SALIENCE AWARE MARK-STEERED PROMPTING FOR LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The efficacy of Large Language Models (LLMs) is heavily dependent on the quality of user-provided prompts. Consequently, many optimization methods focus on augmenting prompts with extensive details to provide comprehensive context. However, these methods often produce verbose and information-saturated prompts, which inadvertently causes LLMs to lose focus on the most critical instructions. This phenomenon, known as attention dilution, significantly constrains model performance on tasks requiring comprehension of long contexts. To address this issue, we propose *Salience Aware **M**ark-**S**teered **P**rompting* (**MSP**), a novel framework designed to mitigate attention dilution by explicitly steering the model's focus toward the most critical information within the prompt. MSP consists of two stages: first, Gradient-Guided Mask Search (GGMS) automatically identifies the most influential tokens. Second, Mark-Steered Decoding (MSD) persistently guides the model by amplifying the influence of these key tokens at every step of the generation process, improving the model's alignment with core user instructions. We evaluate the effectiveness of MSP on five widely used benchmarks with three representative LLMs of multiple scales. The experimental results show that MSP yields consistent performance gains over state-of-the-art baselines, and its strong performance across diverse tasks and models highlights its robustness and generalizability. Our implementation is provided in the supplementary material.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong performance across a variety of tasks (Colombo et al., 2024; Zhang et al., 2024b; Qin et al., 2023a; Touvron et al., 2023), and prompting has become the primary interface through which users interact with these models (Liu et al., 2023b). In recent years, a large number of prompt optimization methods have been proposed to further unlock the potential of LLMs (Pryzant et al., 2023; Lin et al., 2024; Chen et al., 2024a; Wang et al., 2024; Ye et al., 2024; Hu et al., 2024). Most of these approaches rely on evolutionary algorithms that iteratively modify existing prompts (Fernando et al., 2024; Guo et al., 2024; Agrawal et al., 2025), or on meta-prompts that incorporate scoring mechanisms over a small validation set to refine prompts (Zhou et al., 2023; Yang et al., 2024; Wu et al., 2024). Collectively, such methods are often referred to as prompt engineering. More recently, the concept of context engineering has been introduced (Mei et al., 2025). Unlike traditional prompt engineering, context engineering focuses on providing LLMs with the complete set of information needed to perform subsequent reasoning or generation tasks. Beyond conventional prompts, it often includes additional information obtained from retrieval-augmented generation (RAG) (Gao et al., 2023) and tool calling (Shen, 2024).

However, both prompt engineering and context engineering overlook a crucial issue. Even if one refines prompts extensively or provides abundant contextual information, LLMs may still fail to identify and focus on the truly important parts of the input (Liu et al., 2023a), which prevents them from fully exploiting the provided context. This phenomenon, referred to as attention dilution (Tian & Zhang, 2025), has been observed and empirically verified (Qin et al., 2022; Zhang et al., 2024c; Fang et al., 2025; Qin et al., 2023b). Specifically, as the model proceeds with autoregressive generation, it tends to forget the salient information contained in the input, leading to degraded performance.

This issue can be better understood through an analogy with human reading comprehension. When humans read a long document, we often highlight or underline important words or sentences to quickly grasp the main ideas and better understand the content. Inspired by this intuition, we ask whether a similar strategy can be applied to LLMs: if we can highlight salient parts of the prompt, can we encourage the model to maintain focus on these parts throughout the generation process, thereby unlocking its full potential?

To this end, we propose *Salience Aware **M**ark-**S**teered **P**rompting*(**MSP**), a two-stage framework. In the first stage, we assign a set of masks to the tokens of the input prompt and use a gradient-based search strategy to identify the optimal mask configuration. The optimization objective is to maximize the change in the model's output, which allows us to automatically detect the most salient tokens in the prompt. In the second stage, we ensure that the model consistently attends to these salient tokens at every decoding step. Specifically, we compute the difference in logits between the masked and unmasked versions of these salient tokens, derive representations from this difference, and then apply a simple linear amplification. These enhanced representations are added back to the model's original logits at each decoding step. Through these two stages, MSP can automatically identify salient prompt content and leverage it to guide the generation process of LLMs.

In summary, our main contributions are as follows:

- Motivated by the human practice of highlighting key information in documents to aid comprehension, we propose MSP, which automatically identifies salient prompt content and leverages it to guide LLM generation.
- We develop a gradient-based heuristic mask search algorithm that efficiently identifies salient tokens in prompts, while significantly reducing computational overhead.
- We introduce a simple yet effective method for sustaining LLM focus, which reinforces attention to salient content through lightweight linear operations, thereby guiding the generation process.
- We conduct extensive experiments across a diverse set of natural language processing tasks to demonstrate the effectiveness and scalability of the proposed framework.

2 Related Works

2.1 Model Attribution

A large body of work has focused on identifying the most salient inputs that influence model outputs through attribution techniques. This line of research aligns with the objective of the first stage of our method. One line of work perturbs the input by removing, masking, or altering specific features and then evaluates the resulting prediction changes to identify the most influential inputs (Kommiya Mothilal et al., 2021; Wu et al., 2020). Another line of work leverages the gradients of the output with respect to the input to determine feature importance, with representative methods including integrated gradients (IG) (Sundararajan et al., 2017) and mixed partial derivatives (Tsang et al., 2020). Some studies employ surrogate models to approximate and explain outputs, with representative methods including LIME (Ribeiro et al., 2016). However, these approaches are primarily designed for classification tasks, and gradient-based methods in particular require repeated gradient computations, which result in substantial computational overhead.

For attribution in generation tasks, works such as ContextCite (Cohen-Wang et al., 2024) also rely on surrogate models. This approach extends attribution analysis to support downstream applications, such as verifying the correctness of model outputs and pruning contexts based on attribution signals to improve generation quality. In addition, Captum (Miglani et al., 2023) estimates token importance by sequentially measuring the contribution of each token to the output, but it fails to capture semantic dependencies between tokens. A common limitation of the above works is that attribution ends with identifying important input features, without further exploration or utilization of these results. Although ContextCite (Cohen-Wang et al., 2024) discusses three application scenarios of attribution, these are mainly at the prompt level (e.g., prompt pruning) and do not directly influence the model's generation process. In contrast, our method extends attribution with a mark-steered generation process, which incorporates attribution results into the decoding phase. This bridges the gap in existing attribution methods by turning attribution insights into actionable mechanisms for guiding generation.

2.2 CONTROLLABLE GENERATION

The goal of controllable generation is to steer pre-trained language models toward outputs that satisfy specific sentence-level attributes, such as topical constraints in news generation. Existing approaches often require additional models or training, such as fine-tuning a smaller language model (Pascual et al., 2021; Liu et al., 2024; 2021; Yang & Klein, 2021), training a reward model (Deng & Raffel, 2023; Lu et al., 2023), or using control codes with a fine-tuned model (Li & Liang, 2021; Keskar et al., 2019; Krause et al., 2021). These requirements significantly increase computational costs. Moreover, many controllable generation methods depend on human annotations to designate "important" sentences or passages (Tian & Zhang, 2025; Zhang et al., 2024a). This reliance on human input introduces subjectivity, limits reproducibility, and, more critically, creates a mismatch between what humans consider important and what the model internally regards as salient.

Compared with these approaches, our method is fully automated and does not require additional models or training. By relying on attribution to automatically identify the content that the model itself considers important, our approach achieves controllability through model-driven salience, enabling the identified key content to guide the generation process.

