation, feedback, or automated methods—to maximize performance, accuracy, or efficiency. The prompt optimization methods fall broadly into *continuous* and *discrete* approaches.

### 2.1. Continuous (soft) prompts

such as *Prefix-Tuning* (Li and Liang 2021) learn task-specific embedding vectors prepended to model layers while keeping the backbone frozen. This approach is parameter-efficient, performs well in low-data regimes, and can generalize across domains, though the learned prompts are not interpretable.

### 2.2. Discrete prompt optimization

focuses on natural token sequences. *RLPrompt* (Deng et al. 2022) formulates prompt search as a reinforcement learning problem, showing that even ungrammatical prompts can be effective and transferable.

Václav Diviš, Marek Hrúz, Bastian Spatz, *Revisiting the Discrete System-Prompt Optimization in LLM*. In: Z. Vetulani, P. Paroubek, I. Kardava (Eds.), Human Language Technologies as a Challenge for Computer Science and Linguistics – 2025, pp. 23–26. Poznań: Adam Mickiewicz University Press, 2025. © Authors, 2025. https://doi.org/10.14746/amup.9788323245414.4. Open Access chapter, distributed under the terms of the CC licence (BY-NC-ND, https://creativecommons.org/licenses/by-nc-nd/4.0/).

*ProTeGi* (Pryzant et al. 2023) instead uses "natural language gradients" with beam search to iteratively refine prompts, yielding significant improvements across benchmarks.

Finally, recent surveys (Ramnath et al. 2025) provide systematic taxonomies of prompt optimization techniques, highlighting trade-offs between interpretability, performance, and efficiency, as well as open challenges such as initialization sensitivity, cost, and transferability.

## 3. Proposed methodology

Our method falls into the group of discrete prompt optimization. In our theoretical approach, we consider a model architecture similar to Llama-3-8B (Touvron et al. 2023).

- The **Embedding Layer:**

  $$\text{embed_tokens} \in \mathbb{R}^{128\,256\times 2048},$$

  maps the token indices, whose maximum index can be equal to the size of the vocabulary $\mathcal{V}$, to 2048-dimensional vectors.

- The embeddings directly are inputted into the **Transformer Decoder Layers** which, in the case of Llama-3-8B parameter model, has 16 layers each with Self-Attention (SA) and Multi Layer Perceptron (MLP) blocks. For example, in each layer:

  - the SA blocks use query, key, value, and output matrices with the following structure:
    $\text{q_proj}, W_q: \mathbb{R}^{2048\to 2048}$,
    $\text{k_proj}, W_k: \mathbb{R}^{2048\to 512}$,
    $\text{v_proj}, W_v: \mathbb{R}^{2048\to 512}$,
    $\text{o_proj}, W_o: \mathbb{R}^{2048\to 2048}$.
  - The MLP block uses projections:
    $\text{gate_proj}, W_g: \mathbb{R}^{2048\to 8192}$,
    $\text{up_proj}, W_u: \mathbb{R}^{2048\to 8192}$,
    $\text{down_proj}, W_d: \mathbb{R}^{8192\to 2048}$.

- The **final LM Head** linearly projects the intermediate state (of shape 2048) back to the vocabulary logits:

  $$W \in \mathbb{R}^{2048\times 128\,256}.$$

## 4. Forward Pass

The forward pass consists of embedding, processing through the decoder only blocks consisting of SA and MLP, and linear mapping to the logits. Let:

- $P = (p_1, \dots, p_N)$ be the system prompt tokens and
- $Q = (q_1, \dots, q_M)$ be the query tokens.

### 4.1. Input Embeddings

The tokens are embedded as:

$$E(P) \in \mathbb{R}^{N\times 2048}, \quad E(Q) \in \mathbb{R}^{M\times 2048}.$$

The full input is concatenated:

$$X = \begin{bmatrix} E(P) \\ E(Q) \end{bmatrix} \in \mathbb{R}^{(N+M)\times 2048}.$$

### 4.2. Processing through the Transformer

The concatenated embedding $X$ is passed through 16 transformer layers, where the output of each layer maintains the shape:

$$H_l \in \mathbb{R}^{(N+M)\times 2048}, \quad l = 1, \dots, 16.$$

After the final layer, we denote the resulting representation $H_{16}$ as:

$$H_{16} \in \mathbb{R}^{(N+M)\times 2048}.$$

### 4.3. Linear Modeling (LM) Head and Logits

The LM head projects the intermediate states to logits:

$$L = H_{16}\, W^{\top} \in \mathbb{R}^{(N+M)\times 128\,256}.$$

For the sake of clarity, we don't consider biases in our notation, which are part of the MLP and LM layers.

## 5. Loss Computation and Output Length Considerations

Let the ground truth output tokens from $D_{\text{train}}$ be:

$$Y = (y_1, \dots, y_T) \in \mathcal{V}^T.$$

Note that the number of output tokens $T$ may differ from the input length $S = N + M$. In autoregressive settings, outputs are often aligned with shifted inputs or generated using teacher forcing. The per-token loss is computed as:

$$\ell(\hat{y}_i, y_i), \quad i = 1, \dots, T,$$

and aggregated into a scalar loss:

$$\mathcal{L} = \sum_{i=1}^{T} \ell\,(\hat{y}_i, y_i) \quad \text{or} \quad \mathcal{L} = \frac{1}{T}\sum_{i=1}^{T} \ell\,(\hat{y}_i, y_i).$$

