Language System: A Lightweight Ranking Framework for Language Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Conventional research on large language models (LLMs) has primarily focused on refining output distributions, while paying less attention to the decoding process that transforms these distributions into final responses. Recent advances, such as scaling the computation of inference time with reward models, have underscored the importance of decoding, but these methods often suffer from high computational costs and limited applicability. In this paper, we revisit LLM decoding through the lens of recommender systems, conceptualizing the decoding process as analogous to the ranking stage in recommendation pipelines. From this perspective, we observe that both traditional decoding methods and reward models exhibit clear limitations such as redundancy. Motivated by this insight, we propose Language System, a novel framework that introduces a lightweight ranker to rerank candidate responses using features extracted by the base model. Experiments across a wide range of tasks show that Language System achieves performance comparable to large-scale reward models, while requiring only <0.5M additional parameters, significantly reducing the computational overhead during both training and inference stages. This highlights the efficiency and effectiveness of our method, showcasing its potential to fully unlock the capabilities of LLMs.

1 Introduction

Traditional research on enhancing the capabilities of large language models (LLMs) has primarily focused on improving the quality of output distributions through approaches such as scaling up model sizes [1], fine-tuning for specific tasks (SFT) [2, 3], and reinforcement learning with human feedback (RLHF) [4, 5]. However, the decoding process, which converts the output distributions into final responses, has not received sufficient attention. Current decoding strategies, including top-k sampling [6, 7], self-consistency [8], and contrastive decoding [9], are largely rule-based and task-specific, limiting their ability to fully exploit the potential of LLMs' powerful output distributions.

[10] found that if an oracle could select the best response from multiple samples generated by a model, the performance of a 7B model could even surpass that of a 70B model as the sample number increases. This finding highlights the tremendous potential of the decoding process in maximizing model performance. To approximate the oracle, recent studies on inference-time computing [11, 12, 13] have introduced reward models to select the best response. While these methods demonstrate strong performance across various tasks, the reliance on auxiliary reward models significantly increases computational and time overhead during both training and inference, thereby limiting their scalability and applicability in broader contexts.

To address these limitations, we rethink LLMs through the lens of recommender systems. As shown in Figure 1, **each LLM can be viewed as a special recommender system**, where the input serves as the user information, and the model's role is to recommend the most appropriate response

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

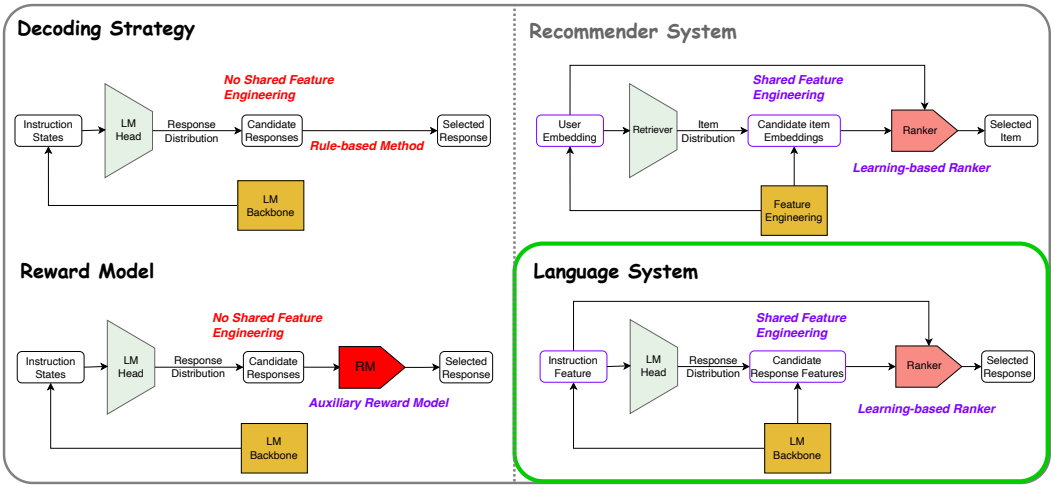


Figure 1: **The comparison among existing methods, recommender system and our Language System.** The two charts on the left highlight the limitations of existing decoding strategies and reward models, in contrast to the recommender system pipeline shown in the top-right. Language System addresses these limitations by incorporating a feature-shared, learnable, and lightweight ranker.

Table 1: Comparison of parameters and performance between the Language System and reward models, using Llama3.1-8B-Instruct as the base model.

Method	**Training Stage**		**Inference Stage**		**Performance**
	Trainable	GPU-Loaded	Sampling	Ranking	
Language System	**<0.5M**	**<0.5M**	8.2B	**<0.5M**	★★★
RM (llama8B-LoRA)	176M	8.2B	8.2B	8.4B	★★★
RM (gpt2)	137M	137M	8.2B	137M	★☆☆

as the "item" tailored to the user's needs. Therefore, the model backbone, language head, and decoding process correspond directly to the feature engineering, retriever, and ranker in a traditional recommender system [14]. When a user provides an input, the model backbone first extracts user features, represented as the hidden states of the last token. The language head then generates a coarse response distribution. Finally, a predefined decoding strategy samples several candidate responses from this distribution and selects the most suitable one among them.