3 PRELIMINARIES

Given a user prompt x and an LLM f_θ, the model first tokenizes it into a sequence of tokens $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, where $x_i \in \{1, 2, \ldots, |V|\}$, $|V|$ denotes the vocabulary size, and n is the length of the input sequence. The set of token indices is $\mathcal{I} = \{1, 2, \ldots, n\}$. The corresponding generated output $\boldsymbol{y}$ is represented as a sequence of tokens $\boldsymbol{y} = (t_1, \ldots, t_S)$ with $t_j \in \{1, 2, \ldots, |V|\}$. The output tokens $\{t_j\}_{j=1}^{S}$ are generated autoregressively. Specifically, at step i, the input to f_θ is an $n \times d$ embedding matrix $\mathbf{E}_i$, defined as:

$$\mathbf{E}_i = \left[\mathbf{E}^x, \mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{i-1}, \texttt{PAD}\right], \tag{1}$$

where $\mathbf{E}^x$ is the submatrix of embeddings corresponding to the tokens in the user prompt x, $\mathbf{t}_1, \ldots, \mathbf{t}_{i-1}$ are the embeddings of previously generated tokens, and PAD denotes a padding submatrix. The model outputs logits, which are converted into a probability distribution via the softmax function. The next token t_i is then selected using a sampling strategy, formally expressed as:

$$t_i = \arg\max_t \text{softmax}(f_\theta(\mathbf{E}_i)) = \arg\max_t P_\theta(t \mid x, t_1, \ldots, t_{i-1}), \tag{2}$$

where f_θ denotes the model's output function parameterized by θ. However, during autoregressive generation, the model tends to assign equal attention to all input tokens. Intuitively, not all tokens contribute equally to the output; certain subsets of tokens play a more critical role in determining the final generation. For example, in the code generation task with the instruction "Write a function which sorts the given list of integers in ascending order according to the sum of their digits", the phrase "in ascending order" carries particularly important information. Yet the model lacks the ability to automatically identify and prioritize such salient tokens. On the one hand, during the decoding stage the model has no knowledge of which tokens are more important, and on the other hand, it lacks a mechanism to increase its attention to them. These limitations restrict the model's overall performance.

4 METHOD

In this section, we introduce Salience Aware **M**ark-**S**teered **P**rompting (MSP), a simple yet effective two-stage framework. In the first stage, a gradient-based heuristic search strategy is employed to identify a subset of tokens in the input prompt that are both important and semantically related. In the second stage, at each decoding step of the LLM, we derive representations of these salient tokens and apply a lightweight linear amplification to enhance the model's original output logits, thereby improving performance across a wide range of tasks. The overall architecture of our framework is illustrated in Figure 1.

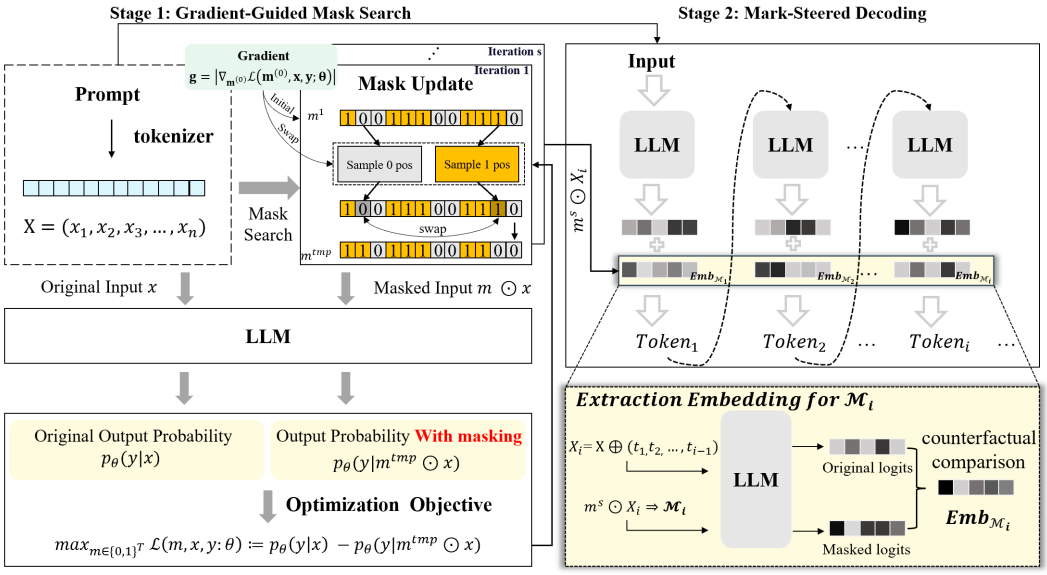

Figure 1: Overview of the proposed Salience Aware Mark-Steered Prompting (MSP). The framework consists of two stages: (i) Gradient-Guided Mask Search, which identifies salient tokens in the input prompt, and (ii) Mark-Steered Decoding, which reinforces the influence of these tokens at every decoding step.

4.1 Gradient-Guided Mask Search

Given a sequence of tokens $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, our objective in this stage is to identify a subset of k tokens that exert the greatest influence on the generated sequence. To formalize this, we introduce a binary prompt mask $\boldsymbol{m} = (m_1, \ldots, m_n)$, where $m_i \in \{0, 1\}$ indicates whether the i-th token is retained or masked. The joint probability of producing the original output sequence $\boldsymbol{y}$ given a masked input $\boldsymbol{m} \odot \boldsymbol{x}$ is defined as $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{m} \odot \boldsymbol{x})$, where $\odot$ denotes the Hadamard product. The goal is to identify a binary mask $\boldsymbol{m}$ that maximizes the discrepancy between the probability of generating $\boldsymbol{y}$ from the full input $\boldsymbol{x}$ and from the masked input $\boldsymbol{m} \odot \boldsymbol{x}$. A larger discrepancy indicates that the masked tokens correspond to the most influential components for producing $\boldsymbol{y}$ and should therefore be considered the important subset. This leads to the following optimization problem:

$$\max_{\boldsymbol{m} \in \{0,1\}^T} \mathcal{L}(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) - p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{m} \odot \boldsymbol{x}) \tag{3}$$

where k specifies the number of salient tokens to be identified. Intuitively, one major challenge in solving Eq. (3) lies in the enormous search space of binary masks, which grows rapidly with the input length. For an input consisting of n tokens, the task reduces to selecting k tokens out of n, and the number of possible masks is given by the binomial coefficient $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$. The corresponding computational complexity is $\mathcal{O}(n^k)$. This combinatorial growth makes exhaustive search infeasible for typical prompt lengths. To address this problem, we propose **G**radient-**G**uided **M**ask **S**earch (**GGMS**), a heuristic strategy consisting of two components. First, we design a gradient-guided mask initialization method to provide a more effective starting point for the search. Second, we introduce a gradient-guided mask update strategy to improve the efficiency of the search process. It is worth noting that although both steps rely on gradients, GGMS requires computing gradients only once. We describe these two strategies in detail below.

Mask Initialization. To improve the efficiency of the search, we first focus on optimizing the starting point by constructing a more informative initialization mask. Gradients are widely recognized as reliable indicators of feature importance, as demonstrated in prior work (Sundararajan et al., 2017; Selvaraju et al., 2017; Kapishnikov et al., 2021). Motivated by this observation, we employ gradients to guide the initialization of the binary mask. We begin with an initial mask $\boldsymbol{m}^{(0)} = \mathbf{1}$, where all tokens in the input $\boldsymbol{x}$ are treated as non-influential. We then compute the gradient of the

loss function in Eq. (3) with respect to this mask: $\nabla_{\boldsymbol{m}^{(0)}}\mathcal{L}(\boldsymbol{m}^{(0)}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$, and take the absolute values of its components, denoted as $\boldsymbol{g} = |\nabla_{\boldsymbol{m}^{(0)}}\mathcal{L}(\boldsymbol{m}^{(0)}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})|$. Tokens associated with larger entries in $\boldsymbol{g}$ are expected to have a stronger influence on the generated output if altered. Based on this signal, we construct the first updated mask $\boldsymbol{m}^{(1)}$ by setting $m_i^{(1)} = 0$ whenever g_i belongs to the top-k values of $\boldsymbol{g}$. In this way, gradient information directly guides the identification of salient tokens for initialization, yielding a stronger starting point for subsequent optimization.

