Language System: A Lightweight Ranking Framework for Language Models

Chenheng Zhang [*1] Tianqi Du [*1] Jizhe Zhang [1] Mingqing Xiao [1] Yifei Wang [2] Yisen Wang [1 3]
Zhouchen Lin [1 3]

Abstract

Conventional research on large language models (LLMs) has primarily focused on refining output distributions, with less attention to the decoding process that transforms these distributions into final responses. Although recent work on inference-time scaling with reward models highlights the importance of decoding, such methods often incur 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, both traditional decoding methods and reward models show clear limitations, including redundancy. To address this, we propose Language System, a lightweight framework that reranks candidate responses using features extracted by the base model. Experiments across diverse tasks demonstrate that Language System achieves performance comparable to large-scale reward models with <0.5M additional parameters, significantly reducing overhead during both training and inference. This highlights the efficiency and effectiveness of our approach in unlocking LLM capabilities.

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 (Kojima et al., 2022) and fine-tuning for specific tasks (SFT) (Gao et al., 2024; Malladi et al., 2023). However, the decoding process, which converts the output distributions into final responses, has not received sufficient attention. Brown et al. (2024) showed that if an oracle selects the best response from multiple samples, a 7B model can outperform a 70B model as sample count increases. This underscores the untapped potential of decoding in maximizing performance. To approximate such an oracle, recent work on inference-time computation (Snell et al., 2024; Wu et al., 2024; Setlur et al., 2024) introduces reward models for reranking. However, these methods incur substantial overhead during both training and inference, limiting their scalability and real-world applicability.

To address these limitations, we reinterpret 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 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 (Zhang et al., 2020). Given an input, the backbone extracts user features (i.e., hidden states of the final token), the language head generates a coarse response distribution, and a decoding strategy samples candidate responses and selects one for output.

In this analogy, the limitations of both existing decoding strategies and reward models become evident. As illustrated in Figure 1, existing decoding strategies are often simple and rule-based, neglecting the importance of reranking. While reward models serve as effective rankers, they introduce significant overhead during both training and inference. From the perspective of recommender systems, they 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 incor-

[*]Equal contribution [1]State Key Lab of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University, China [2]MIT CSAIL, USA [3]Institute for Artificial Intelligence, Peking University, China. Correspondence to: Zhouchen Lin <zlin@pku.edu.cn>.

Proceedings of the 3^{rd} Efficient Systems for Foundation Models Workshop at the International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

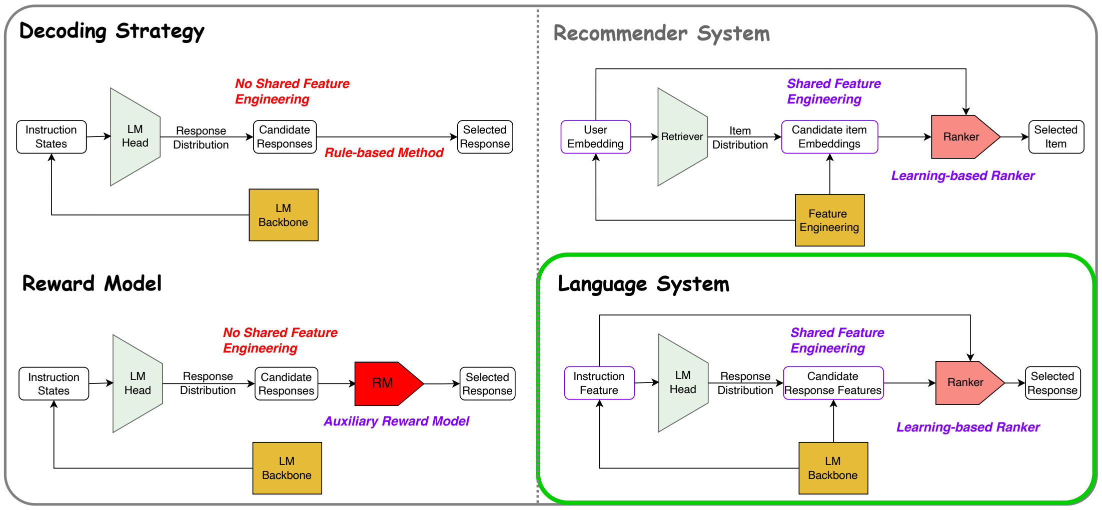


Figure 1: **The comparison among existing methods, recommender system and our Language System.** The two charts on the left highlight the limitations of existing decoding strategies and reward models, in contrast to the recommender system pipeline shown in the top-right. Language System addresses these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