In this analogy, the limitations of both existing decoding strategies and reward models become evident. As illustrated in Figure 1, current decoding strategies are typically simple and rule-based, neglecting the crucial role of reranking model responses. Meanwhile, reward models, though effective as rankers, introduce substantial computational overhead both in training and inference. From the perspective of recommender systems, these methods essentially redo the feature engineering for ranking from scratch, ignoring the features already extracted during the recall stage that could have been shared. This redundancy leads to significant unnecessary computations and inefficiency.

In this paper, we propose **Language System**, inspired by recommender systems, to address aforementioned shortcomings of existing methods. The Language System incorporates a carefully designed lightweight ranker to rerank candidate responses generated by the base model. As illustrated in Figure 3, the earlier layers of the base model can be viewed as shared feature engineering for both the retriever and ranker, similar to recommender systems. Once the candidate responses are sampled, the ranker utilizes the extracted features to rerank the candidates and identify the most appropriate response.

As shown in Table 1, by leveraging the representations of base models, our method achieves performance comparable to that of the Llama8B-based reward model, while requiring only <0.5M additional parameters, significantly reducing the computational overhead during both training and inference stages. Table 3 shows that the lightweight ranker supports both training and inference independently on CPUs, demonstrating the potential to build a personalized Language System. As illustrated in Figure 2, the base model can be paired with different rankers to enhance capabilities

across various dimensions. The base model is expected to run on high-resource central nodes, while the rankers can be deployed on edge nodes or even on users' local devices, allowing continual learning and personalized adaptation to diverse user needs.

In conclusion, the main contributions of this paper are:

- We reinterpret LLMs through the lens of recommender systems, revealing the limitations of existing decoding strategies and reward models while highlighting the potential of the decoding process.
- We propose Language System, a novel and lightweight ranking framework for LLMs that is both efficient and effective. It allows a single base model to be flexibly paired with different rankers, allowing personalized adaptation to diverse user needs.
- We conduct extensive experiments across a wide range of tasks using both 7B and 32B base models, demonstrating that our framework achieves competitive performance compared to large-scale reward models with only <0.5M additional parameters, significantly reducing the computational overhead during both training and inference.

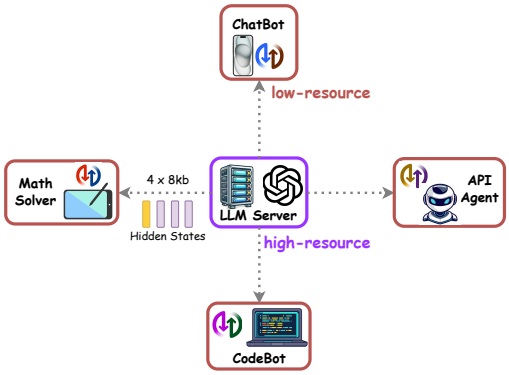


Figure 2: **Personalized Language System.** We can pair a single base model with different rankers to enable personalized adaptation for diverse user needs simultaneously. The base model runs on high-resource central nodes, while rankers can be deployed on edge devices or even local user devices.The CPU-trainability allows each user's ranker to perform continual learning with behavioral data, paving the way for deeper personalization.

2 Method

In this section, we introduce Language System, our lightweight ranking framework for LLMs inspired by recommender system. This framework addresses the limitations of existing decoding strategies and reward models by incorporating an efficient and effective ranking mechanism.

2.1 Language Systems

As shown in Figure 3, we employ a lightweight yet effective ranker to rerank candidate responses generated by language models. Specifically, a hyperparameter is defined to select a specific layer in the model, and the hidden states of this layer are used as features for the ranker. [1] Before inference, the hidden states of the selected layer corresponding to the final token of the given instruction is recorded as the instruction feature, denoted as i. The model then begins the inference process, sampling K candidate responses. Once each candidate response is fully generated, the hidden state of the chosen layer corresponding to the final token is recorded as its feature. These features, representing the candidate responses, are denoted as $\{r_k\}_{k=1}^{K}$. These instruction and response features are then fed into the ranker to identify the most suitable response.

Following methods commonly used in recommender systems [14], we design both a listwise ranker and a pointwise ranker to refine candidate responses. Both rankers first project the input features into a low-dimensional space, compressing information while significantly reducing the parameter count for subsequent processing. They then process the projected features using their respective blocks and compute the similarity between each response and the instruction features, which is used to rerank the responses.

[1] The final layer of the model backbone is often suboptimal for feature extraction; instead, layers located around 60% from the bottom of the model typically yield better representations, as shown in Subsection 4.

Specifically, the listwise ranker processes all candidates simultaneously, enabling direct comparisons between them:

$$\left[\tilde{i}, \tilde{r_1}, \tilde{r_2}, \cdots, \tilde{r_K}\right] = Trans\left(Proj\left([i, r_1, \cdots, r_K]\right)\right), \tag{1}$$

$$[s_1, s_2, \cdots, s_K] = Sim\left(\tilde{i}, [\tilde{r_1}, \tilde{r_2}, \cdots, \tilde{r_K}]\right). \tag{2}$$

As illustrated in Figure 3, after projection, the instruction feature i and the response features $r_k{}_{k=1}^{K}$ interact within a Transformer block. Subsequently, similarity scores between the instruction and each candidate response are computed, and the candidate with the highest score is selected as the final output.