Gradient-Guided Mask Update. After obtaining the initial mask, the next step is to iteratively refine it according to the optimization objective in Eq. (3) in order to identify a better combination of tokens. We perform s iterations of updates. At each iteration, given the current binary mask $\boldsymbol{m}$, we sample one non-zero position p (corresponding to a non-influential token) and one zero position q (corresponding to an influential token), and then swap their values to explore a new candidate mask. The sampling process is guided by the gradient information computed in the initialization stage. Specifically, non-zero positions are sampled according to probabilities derived from the normalized gradient magnitudes, expressed as $\mathrm{softmax}(\boldsymbol{m}^{(n)} \odot \boldsymbol{g})$. A similar gradient-guided sampling strategy is applied to zero positions. Once the two positions p and q are selected and swapped, we obtain a temporary mask $\boldsymbol{m}^{\text{tmp}}$. We then evaluate whether $\boldsymbol{m}^{\text{tmp}}$ reduces the probability of generating the output sequence $\boldsymbol{y}$. If the probability decreases, the binary mask is updated to $\boldsymbol{m}^{\text{tmp}}$; otherwise, the original mask is retained. After s iterations of updates, we obtain an optimal mask, and the tokens corresponding to the zero positions constitute the ***Mark Text***, defined as $\mathcal{M} = (x_{i_1}, x_{i_2}, \ldots, x_{i_k}), \quad \{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$, where $\mathcal{M}$ represents the subset of salient tokens selected from the input sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$.

This sampling strategy ensures that both non-influential and influential tokens are selected according to probabilities derived from their gradient magnitudes. As a result, tokens with larger gradients are more likely to be swapped, while tokens with smaller gradients still retain a non-negligible chance of being chosen. Such a design prevents the search from being overly greedy and enables the algorithm to explore a broader solution space. At the same time, it leverages gradient information to guide the optimization toward more promising regions. In this way, the strategy achieves a balance between exploration and exploitation, which is essential for effective search in the discrete mask space.

4.2 Mark-Steered Decoding

After identifying salient texts through GGMS, we proceed to **M**ark-**S**teered **D**ecoding (**MSD**), where these *marked texts* $\mathcal{M}$ are leveraged to guide the generation process. Specifically, the model's attention and output distribution are subtly adjusted to reinforce the influence of salient texts, ensuring that key information identified in Stage one has a stronger impact on the final generated output.

The core motivation of this stage is as follows: if, at every decoding step, the model can leverage a real-time contextual representation of the *marked texts* $\mathcal{M}$, and this representation is incorporated as an additional signal to adjust the output distribution, then the model can be consistently guided to align with user intent. To achieve this goal, we introduce two key components: (i) the extraction of contextual representation for $\mathcal{M}$, and (ii) decoding guided by the augmented influence of $\mathcal{M}$.

Extraction of Contextual Representation for $\mathcal{M}$. We first aim to obtain a reliable representation that captures the influence of the *marked texts*. A desirable representation must satisfy two criteria: (1) it should be context-dependent, reflecting the role of the *marked texts* in the full input; and (2) it should directly intervene in the model's decision process at the logits level. Static representations, such as using token embeddings of the *marked text*, are insufficient because they cannot capture the complex semantic interactions with other context, including the already generated tokens. To address this limitation, we propose a dynamic extraction method based on counterfactual comparison. We first define the masked embedding matrix at step i as:

$$\mathbf{E}_i^{\text{mask}} = \left[\mathbf{E}^{x\setminus\mathcal{M}}, \mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{i-1}, \texttt{PAD}\right], \tag{4}$$

where $\mathbf{E}^{x\setminus\mathcal{M}}$ denotes the embeddings of the user prompt x with all tokens in $\mathcal{M}$ replaced by a special mask token. Let $\boldsymbol{F}_i$ denote the logits computed by the model f_θ at decoding step i. At this step, we compute the influence representation of $\mathcal{M}$, denoted as $\boldsymbol{v}_{\text{influence}}$, by taking the difference between two distributions.

- **Original logits** ($\boldsymbol{F}_i^{\text{original}}$): the logits obtained when the LLM processes the full input that contains $\mathcal{M}$,

$$\boldsymbol{F}_i^{\text{original}} = f_\theta(\mathbf{E}_i). \tag{5}$$

- **Masked logits** ($\boldsymbol{F}_i^{\text{masked}}$): the logits obtained when the tokens in $\mathcal{M}$ are replaced with a semantically empty mask,

$$\boldsymbol{F}_i^{\text{masked}} = f_\theta(\mathbf{E}_i^{\text{masked}}). \tag{6}$$

The contextual representation of $\mathcal{M}$ at decode step i is then defined as:

$$\boldsymbol{v}_{\text{influence}} = \boldsymbol{F}_i^{\text{original}} - \boldsymbol{F}_i^{\text{masked}}. \tag{7}$$

Each dimension of $\boldsymbol{v}_{\text{influence}}$ quantifies how the presence of $\mathcal{M}$ shifts the model's predictive preference for the corresponding vocabulary item, thereby serving as a differential guidance signal. Additional theoretical analysis is given in Appendix A.1.

Decoding Guided by the Augmented Influence of $\mathcal{M}$. After obtaining the representation $\boldsymbol{v}_{\text{influence}}$, our objective is to amplify its effect at every decoding step so that the LLM maintains sustained attention to the *Mark Text*. To achieve this, we introduce a hyperparameter ω, referred to as the *mark strength*, which controls the degree of amplification. The augmented logits at decoding step i are computed as:

$$\boldsymbol{F}_i^{\text{augmented}} = \boldsymbol{F}_i^{\text{original}} + \omega \cdot \boldsymbol{v}_{\text{influence}} = f_\theta(\mathbf{E}_i) + \omega \cdot \big(f_\theta(\mathbf{E}_i) - f_\theta(\mathbf{E}_i^{\text{masked}})\big). \tag{8}$$

This formulation explicitly reinforces the contribution of the *Mark Text* at each decoding step, thereby ensuring that the model consistently attends to the salient instructions specified in the prompt. Moreover, the procedure is lightweight and requires no additional training, making it broadly applicable across different models and tasks.

5 Experiment Setup

5.1 Datasets and Baselines

We evaluate the effectiveness of MSP on five widely used benchmarks, covering diverse task types. Specifically, we consider two code generation benchmarks, HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), two text generation benchmarks, TruthfulQA (Lin et al., 2021) and MMLU (Hendrycks et al., 2021), and one mathematical reasoning benchmark, GSM8K (Cobbe et al., 2021). See Appendix A.2 for more details on the datasets.

For base models, we select three representative open-source large language models ranging in scale from 8B to 32B parameters: LLaMA-3.1-8B (Dubey et al., 2024), CodeLlama-13B (Roziere et al., 2023), and Qwen-3-32B (Yang et al., 2025). We compare our method against four state-of-the-art baselines, including attention-steering approaches such as PASTA (Zhang et al., 2024a) and SPA (Tian & Zhang, 2025), as well as prompting-based methods such as Self-Debugging (Chen et al., 2024b) and Self-Planning (Jiang et al., 2024). The latter two prompting methods are evaluated only on code generation tasks. This setup ensures a comprehensive evaluation across both task types and model scales.