## 6. Backpropagation and Gradient Flow

### 6.1. Gradient Propagation

Even though the loss is computed over $T$ output tokens, backpropagation propagates gradients through the LM head and transformer layers, resulting in a gradient on the intermediate states:

$$\nabla_L \mathcal{L} \in \mathbb{R}^{(N+M)\times 2048}.$$

Since the transformer layers are frozen, these gradients can flow directly to the input embedding layer, where only the part related to the system prompt $P$ will be further used:

$$\nabla_P \mathcal{L} \in \mathbb{R}^{N \times 2048}.$$

### 6.2. Focusing on the System Prompt

The input $X$ is composed of the system prompt and query embeddings:

$$\nabla_X \mathcal{L} = \begin{bmatrix} \nabla_{E(P)} \mathcal{L} \\ \nabla_{E(Q)} \mathcal{L} \end{bmatrix}.$$

As aforementioned, we update only the system prompt embeddings:

$$E(P)' = E(P) - \eta \, \nabla_{E(P)} \mathcal{L},$$

$$\text{with } \nabla_{E(P)} \mathcal{L} \in \mathbb{R}^{N \times 2048},$$

where $\eta$ is the learning rate.

## 7. Mapping Back to Discrete Tokens

After updating, the optimized continuous system prompt embeddings $E(P)'$ have to be mapped back to discrete tokens. We propose a simple nearest neighbor search in the vocabulary embedding space $\mathcal{V} \subset \mathbb{R}^{2048}$. Specifically, for each updated embedding $e'_i$ (with $i = 1, \ldots, N$), we compute the index of its closest vocabulary embedding as follows:

$$\hat{k}_i = \arg\min_{k \in \{1,2,\ldots,|\mathcal{V}|\}} \| e'_i - v_k \|_2,$$

where $v_k \in \mathcal{V}$ denotes the $k^{\text{th}}$ token embedding. The returned index $\hat{k}_i$ identifies the nearest token in the discrete vocabulary.

This yields the new system prompt as a sequence of token indices:

$$\hat{P} = (\hat{k}_1, \ldots, \hat{k}_N).$$

Accordingly, if we retrieve the embeddings corresponding to these indices, we obtain a matrix $E(\hat{P}) \in \mathbb{R}^{N \times 2048}$ that resides in the original embedded space.

While simple, this alignment poses a challenge. If the update step $\| \eta \nabla_{E(P)} \mathcal{L} \|$ is smaller than the distance $\| e'_i - v_k \|_2$ to the nearest alternative token embedding, the projection will map it back to the original token. In such cases, the update has no effect, and the system prompt may remain stuck at its previous embedding as depicted in Figure 1.

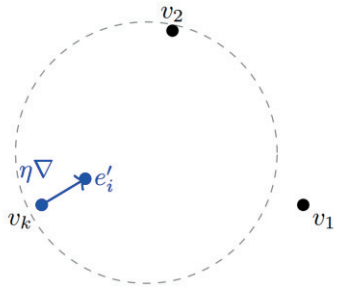


*Figure 1. Illustration of a small gradient step that keeps $e'_i$ inside the Voronoi region of the original embedding $v_k$. The nearest neighbor projection thus maps back to $v_k$, and no effective update occurs.*

There are several approaches that may rectify this situation. One is to omit the mapping to tokens and leave it for the final evaluation, or only map the embeddings after each epoch. Another approach would be to always switch to the nearest token without the possibility of mapping back to the original one. Or one could get k-nearest neighbors in the direction of the gradient and select one randomly. These options are the subject of future work.

## 8. Application

System prompt fine-tuning addresses a critical challenge in adapting language models for specific tasks. Traditionally, there is no automated method available that can establish a perfect system prompt tailored to the nuances of any given task. Our method enables adjustment of the system prompt to optimize performance, potentially unlocking superior results even with smaller models for any task or domain. Moreover, this approach holds promise for specialized applications, including the development of multimodal LLMs.

## 9. Conclusion

In this paper, we presented a unified view of the system prompt optimization process via backpropagation. Starting from the input construction through a Llama 3-like model, we detailed:

1. The forward pass, including embedding lookup, transformer processing, and LM head projection.
2. The computation of a per-token loss, potentially over a sequence length different from the input.
3. The propagation of gradients back to the input embedding layer and the extraction of gradients corresponding to the system prompt.
4. The update of system prompt embeddings and the subsequent mapping back to discrete tokens via a nearest neighbour search.

Our analysis confirms that, with careful handling of tensor shapes, loss aggregation, and mapping updated embeddings into discrete tokens, the proposed optimization of the system prompt is feasible in practice. One aspect that needs to be studied is the possibility of adding and removing the tokens. The presented method is able to substitute only the tokens. However, there are special tokens in the vocabulary that could be utilized for this. For example, the $unk$ token could represent deletion. Also the prompt could be trimmed if one of the embeddings projects to the $end$ token. A deeper look into this problem is needed. We theorize that a reinforcement learning approach may yield interesting results, where the actions of substitution, deletion, and insertion could be learned based on the behavior of the training process.

## Acknowledgements

The work has been supported by ARRK Engineering GmbH.

## References

Deng, Yangyi et al. 2022. “RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning.” *ACL*.

Li, Xiang Lisa, and Percy Liang. 2021. “Prefix-Tuning: Optimizing Continuous Prompts for Generation.” *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)*, 4582–97.

Pryzant, Reid, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. “Automatic Prompt Optimization with ‘Gradient Descent’ and Beam Search.” *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 7957–68.

Ramnath, Kiran, Kang Zhou, Sheng Guan, et al. 2025. “A Systematic Survey of Automatic Prompt Optimization Techniques.” *arXiv Preprint arXiv:2502.16923*.

Sahoo, Pranab, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. “A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications.” *CoRR*.

Touvron, Hugo, Thibaut Lavril, Gautier Izacard, et al. 2023. “Llama: Open and Efficient Foundation Language Models.” *arXiv Preprint arXiv:2302.13971*.