The pointwise ranker, in contrast, evaluates each candidate response individually based on the given instruction feature:

$$\left[\tilde{i}, \tilde{r_k}\right] = \left[MLP\left(Proj(i)\right), MLP\left(Proj(r_k)\right)\right], \tag{3}$$

$$s_k = Sim\left(\tilde{i}, \tilde{r_k}\right). \tag{4}$$

Each projected feature is independently processed using a shared MLP block. The ranker then computes a similarity score between the instruction feature and each response feature. The response with the highest similarity score is selected as the final result.

Notably, the choice of similarity function for both the listwise and pointwise rankers depends on the form of the response labels. If the labels are binary (0 or 1), cosine similarity is applied, framing the task as a classification problem. In contrast, if each response is assigned a specific score, a learnable similarity function is employed to fit these scores:

$$s_k = Sim\left(\tilde{i}, \tilde{r_k}\right), \tag{5}$$

$$= W * concat\left(\tilde{i}, \tilde{r_k}\right). \tag{6}$$

The Transformer block in the listwise ranker and the MLP block in the pointwise ranker can both be extended to multiple blocks. For efficiency, we use a single block in all main experiments, and the impact of the block number is analyzed in Subsection 4.

2.2 Dataset construction and Ranker Training

The training dataset for our ranker is constructed in a manner similar to that of reward model datasets [15], introducing virtually no additional computational or time overhead. For each task, we allow the base model to perform sampling and generate 100 responses for each instruction in the training set. During this process, the corresponding instruction features and response features are recorded. These features are then used to train the ranker effectively. After collecting all responses, we assign labels depending on the characteristics of the task. These details will be discussed within the context of specific tasks in Section 3.

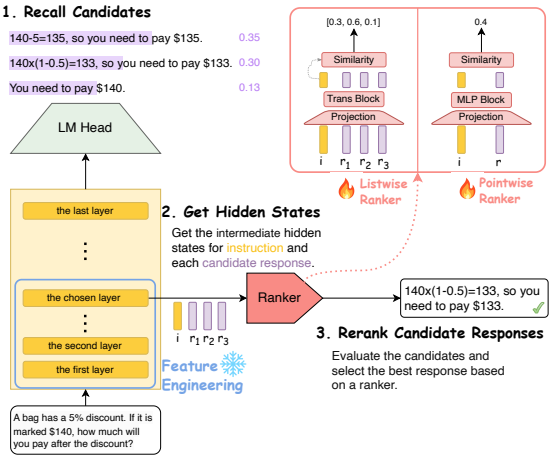


Figure 3: **The framework of Language System.** The base model generates multiple candidate responses, and then the hidden states of both the instruction and each candidate response are extracted from a predefined layer as features. Finally, the ranker selects the most suitable response based on these features.

The listwise ranker processes a list of candidates simultaneously, allowing for direct comparisons among them. To prepare the training data, K candidate responses are randomly sampled for each query from the previously constructed dataset. This process is repeated multiple times, and groups that do not contain both positive and negative responses are filtered out. Ultimately, N data groups per query, along with their corresponding hidden

states, are collected for training, formally represented as $\left[i, (r_1^{(n)}, y_1^{(n)}), \cdots, (r_K^{(n)}, y_K^{(n)})\right]_{n=1}^{N}$. For training process, the loss function is selected based on the form of the labels. If $y_k^{(n)} \in \{0, 1\}$, the task is framed as a classification problem, where $s_k^{(n)}$ is computed using cosine similarity. We optimize the ranker using the following KL divergence loss:

$$\pi_y^{(n)} = \frac{y_k^{(n)}}{\sum_{k=1}^{K} y_k^{(n)}}, \quad \pi_s^{(n)} = \frac{\exp(s_k^{(n)})}{\sum_{k=1}^{K} \exp(s_k^{(n)})}, \tag{7}$$

$$\mathcal{J}_{cls}^{list} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{D}_{KL} \left(\pi_y^{(n)} \| \pi_s^{(n)} \right). \tag{8}$$

If $y_k^{(n)} \in \mathbb{R}$, the task is treated as a regression problem, where $s_k^{(n)}$ is computed by the learnable similarity function previously introduced. We apply the mean squared error (MSE) loss:

$$\mathcal{J}_{reg}^{list} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{K} \sum_{k=1}^{K} \left(s_k^{(n)} - y_k^{(n)} \right)^2. \tag{9}$$

The pointwise ranker is much simpler than the listwise ranker. For each query, it independently pairs each candidate response with its corresponding instruction, formally represented as: $\left[i, (r^{(n)}, y^{(n)})\right]_{n=1}^{N}$. The choice of loss function also depends on the form of the labels. Following the discussion for the listwise ranker, we summarize the corresponding loss functions for the two forms of labels below:

$$p_k^{(n)} = \frac{\exp(s^{(n)})}{1 + \exp(s^{(n)})}, \tag{10}$$

$$\mathcal{J}_{cls}^{point} = -\frac{1}{N} \sum_{n=1}^{N} y^{(n)} \log p^{(n)} + (1 - y^{(n)}) \log(1 - p^{(n)}). \tag{11}$$

$$\mathcal{J}_{reg}^{point} = \frac{1}{N} \sum_{n=1}^{N} \left(s^{(n)} - y^{(n)} \right)^2. \tag{12}$$