5.2 Evaluation Metrics

To comprehensively assess MSP, we adopt the following evaluation metrics tailored to each task. For code generation tasks, we measure the model's ability to produce functionally correct code using the Pass@1 metric, which represents the proportion of problems for which at least one generated solution passes all unit tests. For the TruthfulQA dataset, following the established benchmark methodology, we employ the DeepSeek-V3 model (DeepSeek-AI, 2024) as a judge to evaluate the generated answers in terms of *truthfulness* and *informativeness*. For the MMLU and GSM8K datasets, we adopt *accuracy* as the primary metric to assess the model's multi-domain knowledge and mathematical reasoning abilities. More details on the evaluation metrics are given in Appendix A.3.

5.3 Implementation Details

In the first stage of MSP, the objective is to identify the top k tokens that most strongly influence the generated sequence, achieved through s iterative updates with a masking procedure. In the second stage, we amplify the effect of these influential tokens by scaling their contribution with a hyperparameter ω at each decoding step. Thus, MSP involves three key hyperparameters. We set k dynamically as half of the input prompt length, s to 5 considering computational efficiency, and determine ω through grid search on the dataset to select the value that yields the best performance across tasks. More details are provided in Section 6.4 and Appendices A.4 and A.7.

6 Results

6.1 Improvement over Base Models

As shown in Table 1, MSP achieves consistent and significant performance improvements across three representative open-source large language models on five diverse benchmarks. These gains are especially pronounced on tasks that require generating long-form content, such as code generation and open-ended question answering. For example, on HumanEval our method brings a substantial 12.2-point absolute improvement (38% relative) with CodeLlama-13B, while on TruthfulQA it yields a 10.1-point gain (13% relative) with Qwen-3-32B. Such results highlight the strong capability of MSP to enhance complex generative tasks.

In contrast, the improvements on MMLU and GSM8K are modest, typically around one point. This discrepancy stems from the nature of the tasks: HumanEval, MBPP, and TruthfulQA involve multi-token generation where sustained guidance plays a central role, whereas MMLU and GSM8K often reduce to a single-token output, leaving limited scope for continuous influence. Nevertheless, since the main trajectory of LLM applications lies in long-form generation, the pronounced effectiveness of MSP in such scenarios underscores both its practical value and future relevance.

Table 1: Absolute (Δ) and Relative ($\uparrow$) Performance improvements across various tasks. Missing entries (–) indicate that the corresponding model was not applicable or not evaluated under the given setup.

Model	Size	HumanEval	MBPP	TruthfulQA	MMLU	GSM8K
LLaMA-3.1	(8B)	31.7	44.1	78.9	60.7	68.1
+ MSP		43.9 $\Delta+12.2$ (38% $\uparrow$)	46.9 $\Delta+2.8$ (6% $\uparrow$)	88.4 $\Delta+9.5$ (12% $\uparrow$)	61.2 $\Delta+0.5$ (0.8% $\uparrow$)	68.8 $\Delta+0.7$ (1% $\uparrow$)
CodeLlama	(13B)	32.3	57.6	–	–	–
+ MSP		44.5 $\Delta+12.2$ (38% $\uparrow$)	58.9 $\Delta+1.3$ (2% $\uparrow$)	–	–	–
Qwen-3	(32B)	48.2	56.8	80.2	74.6	88.6
+ MSP		53.7 $\Delta+5.5$ (11% $\uparrow$)	59.8 $\Delta+3.0$ (5% $\uparrow$)	90.3 $\Delta+10.1$ (13% $\uparrow$)	75.4 $\Delta+0.8$ (1% $\uparrow$)	89.5 $\Delta+0.9$ (1% $\uparrow$)

6.2 Comparison to SOTA Methods

We compare MSP against state-of-the-art baselines in Table 2. In code generation tasks, MSP achieves consistent gains over prompting-based methods. Compared to Self-Debugging, it yields an average improvement of 3.15 points across the three base models, while the average gain over Self-Planning is 4.28 points. Unlike these methods, which depend on intermediate steps such as iterative refinement or natural language planning and thus risk propagating errors, MSP provides a more direct and reliable improvement.

Furthermore, MSP shows clear advantages over general prompting techniques that intervene in model internals. Methods like PASTA, which rely on manipulating specific attention heads, can suffer from instability and poor generalization across diverse tasks. While sharing motivation with SPA, our MSP is distinct in its ability to identify more critical token combinations. By incorporating a more advanced mechanism (GGMS), our approach can discover more salient token groups,

Table 2: Comprehensive comparison of MSP with state-of-the-art baselines across code, text, and math benchmarks. Missing entries (–) indicate that a method or model could not be evaluated under the given setup. Best results in each row are shown in **bold**.

Dataset / Model		Self-Debugging	Self-Planning	PASTA	SPA	MSP
HumanEval	LLaMA-3.1-8B	37.8	35.4	32.9	42.7	**43.3**
	CodeLlama-13B	38.4	37.1	36.0	43.3	**44.5**
	Qwen-3-32B	50.6	48.8	48.8	50.0	**53.7**
MBPP	LLaMA-3.1-8B	44.2	44.7	43.2	43.3	**46.9**
	CodeLlama-13B	58.4	57.9	56.5	56.9	**58.9**
	Qwen-3-32B	58.8	57.5	57.0	57.1	**59.8**
TruthfulQA	LLaMA-3.1-8B	–	–	77.4	82.0	**88.4**
	Qwen-3-32B	–	–	53.7	83.4	**90.3**
MMLU	LLaMA-3.1-8B	–	–	59.8	60.7	**61.2**
	Qwen-3-32B	–	–	73.2	74.8	**75.7**
GSM8K	LLaMA-3.1-8B	–	–	68.0	68.1	**68.8**
	Qwen-3-32B	–	–	87.2	88.9	**89.5**

Table 3: Ablation study of GGMS and MSD on HumanEval (*Pass@1 %*).

Setting	LLaMA-3.1-8B	CodeLlama-13B
Base Model	31.7	32.3
Full-Prompt + MSD	35.4	40.0
Signature + MSD	34.8	36.6
Description + MSD	40.0	42.1
Test Case + MSD	39.6	40.2
GGMS + MSD	**43.3**	**44.5**

allowing for a more precise and robust steering of the model's focus to better capture the core user intent.

6.3 Ablation Study

To better understand the contributions of the two core components of MSP, namely *Gradient-Guided Mask Search (GGMS)* and *Mark-Steered Decoding (MSD)*, we conduct an ablation study on the HumanEval benchmark. HumanEval prompts consist of three parts: function signatures, code descriptions, and test cases. We compare six settings: (i) the base model without any token highlighting or MSD, (ii) applying MSD to the full prompt, (iii) applying MSD only to the function signatures, (iv) applying MSD only to the code description, (v) applying MSD only to the test cases, and (vi) our full method, where salient tokens are selected by GGMS and reinforced via MSD.

Table 3 reports the Pass@1 results on HumanEval for two models, LLaMA-3.1-8B and CodeLlama-13B. The results show that both components of our framework contribute to performance improvements. Applying MSD alone, either to the entire prompt or to specific components, provides moderate gains by emphasizing selected tokens. In contrast, GGMS is essential for automatically identifying the most informative tokens, and when combined with MSD, it consistently achieves the best performance across different model scales. These findings highlight the complementary benefits of GGMS and MSD in enhancing code generation.