3 Main Experiments

In this section, we conduct experiments on three widely studied tasks for LLMs: math, coding, and function calling. To further demonstrate the generality of our approach, we also evaluate its general instruction-following ability, as detailed in Appendix B. Full hyperparameters are listed in Appendix A

Baselines For each task, we train two reward models of different scales for comparison. The first is based on GPT-2 [16], a relatively small model that nevertheless has over 100 times more parameters than our ranker. The second reward model is trained from the corresponding base model using LoRA. Although the number of trainable parameters is similar to GPT-2, the full model must be loaded into GPU memory for both training and inference, resulting in substantially greater computational overhead. In addition, we use the first sampled response from the base model as a simple baseline and adopt deterministic beam search as a representative decoding strategy.

Ranker Settings In all experiments, the rankers are implemented using either a single Transformer block or a single MLP block, and they operate on features extracted from approximately the bottom 60% of the base model's layers. During both training and evaluation, each data group consists of 10 candidate responses. The ranker is trained to classify each response as correct or incorrect, formulating the task as a binary classification problem. Cosine similarity is used to compute the final logits, and the training objective is defined by the classification loss $\mathcal{J}_{cls}$, as specified in Equations 8 or 11.

Models We evaluate our method on Llama3.1-8B-Instruct [17], Qwen2.5-7B-Instruct, and Qwen2.5-32B-Instruct [18] to demonstrate its generality across different model architectures and

Table 2: The total performance across the three tasks compares our methods with reward models and common decoding strategies. The RM means reward model. In the Parameter column, we report the number of trainable parameters for each method. For reward models trained with LoRA, we additionally report the number of GPU-loaded parameters.

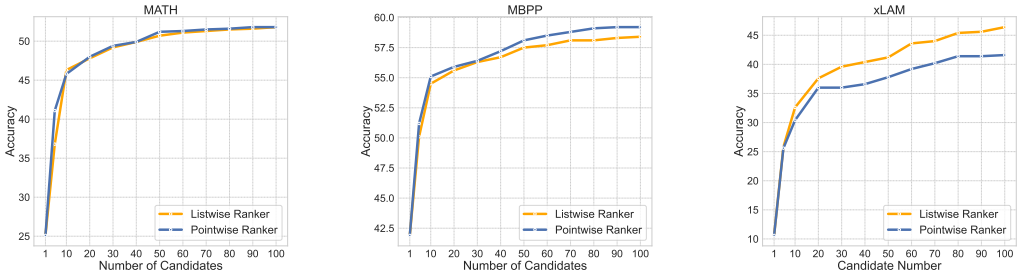
Method	Parameter	MATH	MBPP	xLAM
Llama3.1-8B-Instruct				
ListRanker (ours)	**0.30M**	**46.3**	54.5	32.6
PointRanker (ours)	**0.28M**	45.8	**55.1**	30.4
RM (gpt2)	137M	42.9	47.7	29.4
RM (Llama8B)	176M / 8.2B	45.1	52.9	**32.8**
Beam Search	—	40.3	42.3	27.0
First Sample	—	25.1	41.9	10.6
Qwen2.5-7B-Instruct				
ListRanker (ours)	**0.27M**	74.8	**63.2**	**71.0**
PointRanker (ours)	**0.25M**	**75.2**	62.7	70.4
RM (gpt2)	137M	71.9	60.2	65.4
RM (Qwen7B)	161M / 7.6B	74.6	62.9	70.2
Beam Search	—	67.9	62.2	68.0
First Sample	—	68.7	60.6	57.0
Qwen2.5-32B-Instruct				
ListRanker (ours)	**0.36M**	81.1	74.2	72.8
PointRanker (ours)	**0.34M**	**81.3**	74.6	72.4
RM (gpt2)	137M	78.8	70.6	68.8
RM (Qwen32B)	537M / 32.8B	80.7	**75.9**	**73.6**
Beam Search	—	78.1	71.4	70.6
First Sample	—	75.9	68.2	65.2

scales. To ensure fairness, all models are evaluated using zero-shot prompts, as detailed in Appendix D.

Datasets For the mathematics task, we use the MATH dataset [19], which contains 12,500 competition-level problems spanning seven topics and five difficulty levels. To ensure coverage while maintaining efficiency, we uniformly sample 1,000 problems each for training and testing across different topics and difficulty levels. For the coding task, we use the complete MBPP dataset [20], which consists of short Python programming problems, 374 for training and 500 for testing, each paired with test cases to evaluate the correctness of the generated solutions. For the function calling task, we adopt the xlam-function-calling-60k dataset [21], which comprises 60,000 high-quality function calling problems and answers. We randomly sample 1,500 more challenging problems with more than three APIs, and split them into 1,000 training and 500 testing examples.

Metrics For the mathematics and function calling tasks, we extract the final answer from the model's response and verify its correctness by comparing it with the ground-truth answer. For the coding task, we extract the generated code and evaluate its correctness using the test cases provided in the MBPP dataset.