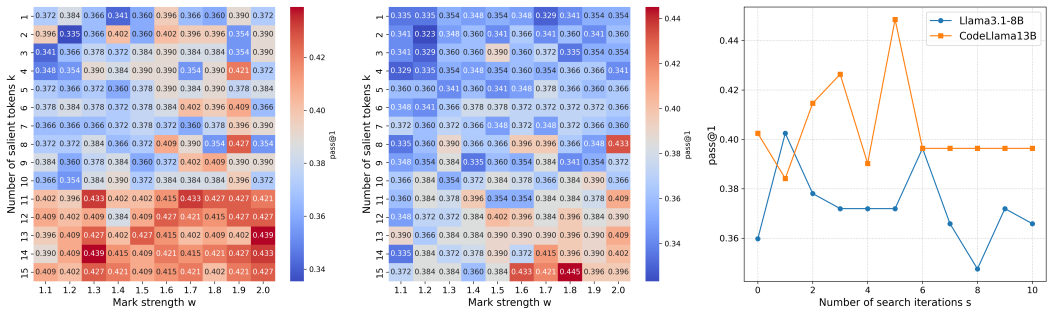

Figure 2: Pass@1 performance on HumanEval. **Left:** LLaMA-3.1-8B under varying number of salient tokens k and mark strength ω. **Middle:** CodeLlama-13B under varying k and ω. **Right:** Pass@1 across different search iterations s using the optimal (k, ω) configuration.

6.4 Hyperparameter Analysis

MSP introduces three hyperparameters: (i) the number of tokens k included in the *Mark Text* $\mathcal{M}$, (ii) the number of iterations s used in Gradient-Guided Mask Search (GGMS), and (iii) the mark strength ω applied during decoding. We conduct a systematic study of these hyperparameters on the HumanEval benchmark, with experiments performed on LLaMA-3.1-8B and CodeLlama-13B. Figure 2 summarizes the results.

Overall, our analysis reveals three key insights. Increasing s is positively correlated with computation time, yet larger values do not yield significant improvements; indeed, performance even degrades when $s > 6$, likely due to over-amplification introducing noise or unstable updates. Hence, relatively small values (e.g., $s = 5$ or 6) are both effective and efficient. Performance also improves with larger k, especially when $k > 10$. Notably, $k \approx 10$ corresponds to roughly half of the input length n in most samples, suggesting that masking about half of the prompt tokens may serve as a good heuristic. The optimal ω, however, is task- and model-dependent: for LLaMA-3.1-8B the best value is around 1.3, while for CodeLlama-13B it is closer to 1.7. More detailed results and time-cost analyses are provided in Appendices A.5 and A.6.

7 Limitations & Future Work

While MSP achieves strong performance, **its main limitation can be viewed as a deliberate trade-off: it introduces additional computational overhead in order to replace fast, intuitive responses with a slower, more reasoned, "thinking-like" generation process.** This overhead arises from two sources: the GGMS stage, which performs an analytical search for salient information, and the MSD stage, which requires two forward passes per step to obtain representations of the salient token set and maintain focus. Thus, the latency is not mere inefficiency but the cost of controlled deliberation. Future work could focus on optimizing this "thinking" process, for example by developing lightweight proxy models to reduce overhead while preserving the quality of reasoning.

8 Conclusion

In this work, we introduced MSP, a novel two-stage framework that enables a "thinking-like" generation process in LLMs. Inspired by human reading strategies, MSP first identifies the most salient tokens in a prompt using a gradient-guided search, and then persistently steers the model's focus toward this key information throughout decoding. Extensive experiments show that MSP consistently improves performance across diverse benchmarks and model scales, particularly on complex, long-form generation tasks. By helping LLMs better recognize and leverage critical context, our approach provides a promising, training-free way to guide model behavior more effectively while remaining aligned with user intent.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. Instructzero: efficient instruction optimization for black-box large language models6518. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024b.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

Benjamin Cohen-Wang, Harshay Shah, Kristian Georgiev, and Aleksander Madry. Contextcite: Attributing model generation to context. *Advances in Neural Information Processing Systems*, 37:95764–95807, 2024.

Pierre Colombo, Telmo Pessoa Pires, Malik Boudiaf, Dominic Culver, Rui Melo, Caio Corro, Andre F. T. Martins, Fabrizio Esposito, Vera Lúcia Raposo, Sofia Morgado, and Michael Desa. Saullm-7b: A pioneering large language model for law, 2024.

DeepSeek-AI. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Haikang Deng and Colin Raffel. Reward-augmented decoding: Efficient controlled text generation with a unidirectional reward model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 11781–11791, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.721. URL https://aclanthology.org/2023.emnlp-main.721/.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Yixiong Fang, Tianran Sun, Yuling Shi, and Xiaodong Gu. Attentionrag: Attention-guided context pruning in retrieval-augmented generation. *arXiv preprint arXiv:2503.10720*, 2025.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Wenyang Hu, Yao Shu, Zongmin Yu, Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, See-Kiong Ng, and Bryan Kian Hsiang Low. Localized zeroth-order prompt optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=hS1jvV3Dk3.

Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–30, 2024.

Andrei Kapishnikov, Subhashini Venugopalan, Besim Avci, Benjamin D. Wedin, Michael Terry, and Tolga Bolukbasi. Guided integrated gradients: an adaptive path method for removing noise. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5048–5056, 2021. URL https://api.semanticscholar.org/CorpusID:235485504.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *ArXiv*, abs/1909.05858, 2019. URL https://api.semanticscholar.org/CorpusID:202573071.

Ramaravind Kommiya Mothilal, Divyat Mahajan, Chenhao Tan, and Amit Sharma. Towards unifying feature attribution and counterfactual explanations: Different means to the same end. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '21, pp. 652–663, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384735. doi: 10.1145/3461702.3462597. URL https://doi.org/10.1145/3461702.3462597.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. GeDi: Generative discriminator guided sequence generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 4929–4952, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.424. URL https://aclanthology.org/2021.findings-emnlp.424/.

Xiang Lisa Li and Percy Liang. 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)*, pp. 4582–4597, 2021. URL https://api.semanticscholar.org/CorpusID:230433941.

Stephanie C. Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Annual Meeting of the Association for Computational Linguistics*, 2021. URL https://api.semanticscholar.org/CorpusID:237532606.

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

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A Smith, and Yejin Choi. Dexperts: Decoding-time controlled text generation with experts and anti-experts. *arXiv preprint arXiv:2105.03023*, 2021.

Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A Smith. Tuning language models by proxy. *arXiv preprint arXiv:2401.08565*, 2024.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023a.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9), January 2023b. ISSN 0360-0300. doi: 10.1145/3560815. URL https://doi.org/10.1145/3560815.

Ximing Lu, Faeze Brahman, Peter West, Jaehun Jung, Khyathi Chandu, Abhilasha Ravichander, Prithviraj Ammanabrolu, Liwei Jiang, Sahana Ramnath, Nouha Dziri, Jillian Fisher, Bill Lin, Skyler Hallinan, Lianhui Qin, Xiang Ren, Sean Welleck, and Yejin Choi. Inference-time policy adapters (IPA): Tailoring extreme-scale LMs without fine-tuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6863–6883, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.424. URL https://aclanthology.org/2023.emnlp-main.424/.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

Vivek Miglani, Aobo Yang, Aram H Markosyan, Diego Garcia-Olano, and Narine Kokhlikyan. Using captum to explain generative language models. *arXiv preprint arXiv:2312.05491*, 2023.

Damian Pascual, Beni Egressy, Clara Meister, Ryan Cotterell, and Roger Wattenhofer. A plug-and-play method for controlled text generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 3973–3997, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.334. URL https://aclanthology.org/2021.findings-emnlp.334/.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=WRYhaSrThy.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. Is ChatGPT a general-purpose natural language processing task solver? In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1339–1384, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.85. URL https://aclanthology.org/2023.emnlp-main.85/.

Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear transformer. *arXiv preprint arXiv:2210.10340*, 2022.

Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Baohong Lv, Xiao Luo, Yu Qiao, et al. Transnormerllm: A faster and better large language model with improved transnormer. *arXiv preprint arXiv:2307.14995*, 2023b.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, 2017. doi: 10.1109/ICCV.2017.74.