Results Both the listwise and pointwise rankers significantly improve model performance across all tasks. Our lightweight method consistently outperforms the reward model (gpt2), despite being over 100 times smaller in scale, and even achieves performance comparable to reward models trained from the base model. Specifically, for Llama3.1-8B-Instruct, our approach improves over the first-sample baseline by more than 20% on MATH and 12% on MBPP, substantially outperforming both larger reward models. On the function calling task, it trails the Llama8B-based reward model by only 0.2%. For Qwen2.5-7B-Instruct, the Language System outperforms all baselines. For Qwen2.5-32B-Instruct, rankers with fewer than 1M parameters achieve performance comparable to 32B-scale reward models, demonstrating remarkable potential. This suggests that rankers can adapt to even larger base models, as the extracted features they rely on become increasingly expressive—allowing them to "stand on the shoulders of giants."



(a) Mathematics Task (b) Coding Task (c) Function-Calling Task

Figure 4: The performance of the Language System built on Llama3.1 improves consistently across all three tasks as the number of candidate responses increases.

Table 3: The total training time on the MBPP dataset for both CPU and GPU settings, including data loading stages.

Method	CPU	A100
Listwise Ranker	67s	44s
Pointwise Ranker	71s	42s
RM (gpt2)	>1h	72s
RM (Llama8b)	too long	24min

Table 4: Comparison of performance and parameter on MATH under different ranker architecture ablations, using Llama3.1-8B as the base model.

Ranker Setting	Accuracy	Parameter
Listwise Ranker	46.3	0.30M
– remove projection	46.4	192M
– remove instruction	44.2	0.30M
Pointwise Ranker	45.8	0.28M
– remove projection	46.0	128M
– remove MLP block	42.5	0.25M
– remove instruction	44.1	0.28M

4 Analysis and Ablation Study

Ranker Scaling Law Figure 4 illustrates the relationship between the performance of the Language System and the number of candidate responses provided to the ranker. We found that performance improves across diverse tasks as the number of candidates increases, demonstrating the Ranker Scaling Law. A key open problem in current research is how to effectively scale inference-time computation in large language models to enhance their performance. Most existing work has focused on optimizing inference configurations during the sampling stage, often relying on traditional reward models for response reranking [12, 11, 22]. In contrast, our approach focuses on the ranking stage, providing an efficient, effective, and scalable alternative. This distinction underscores the complementarity between our method and sampling-based techniques, suggesting that they can be integrated to further improve model performance.

CPU Trainability As shown in Table 3, the lightweight rankers can be efficiently run on CPUs, demonstrating the potential of constructing a personalized Language System. As illustrated in Table 2, the base model can be paired with different rankers to enhance capabilities across various dimensions. The hidden states of the final token (8KB) are compact enough to be transmitted over the internet. In this setup, the base model runs on high-resource central nodes, while rankers can be deployed on edge devices or even local user devices, enabling flexible adaptation to diverse user needs. Moreover, the CPU-trainability allows each user's ranker to perform continual learning with behavioral data, paving the way for deeper personalization.

Ablation Study To better understand the design of our ranker, we conduct ablation studies on all key components of both the listwise and pointwise architectures. As shown in Table 4, the projection layer compresses high-dimensional features into a lower-dimensional space, playing a critical role in keeping the ranker lightweight. Removing this layer results in a much larger ranker with minimal performance gain. Additionally, we examine the role of the instruction feature, which is used to compute similarity scores with each candidate for ranking. Replacing this feature with a learnable vector leads to a noticeable drop in performance, underscoring the its importance as a form of user information, consistent with our perspective of recommender system

Table 5: Performance comparison across different ranker configurations in MATH for Llama3.1-8B-Instruct.

Ranker Type	Hidden States Layer				Block Number			
	0.1	0.3	0.6	1.0	1	2	3	4
Llama3.1-8B-Instruct								
Listwise Ranker	41.2	44.6	**46.3**	44.9	46.3	46.7	46.6	**46.9**
Pointwise Ranker	40.6	43.6	**45.8**	44.0	45.8	46.2	**46.4**	46.3
Qwen2.5-7B-Instruct								
Listwise Ranker	70.6	72.7	**74.8**	73.6	74.8	74.9	75.2	**75.4**
Pointwise Ranker	71.4	73.1	**75.2**	73.9	75.2	75.1	**75.6**	75.5

Table 6: Performance across different hyperparameter configurations for Llama3.1-8B-Instruct on the MATH dataset. Green indicates the best resuls, while red indicates the worst results.