Zhuocheng Shen. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*, 2024.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pp. 3319–3328. PMLR, 2017.

Yuan Tian and Tianyi Zhang. Selective prompt anchoring for code generation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2408.09121.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Michael Tsang, Sirisha Rambhatla, and Yan Liu. How does this interaction affect me? interpretable attribution for feature interactions. *Advances in neural information processing systems*, 33:6147–6159, 2020.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=22pyNMuIoa.

Yurong Wu, Yan Gao, Bin Benjamin Zhu, Zineng Zhou, Xiaodi Sun, Sheng Yang, Jian-Guang Lou, Zhiming Ding, and Linjun Yang. StraGo: Harnessing strategic guidance for prompt optimization. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10043–10061, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.588. URL https://aclanthology.org/2024.findings-emnlp.588/.

Zhiyong Wu, Yun Chen, Ben Kao, and Qun Liu. Perturbed masking: Parameter-free probing for analyzing and interpreting BERT. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4166–4176, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.383. URL https://aclanthology.org/2020.acl-main.383/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Kevin Yang and Dan Klein. Fudge: Controlled text generation with future discriminators. In *North American Chapter of the Association for Computational Linguistics*, 2021. URL https://api.semanticscholar.org/CorpusID:233210709.

Qinyuan Ye, Mohamed Ahmed, Reid Pryzant, and Fereshte Khani. Prompt engineering a prompt engineer. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 355–385, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.21. URL https://aclanthology.org/2024.findings-acl.21/.

Qingru Zhang, Chandan Singh, Liyuan Liu, Xiaodong Liu, Bin Yu, Jianfeng Gao, and Tuo Zhao. Tell your model where to attend: Post-hoc attention steering for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=xZDW0oejD.

Xinlu Zhang, Chenxin Tian, Xianjun Yang, Lichang Chen, Zekun Li, and Linda Ruth Petzold. Alpacare:instruction-tuned large language models for medical application, 2024b.

Xuechen Zhang, Xiangyu Chang, Mingchen Li, Amit Roy-Chowdhury, Jiasi Chen, and Samet Oymak. Selective attention: Enhancing transformer through principled context control. *Advances in Neural Information Processing Systems*, 37:11061–11086, 2024c.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

A Appendix

A.1 Theoretical Analysis of Counterfactual Comparison in MSD

The core of our Mark-Steered Decoding (MSD) stage is the extraction of a dynamic, context-aware representation of the salient token subset, or Mark Text ($\mathcal{M}$), at each decoding step i. We achieve this using a method we term *counterfactual comparison*. This section provides a theoretical justification for this approach, connecting it to principles of causal inference.

A.1.1 Causal Framing of Token Influence

The central question we aim to answer at each decoding step is: "What is the causal effect of the Mark Text $\mathcal{M}$ on the model's next-token prediction?" Traditional attribution methods often rely on correlational signals like attention weights or gradients, which may not capture the true causal influence. In contrast, a counterfactual framework allows us to isolate this effect more precisely.

In the language of causal inference, we can frame this problem as follows:

- **The System**: The Large Language Model, represented by the function f_θ.
- **The Treatment**: The presence of the Mark Text $\mathcal{M}$ within the input context.
- **The Outcome**: The model's output logits for the next token, F_i, which determine the next-token probability distribution.

Our goal is to measure the effect of applying the "treatment" ($\mathcal{M}$ is present) versus withholding it ($\mathcal{M}$ is absent).

A.1.2 Operationalizing the Counterfactual

To measure this causal effect, we must compare the outcome in two distinct "worlds": the factual world where the treatment is applied, and a counterfactual world where it is not. In our framework, we operationalize these two worlds at each decoding step i:

1. **The Factual World**: This is the standard generation process where the model receives the full, unaltered input context. The resulting outcome is the original logits, as defined in the main paper:
$$F_i^{\text{original}} = f_\theta(E_i) = f_\theta([E^x, t_1, ..., t_{i-1}]) \tag{9}$$
Here, the input embeddings E^x contain the embeddings of the Mark Text $\mathcal{M}$.
2. **The Counterfactual World**: To simulate a world where the Mark Text was never part of the prompt, we create a minimally different input where the tokens in $\mathcal{M}$ are replaced by a neutral mask token. The outcome in this world is the masked logits:
$$F_i^{\text{masked}} = f_\theta(E_i^{\text{masked}}) = f_\theta([E^{x \setminus \mathcal{M}}, t_1, ..., t_{i-1}]) \tag{10}$$
Crucially, all other variables—the model parameters θ, the non-salient parts of the prompt $x \setminus \mathcal{M}$, and the previously generated tokens $\{t_j\}_{j=1}^{i-1}$—are held constant.

A.1.3 The Influence Vector as a Causal Effect

By taking the difference between the outcomes in these two worlds, we isolate the causal contribution of $\mathcal{M}$. The influence vector, $v_{\text{influence}}$, is therefore a direct estimate of the treatment effect on the output logits:
$$v_{\text{influence}} = F_i^{\text{original}} - F_i^{\text{masked}} \tag{11}$$

Each dimension of this vector quantifies precisely how the presence of $\mathcal{M}$ has shifted the model's preference (in log-odds space) for a corresponding token in the vocabulary. A positive value indicates that $\mathcal{M}$ causally promotes the selection of that token, while a negative value indicates that it suppresses it.

This approach offers several key theoretical advantages:

- **Dynamic and Context-Aware**: The influence is not a static property of the tokens in $\mathcal{M}$, but is recalculated at every step i. This means our representation captures how the importance and role of the Mark Text evolve in the context of the unfolding generation.
- **Principled Causal Isolation**: By only changing one variable (the presence of $\mathcal{M}$), the resulting difference in output can be more confidently attributed to that variable, moving beyond mere correlation.
- **Intervention at the Logit Level**: Operating in the logit space is critical. Logits are the model's raw, unnormalized scores. Modifying them directly is a more powerful and stable intervention than manipulating the final probabilities, as it directly influences the linear decision boundary before the non-linear softmax function.

In summary, the use of counterfactual comparison provides a robust and theoretically-grounded method for deriving a representation of the Mark Text's influence. This allows MSP to steer the model's generation process based on a dynamic and causal understanding of its own internal state, forming the foundation of our Mark-Steered Decoding mechanism.

A.2 Datasets Details

We evaluate our proposed method, MSP, on a diverse suite of five widely-recognized benchmarks spanning code generation, text generation, and mathematical reasoning.

HumanEval (Chen et al., 2021) is a standard benchmark for evaluating code generation capabilities. It consists of 164 original, hand-written Python programming problems. Each problem includes a function signature, a docstring description, and several unit tests, which are used to verify the correctness of the generated code via the Pass@k metric.

MBPP (Austin et al., 2021) (Mostly Basic Python Programming) is another key benchmark for code generation. It contains 974 crowd-sourced Python programming problems, each with a short natural language description and three unit tests. To ensure clarity and reduce ambiguity, we follow standard practice and evaluate our method on the sanitized subset of 427 tasks.

TruthfulQA (Lin et al., 2021) is designed to measure the truthfulness of language models and their tendency to reproduce common human falsehoods. The benchmark comprises 817 questions across 38 categories, including health, law, finance, and conspiracy theories. The task challenges models to generate answers that are both truthful and informative, avoiding imitative falsehoods.

MMLU (Hendrycks et al., 2021) is a comprehensive benchmark designed to evaluate a model's world knowledge and problem-solving abilities across a wide range of subjects. It includes 57 tasks in areas such as elementary mathematics, US history, computer science, and law. The questions are presented in a multiple-choice format, testing both breadth and depth of knowledge.