Optimizer	SGD				AdamW	
Learning Rate	0.05	0.1	0.5	1.0	1e-5	1e-4
Batch Size=256	46.2	46.1	45.8	45.7	46.2	46.1
Batch Size=1024	46.1	45.9	46.3	45.9	45.9	46.1

(a) Listwise Ranker

Optimizer	AdamW			
Learning Rate	5e-5	1e-4	2e-4	5e-4
Batch Size=64	41.2	42.2	45.1	43.6
Batch Size=256	43.2	41.9	42.8	44.7

(b) Reward Model (Llama8B)

Ranker Configurations The last layer of the model backbone is often not the best choice for providing features. Since the backbone is trained for next-token prediction, the final layers tend to overfit to this specific task. In contrast, intermediate layers typically provide more comprehensive representations of the preceding context, making them better suited for capturing the overall features required for ranking [23]. As shown in the Hidden States Layer part of Table 5, the most effective features for the rankers are extracted from the 60% from the bottom of the model layers. As shown in the Block Number part of Table 5, increasing the ranker's scale has only a marginal impact. Since the base model has already extracted high-quality features, the ranker's task remains relatively simple, making further scaling unnecessary.

hyperparameter robustness Table A illustrates the hyperparameter robustness of our method, particularly in comparison to reward models. For the listwise ranker, the accuracy range across 12 hyperparameter configurations is only 0.6%, whereas the reward model exhibits a much larger range of 3.9% across 8 configurations.

Transferability To assess this, we use the MATH dataset, which includes seven distinct problem types. We train the ranker on a single task type and evaluate its generalization to the remaining tasks (see Table 7). Results show that rankers trained on any individual task maintain robust performance across all others. Remarkably, in some cases, the cross-task performance approaches that of task-specific rankers, highlighting the system's adaptability to unseen domains.

5 Related Work

Decoding methods A variety of rule-based decoding methods have been proposed to improve language model performance, including top-k sampling [6, 7], temperature-based sampling [24], and nucleus sampling [25]. Beyond these, more refined algorithms have been developed to focus specific task. [8, 26] introduce self-consistency as a method to improve Chain-of-Thought (CoT) reasoning by majority voting. [27, 9, 28] select or even finetune another auxiliary model to assist in generating responses that better align with specific requirements. However, these methods are either rule-based or task-specific, limiting their performance ceiling and application scope. We propose a more general ranking framework to address these limitations.

Reward Models Reward models have been widely adopted for the enhancements of LLMs.They serve as learned proxies for human preferences in RLHF [29, 30], and have also been applied to guide multi-step reasoning processes [31, 32]. While effective in a range of scenarios, reward models typically introduce significant computational overhead, limiting their practical deployment in real-world systems. To address this issue, some efforts aim to teach models to act as self-critics [33, 34], but their performance remains suboptimal. [35] proposes an embedding-based alternative to simplify

Table 7: The Transfer Performance of a Ranker Trained on a Single Task. All results are tested with Llama3.1-8B-Instruct on the MATH dataset. For task types, we use abbreviations in the table due to space constraints: Prealgebra (PA), Algebra (A), Number Theory (NT), Counting and Probability (CP), Geometry (G), Intermediate Algebra (IA), Precalculus (PC).

Source Task	**Target Task**						
	PA	A	NT	CP	G	IA	PC
PA	**67.5**	61.3	38.2	43.7	33.4	21.9	32.2
	0.0	-0.4	-0.5	-0.2	0.0	-0.8	-1.7
A	66.0	**61.7**	38.5	42.0	34.7	22.3	31.1
	-1.5	**0.0**	-0.2	-1.9	-4.0	-0.4	-2.8
NT	64.9	60.2	**38.7**	41.4	35.7	20.7	31.0
	-2.6	-1.5	**0.0**	-2.5	-2.7	-2.0	-2.9
CP	66.5	61.3	37.4	**43.9**	35.3	22.2	32.6
	-1.0	-0.4	-1.3	**0.0**	-2.1	-0.5	-1.3
G	66.0	60.5	36.7	41.4	**37.4**	22.4	31.1
	-1.5	-1.2	-1.8	-2.5	**0.0**	-0.3	-2.8
IA	64.3	58.8	35.7	38.6	32.4	**22.7**	31.3
	-3.2	-2.9	-2.8	-5.3	-5.0	**0.0**	-2.6
PC	63.0	59.1	35.6	41.1	34.5	22.3	**33.9**
	-4.5	-2.6	-2.9	-2.9	-2.9	-0.4	**0.0**

reward model training, while it focuses on RLHF settings and still requires an additional forward pass during both training and inference to extract embeddings. [36] propose a reward model that scores all candidate tokens at once, reducing call frequency but relying more on large-scale models.

Inference-time computing Recently, there has been growing interest in scaling inference-time computation to improve the performance of LLMs. Most existing approaches focus on optimizing configurations at the sampling stage, often relying on traditional reward models to evaluate and rerank generated responses [13, 37, 22, 15]. In contrast, our method shifts the focus to the ranking stage and introduces a lightweight architecture that operates directly on features already extracted by the base model. This design eliminates the need for additional forward passes, providing a scalable, efficient, and effective alternative. We believe our approach complements sampling-based strategies and can be combined with them to further improve model performance.

6 Conclusion and Discussion

In this paper, we have introduced the Language System, a novel lightweight ranking framework for enhancing the LLMs. By rethinking LLMs through the lens of recommender systems, we identified the limitations of existing decoding strategies and reward models, particularly their limitations on rule-based methods and computationally expensive reranking. Our approach integrates a lightweight ranker that leverages features extracted by the base model, enabling more efficient, effective and scalable reranking with minimal computational overhead.

Moreover, the separability of the ranker from the base model further enhances the flexibility of our approach. This decoupling allows for the independent optimization of the ranker, enabling a base model to be paired with multiple rankers that enhance different aspects of its capabilities, making it adaptable to various domains. We hope that our approach can offer new perspectives for the future of language model inference and contribute to the development of more resource-efficient AI systems.

References

[1] Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[2] Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*, 2024.

[3] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In *Advances in Neural Information Processing Systems*, 2023.

[4] Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, 2023.

[5] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

[6] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018.

[7] Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018.

[8] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

[9] Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023.

[10] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[11] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[12] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2024.

[13] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

[14] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 2020.

[15] Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. Reft: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

[16] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[17] 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 preprint arXiv:2407.21783*, 2024.

[18] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[19] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems*, 2021.

[20] 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.

[21] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*, 2024.

[22] Kexun Zhang, Shang Zhou, Danqing Wang, William Yang Wang, and Lei Li. Scaling llm inference efficiently with optimized sample compute allocation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7959–7973, 2025.

[23] Oscar Skean, Md Rifat Arefin, Yann LeCun, and Ravid Shwartz-Ziv. Does representation matter? exploring intermediate layers in large language models. *arXiv preprint arXiv:2412.09563*, 2024.

[24] Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. In *Proceedings of the Workshop on Stylistic Variation*, 2017.

[25] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

[26] Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting. *arXiv preprint arXiv:2402.10200*, 2024.

[27] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Jinyuan Jia, Bill Yuchen Lin, and Radha Poovendran. SafeDecoding: Defending against jailbreak attacks via safety-aware decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

[28] Yue Zhang, Leyang Cui, Wei Bi, and Shuming Shi. Alleviating hallucinations of large language models through induced hallucinations. *arXiv preprint arXiv:2312.15710*, 2024.

[29] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[30] Yifan Zhang, Ge Zhang, Yue Wu, Kangping Xu, and Quanquan Gu. General preference modeling with preference representations for aligning language models. *arXiv preprint arXiv:2410.02197*, 2024.

[31] Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

[32] Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

[33] Yuancheng Xu, Udari Madhushani Sehwag, Alec Koppel, Sicheng Zhu, Bang An, Furong Huang, and Sumitra Ganesh. Genarm: Reward guided generation with autoregressive reward model for test-time alignment. *arXiv preprint arXiv:2410.08193*, 2024.

[34] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

[35] Hao Sun, Yunyi Shen, Jean-Francois Ton, and Mihaela van der Schaar. Reusing embeddings: Reproducible reward model research in large language model alignment without gpus. *arXiv preprint arXiv:2502.04357*, 2025.

[36] Ahmad Rashid, Ruotian Wu, Rongqi Fan, Hongliang Li, Agustinus Kristiadi, and Pascal Poupart. Towards cost-effective reward guided text generation. *arXiv preprint arXiv:2502.04517*, 2025.

[37] Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. Interpretable preferences via multi-objective reward modeling and mixture-of-experts, 2024.

[38] Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023.

[39] Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2024.

[40] Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

A Hyperparameter settings

In sampling process, we set temperature as 1.5 for diverse responses and max_new_tokens as 1024 to make sure completed answers. We sample 100 responses for each problem. During training, we perform a grid search over the parameter ranges specified in Table 8.

Table 8: The hyperparameter list

Hyperparameter	Value
Sampling	
Sampling Temperature	1.5
Sampling Max New Tokens	1024
Ranker Training	
Batch Size	[256, 1024]
Epoch	1
Optimizer	[SGD, AdamW]
SGD LR	[0.05, 0.1, 0.5, 1.0]
SGD Momentum	[0.0, 0.9]
AdamW LR	[1e-5, 1e-4]
AdamW Betas	(0.9, 0.999)
Weight Decay	1e-4
LR Schedule	[Constant, Cosine Decay]
Projection Dimension	64
Reward Model Training	
Batch Size	[64, 256]
Epoch	1
Optimizer	AdamW
AdamW LR	[5e-5, 5e-4]
AdamW Betas	(0.9, 0.999)
Weight Decay	1e-4
LR Schedule	[Constant, Cosine, Decay]
LoRA r	64
LoRA alpha	[64, 128]

B Instruction-Following Task

Our framework performs well on the three tasks presented in Section 3. To further demonstrate its general applicability, we also evaluate it on a mixed instruction-following task.

Models Considering that instruct models are specifically fine-tuned for instruction-following tasks, we conduct evaluations on this task with Llama3.1-8B-Base [17] and Qwen2.5-7B-Base [18]. For fairness, all models are evaluated using zero-shot prompts, as shown in Appendix D.

Datasets We use the first 1,000 queries from the Databricks-Dolly-15k dataset [38] for training. For evaluation, we adopt AlpacaEval [39], a widely recognized benchmark for assessing instruction-following capabilities in LLMs. It consists of diverse test queries sourced from Self-Instruct, OASST, Anthropic's Helpful dataset, Vicuna, and Koala.