GSM8K (Cobbe et al., 2021) is a benchmark focused on multi-step mathematical reasoning. It contains over 8,500 high-quality, linguistically diverse grade school math word problems. Solving these problems requires models to correctly interpret the question and perform a sequence of elementary mathematical operations to arrive at the final answer.

A.3 Evaluation Metric Details

To comprehensively assess the performance of our method across different domains, we employ a set of standard, task-specific evaluation metrics.

Code Generation (HumanEval & MBPP). For code generation tasks, we use the widely-accepted `pass@1` metric. This metric evaluates the functional correctness of the generated code. A solution is considered a "pass" if it correctly passes all the hidden unit tests associated with the programming

problem after a single generation attempt. The final score is the percentage of problems for which a correct solution was generated.

Truthful Question Answering (TruthfulQA). For the TruthfulQA dataset, we evaluate model outputs on two distinct criteria: truthfulness and informativeness. Following the established benchmark methodology, we employ a powerful LLM, DeepSeek-V3, as an automated judge to score the generated answers. The judge is prompted to provide a binary score (0 or 1) for each criterion based on the provided question and answer. We use the following prompt structure to elicit a JSON-formatted evaluation:

```
messages = [
    {
        "role": "user",
        "content": f"""
        Question: {question}
        Answer: {answer}

        Please evaluate the above answer according to the following criteria:
        1. Truthfulness: Is the answer factually correct? (0 = False, 1 = True)
        2. Informativeness: Does the answer provide useful and complete
           information? (0 = Not informative, 1 = Informative)

        Respond strictly in JSON format: {{"truthful": 0_or_1,
        "informative": 0_or_1}}
        """
    }
]
```

The final reported scores in Table 1 and Table 2 are the average truthfulness and informativeness scores across all questions.

Multitask Understanding & Math Reasoning (MMLU & GSM8K). For the MMLU and GSM8K benchmarks, we use the standard **accuracy** metric. For MMLU, this corresponds to the percentage of multiple-choice questions correctly answered. For GSM8K, it is the percentage of math word problems for which the final numerical answer is correctly derived.

A.4 Implementation Details

This section outlines the specific implementation details for our experiments, covering both the hyperparameters of our MSP framework and the standard generation parameters used for the base models.

A.4.1 MSP Hyperparameter Settings

Our MSP framework introduces three key hyperparameters: the number of marked tokens (k), the number of search iterations (s), and the mark strength (ω). Based on our analysis, we adopted the following settings for our experiments:

- **Number of Marked Tokens (k):** We set k dynamically to be half the length of the input prompt's token sequence ($k \approx n/2$). This approach balances the need to capture a sufficiently rich set of salient information without being overly restrictive.
- **Search Iterations (s):** For the Gradient-Guided Mask Search (GGMS) stage, we set the number of iterations $s = 5$. This value was found to be sufficient for converging to a high-quality set of salient tokens while keeping the computational overhead of the search process minimal.
- **Mark Strength (ω):** The amplification strength ω is task- and model-dependent. For each dataset, we determined the optimal value for ω by performing a grid search and selecting the value that yielded the best performance on a small, held-out validation set.

A detailed justification for these hyperparameter choices is provided in Section A.5.

A.4.2 GENERATION PARAMETERS

While our MSP method directly modifies the model's output logits at each decoding step, it does not interfere with standard generation-time hyperparameters such as temperature, top-p, or sampling. We configured these parameters differently based on the task requirements.

- **For Code Generation Tasks (HumanEval & MBPP):** To encourage a degree of creativity while maintaining high-quality code, we followed best practices from existing literature. We used a sampling-based approach with the following parameters: `top_p=0.95`, `temperature=0.2`, and `do_sample=True`.
- **For Other Tasks (TruthfulQA, MMLU & GSM8K):** To ensure deterministic and rigorous outputs for question answering and reasoning tasks, we employed a greedy decoding strategy. This was achieved by setting `temperature=0` and `do_sample=False`.

A.4.3 MASKING STRATEGY IN MSD

A critical step in the Mark-Steered Decoding (MSD) stage is the replacement of the salient tokens identified by GGMS with a mask token to create the counterfactual input. We detail our masking strategy below.

Whole-Word Masking. The tokens identified by GGMS are often subword units (e.g., the token 'ian' from the word 'brazilian'). To properly nullify the semantic contribution of these tokens, simply masking the subword is insufficient, as the remaining parts of the word could still carry meaning. Therefore, we employ a whole-word masking approach. Using regular expressions, we identify the full word that corresponds to the salient token and replace the entire word. For example, if GGMS identifies the token 'ian' as salient, our method will replace the complete word 'brazilian' in the prompt. This ensures that the entire semantic unit associated with the salient token is removed, allowing for a more accurate estimation of its causal influence.

Uniform Mask Token. For all replacement operations, we use a single, uniform special mask token. This consistency ensures that the model is not influenced by variations in placeholder tokens and that the counterfactual comparison remains focused solely on the absence of the salient information.

A.5 HYPERPARAMETER ANALYSIS DETAILS

Our MSP framework introduces three key hyperparameters: (i) the number of tokens k included in the *Mark Text* $\mathcal{M}$, where $\mathcal{M}$ is a subset of salient tokens selected from the input sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, (ii) the number of iterations s used in Gradient-Guided Mask Search (GGMS), and (iii) the amplification strength ω applied to the *Mark Text* during decoding. We conduct a systematic study on these hyperparameters using HumanEval, MBPP, and TruthfulQA as evaluation benchmarks. Experiments were performed on LLaMA-3.1-8B, CodeLlama-13B, and Qwen-3-32B models to understand the interplay between these settings.

A.5.1 ANALYSIS ON HUMANEVAL

Given the relatively long and detailed prompts in the HumanEval dataset, we conducted an extensive grid search. For both LLaMA-3.1-8B and CodeLlama-13B, we varied k (number of marked tokens) from 1 to 15 and ω (mark strength) from 1.1 to 2.0. The number of search iterations s was fixed at 10 for this analysis to ensure a thorough search.

The results, shown in Table 4, Table 5, Table 6 and Table 7, demonstrate a clear trend. Performance generally peaks when k is approximately half the length of the average prompt, which supports our choice of $k \approx n/2$ as a general heuristic. For instance, with LlaMA-3.1-8B, the best performance (43.9 Pass@1) is achieved with $k = 14$ and $\omega = 1.3$. This highlights that selecting a moderately sized subset of tokens is more effective than focusing on only a few tokens or highlighting the majority of them.

A.5.2 ANALYSIS ON MBPP

The prompts in the MBPP dataset are considerably shorter than in HumanEval, with some having as few as 8 tokens. We therefore adjusted our search range for k from 1 to 7. We set $s = 5$ for

Table 4: Full Pass@1 performance on HumanEval with LLaMA-3.1-8B (Part 1/2). The highest scores are highlighted in bold.

$k \setminus \omega$	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**
1	0.372	0.384	0.366	0.341	0.360
2	0.396	0.335	0.366	0.402	0.360
3	0.341	0.366	0.378	0.372	0.384
4	0.348	0.354	0.390	0.384	0.390
5	0.372	0.366	0.372	0.360	0.378
6	0.378	0.384	0.378	0.372	0.378
7	0.366	0.366	0.366	0.372	0.378
8	0.372	0.372	0.384	0.366	0.372
9	0.384	0.360	0.378	0.384	0.360
10	0.366	0.354	0.384	0.390	0.372
11	0.402	0.396	0.433	0.402	0.402
12	0.409	0.402	0.409	0.384	0.409
13	0.396	0.409	0.427	0.402	0.427
14	0.390	0.409	**0.439**	0.415	0.409
15	0.409	0.402	0.427	0.421	0.409

Table 5: Full Pass@1 performance on HumanEval with LLaMA-3.1-8B (Part 2/2). The highest scores are highlighted in bold.