Metrics Unlike tasks such as mathematics, instruction-following lacks objective ground-truth answers, making rule-based evaluation infeasible. To address this, we follow the AlpacaFarm [39] method and prompt DeepSeek-V3 to simulate human judgment by assigning scores from 0 to 5 to all sampled responses. These responses are then used to train both our ranker and reward models. For evaluation, we adopt the official AlpacaEval evaluator to compute the Length-Controlled Win Rate metric [40], a relative measure based on a reference model. For each base model, we use its corresponding instruct variant as the reference—for example, Llama3.1-8B-Instruct for Llama3.1-8B-Base.

Table 9: The evaluation on general instruction-following tasks compares our method against reward models and common decoding strategies. We conduct experiments on Llama3.1-8B-Base and Qwen2.5-7B-Base, reporting the win rate using the corresponding Instruct model as the reference. RM (base) refers to reward models trained from the respective base model.

Method	**Parameter**	**Llama3.1-8B-Base**	**Qwen2.5-7B-Base**
ListRanker (ours)	<0.3M	<u>30.7</u>	**46.3**
PointRanker (ours)	<0.3M	27.1	<u>45.8</u>
RM (gpt2)	137M	27.1	42.9
RM (base)	~170M/8B	**31.6**	45.3
First Sample	—	19.0	25.1
Beam Search	—	20.4	40.3

Results As shown in Table 2, the performance of our methods far exceeds that of vanilla decoding strategies and is comparable to the reward models trained from base models. Notably, with the assistance of a 0.3M-level ranker, Qwen2.5-7B-Base achieves a 46.3% win rate compared to Qwen2.5-7B-Instruct, which has undergone extensive fine-tuning on various instruction-following tasks.

C Limitations

The ranker is trained using hidden states saved during data sampling, which introduces additional I/O overhead. For example, when training on the MBPP dataset with 374 queries, data loading took 31 seconds using 8 parallel threads. This accounts for the comparable CPU and GPU training times observed in Table 3. When scaling to significantly larger datasets, this I/O overhead may become a limiting factor and should be taken into consideration.

D Prompts for Each Task

Prompt for mathematics task

system:
You are a math expert.

user:
Please solve the given math problem step by step and present the answer in the following format: "\boxed{X}", where X is the answer.
{Question}

assistant:

Prompt for coding task

system:
You are an expert Python programmer.

user:
Write a Python function based on the following instructions and test example. Please ensure that the function is clearly marked with a start and end so I can easily extract it from your output.

Instructions:
{question}

Test Example:
{test_list[0]}

Please provide your code with clear start and end markers, like so:

```
#START OF CODE
def {function_name(input)}:
    ... function code ...
    return result
#END OF CODE
```

assistant:

Prompt for function calling task

system:
You are a function-calling assistant. Your role is to complete tasks solely through correct function calls, without generating any additional text. For each task, directly output the function call(s) required to complete it. If the task involves multiple steps, you may issue multiple function calls sequentially. Each function call must be formatted as a JSON object. For example: [{"name": "functionA", "arguments": {"param1": "value1", "param2": "value2"}}, {"name": "functionB", "arguments":{"param1": "value1", "param2": "value2"}}]
The following are the available functions: {function_list}

Now, use the appropriate function(s) to complete the given task.

user:
{Question}
Please directly output the function call(s) to solve the task without any other text.

assistant:

Prompt for instruction-following task

system:
You are an assistant.

user:
Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.
{Instruction} Begin!

assistant:

Scoring criteria in instruction-following task

Review the user's question and the corresponding response using the additive 5-point scoring system described below

The user's question is between <question>and </question>The response of the AI Assistant is between <response>and </response>

Points are accumulated based on the satisfaction of each criterion: - Add 1 point if the response is relevant and provides some information related to the user's inquiry, even if it is incomplete or contains some irrelevant content. - Add another point if the response addresses a substantial portion of the user's question, but does not completely resolve the query or provide a direct answer. - Award a third point if the response answers the basic elements of the user's question in a useful way, regardless of whether it seems to have been written by an AI Assistant or if it has elements typically found in blogs or search results. - Grant a fourth point if the response is clearly written from an AI Assistant's perspective, addressing the user's question directly and comprehensively, and is well-organized and helpful, even if there is slight room for improvement in clarity, conciseness or focus. - Bestow a fifth point for a response that is impeccably tailored to the user's question by an AI Assistant, without extraneous information, reflecting expert knowledge, and demonstrating a high-quality, engaging, and insightful answer. - If the response repeats itself or is not concise and to the point, score the response 0.

<question>prompt</question>
<response>response</response>

After examining the user's instruction and the response: - output the score of the evaluation using this exact format: "score: <total points>", where <total points>is between 0 and 5 - Briefly justify your total score, up to 100 words.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: See Abstract and Introduction, where enumerates the contributions.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: see Appendix

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: No theory

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have provided sufficient details and instructions for the repreduction of the main experimental results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We promise to release the code after publication. We have provided sufficient details and instructions for the repreduction of the main experimental results.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: see Experiments and Appendix

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We tune the learning rate for most our experiments. Due to limited computing resources, we are not able to run the experiments many times to derive the error bars.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: see Appendix

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This paper only include experiments on public open datasets.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: No societal impact

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper poses no such risks

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: see experiment setup

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: We introduce new assets. However, we do not release them at submission time. We commit to releasing the codebase upon publication.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: the paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: the paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: We only use LLM for writing and formatting purposes.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.