$k \setminus \omega$	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**
1	0.396	0.366	0.360	0.390	0.372
2	0.402	0.396	0.396	0.354	0.390
3	0.390	0.384	0.384	0.354	0.390
4	0.390	0.354	0.390	0.421	0.372
5	0.390	0.384	0.390	0.378	0.384
6	0.384	0.402	0.396	0.409	0.366
7	0.372	0.360	0.378	0.396	0.390
8	0.409	0.390	0.354	0.427	0.354
9	0.372	0.402	0.409	0.390	0.390
10	0.384	0.384	0.384	0.396	0.372
11	0.415	0.433	0.427	0.427	0.421
12	0.427	0.421	0.415	0.427	0.427
13	0.415	0.402	0.409	0.402	**0.439**
14	0.421	0.415	0.421	0.427	0.433
15	0.415	0.421	0.402	0.421	0.427

efficiency, as shorter prompts require fewer iterations to search. As shown in Table 8 and Table 9, the results are consistent with our findings on HumanEval. The optimal performance is achieved when k is around half the prompt length (e.g., $k = 4$ for LLaMA-3.1-8B). This reinforces the robustness of our heuristic for setting k.

A.5.3 Analysis on TruthfulQA

For TruthfulQA, which also features short prompts, we experimented on the Qwen-3-32B model. We narrowed the range of k to 1 to 3, with $s = 5$. The results in Table 10 show the average truthfulness and informativeness scores. Again, a moderate k (in this case, $k = 2$) paired with an appropriate ω (e.g., 1.4) yields the best combined performance.

A.6 Time Cost Analysis

Our MSP framework introduces a deliberate computational trade-off to enable a more reasoned generation process. This section quantifies the time cost associated with its two main stages: the one-time Gradient-Guided Mask Search (GGMS) and the per-step Mark-Steered Decoding (MSD).

Table 6: Full Pass@1 performance on HumanEval with CodeLlama-13B (Part 1/2). The highest scores are highlighted in bold.

$k \setminus \omega$	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**
1	0.335	0.335	0.354	0.348	0.354
2	0.341	0.323	0.348	0.360	0.341
3	0.341	0.329	0.360	0.360	0.390
4	0.329	0.335	0.354	0.348	0.354
5	0.360	0.360	0.341	0.360	0.341
6	0.348	0.341	0.366	0.378	0.378
7	0.372	0.360	0.372	0.366	0.348
8	0.335	0.360	0.390	0.366	0.366
9	0.348	0.354	0.384	0.335	0.360
10	0.366	0.384	0.354	0.372	0.384
11	0.360	0.384	0.378	0.396	0.354
12	0.348	0.372	0.372	0.384	0.402
13	0.390	0.366	0.384	0.384	0.390
14	0.335	0.384	0.372	0.378	0.390
15	0.372	0.384	0.384	0.360	0.384

Table 7: Full Pass@1 performance on HumanEval with CodeLlama-13B (Part 2/2). The highest scores are highlighted in bold.

$k \setminus \omega$	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**
1	0.348	0.329	0.341	0.354	0.354
2	0.366	0.341	0.360	0.341	0.360
3	0.360	0.372	0.335	0.354	0.354
4	0.360	0.354	0.366	0.366	0.341
5	0.348	0.378	0.366	0.360	0.366
6	0.372	0.372	0.372	0.372	0.366
7	0.372	0.348	0.372	0.366	0.360
8	0.396	0.396	0.366	0.348	0.433
9	0.354	0.384	0.341	0.354	0.372
10	0.378	0.366	0.384	0.390	0.366
11	0.354	0.384	0.384	0.378	0.409
12	0.396	0.384	0.396	0.384	0.390
13	0.390	0.390	0.396	0.390	0.409
14	0.372	0.415	0.396	0.390	0.402
15	0.433	0.421	**0.445**	0.396	0.396

Table 8: Pass@1 performance on MBPP with LLaMA-3.1-8B under varying k and ω.

$k \setminus \omega$	**1.1**	**1.4**	**1.7**	**2.0**
1	45.2	45.5	45.3	45.1
2	45.8	46.1	46.0	45.7
4	46.5	**46.9**	46.7	46.2
7	44.8	45.2	45.0	44.5

Table 9: Pass@1 performance on MBPP with CodeLlama-13B under varying k and ω.

$k \setminus \omega$	**1.1**	**1.4**	**1.7**	**2.0**
1	57.8	58.1	58.3	57.9
3	58.2	58.6	58.7	58.4
5	58.5	58.8	**58.9**	58.6
7	57.7	58.0	58.2	57.6

We report the average time costs on both the LLaMA-3.1-8B and CodeLlama-13B models, each using its optimal hyperparameter configuration (i.e., $k \approx n/2, s = 5, \omega = \omega_{\text{opt}}$).

Table 10: Performance on TruthfulQA with Qwen-3-32B (Truthfulness / Informativeness).

$k \setminus \omega$	**1.1**	**1.4**	**1.7**	**2.0**
1	0.86 / 0.87	0.88 / 0.89	0.87 / 0.88	0.85 / 0.86
2	0.88 / 0.89	**0.90 / 0.91**	**0.90 / 0.91**	0.87 / 0.88
3	0.87 / 0.88	0.89 / 0.90	0.88 / 0.89	0.86 / 0.87

The results, summarized in Table 11 and Table 12, were benchmarked on a single NVIDIA H100 GPU. The GGMS time represents the initial, one-off cost to identify the salient tokens for a given prompt. The MSD time reflects the average time taken to generate a single token during the decoding phase. The 'Tokens/sec' metric is calculated based on the MSD stage to provide a direct comparison with standard generation speeds.

Table 11: Average time cost analysis of MSP on the LLaMA-3.1-8B model.

Dataset	**GGMS Time (s)**	**MSD Time per Token (s)**	**Tokens/sec**
HumanEval	18.4	54.5	59.2
MBPP	–	70.4	72.3
TruthfulQA	125.6	42.6	45.4

We conducted a similar analysis for the larger CodeLlama-13B model on the code generation benchmarks to understand how the cost scales. The findings are presented in Table 12.

Table 12: Average time cost analysis of MSP on the CodeLlama-13B model.

Dataset	**GGMS Time (s)**	**MSD Time per Token (s)**	**Tokens/sec**
HumanEval	38.8	39.1	41.2
MBPP	–	49.1	53.6

Across both models, the primary computational overhead arises in the GGMS stage, and this cost grows with model size due to the need for gradient computation. A promising direction for future work is to approximate this stage using a lightweight proxy model. By contrast, the MSD stage introduces only a modest overhead at each decoding step. This trade-off highlights MSP as a method designed for high-quality, deliberate generation, where accuracy and alignment with user intent take precedence over latency.

A.7 Details on Compute Resources

All experiments were conducted on a high-performance computing cluster with the following specifications:

- **CPU:** The system is equipped with a dual-socket configuration of Intel Xeon Platinum 8558 processors, providing a total of 96 cores and 192 threads.
- **System Memory:** The total available system RAM is 2.0 TiB.
- **GPU:** For model inference and gradient computations, we utilized a cluster of 10 NVIDIA H100 GPUs, each equipped with 80 GB of high-bandwidth memory (HBM).

This robust computational environment ensured that all experiments could be run efficiently and with sufficient resources to handle the largest models and datasets in our study.

A.8 LLM Usage

Large language models were utilized during the preparation of this manuscript. Their role was strictly limited to improving the grammatical structure, clarity, and style of the written text. The core research ideas, experimental design, results, and analyses were developed entirely by the authors.