porates 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 large-scale reward model, while requiring only <0.5M additional parameters, significantly reducing the computational overhead during both training and inference stages. Table 2 further shows that the ranker can be trained and run efficiently on CPUs, enabling deployment in personalized settings. As illustrated in Figure 2, the base model runs on central servers, while rankers can be deployed on edge or local devices to support continual learning and user-specific adaptation.

In conclusion, the main contributions of this paper are: (1) 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. (2) 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. (3) We conduct extensive experiments on both 7B and 32B models, showing that our method matches large-scale reward models while using <0.5M additional parameters and significantly reducing computational cost.

2. Method

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 common practices in recommender systems (Zhang et al., 2020), we design both a listwise and a pointwise ranker. Both first project the input features into a low-dimensional space to compress information and reduce parameter cost. They then process the projected features using their respective blocks and compute the similarity (cosine similarity by default) between each response and the instruction features, which is used to rerank the responses.

Specifically, the listwise ranker processes all candidates

[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 Section 4 and Appendix A.

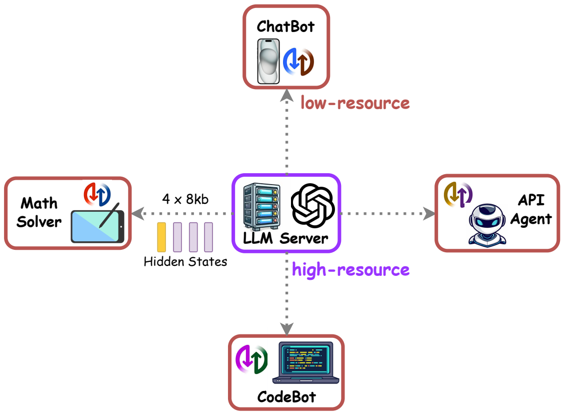


Figure 2: **Personalized Language System.** We can pair a single base model with different rankers to enable personalized adaptation for diverse user needs simultaneously. The base model runs on high-resource central nodes, while rankers can be deployed on edge devices or even local user devices.The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

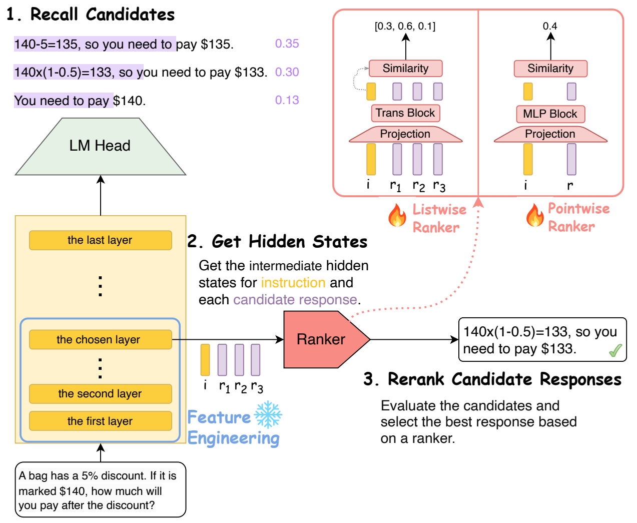

1. Recall Candidates
140-5=135, so you need to pay $135. 0.35
140x(1-0.5)=133, so you need to pay $133. 0.30
You need to pay $140. 0.13
LM Head
the last layer
the chosen layer
the second layer
the first layer
A bag has a 5% discount. If it is marked $140, how much will you pay after the discount?
2. Get Hidden States
Get the intermediate hidden states for instruction and each candidate response.
Feature Engineering
Ranker
140x(1-0.5)=133, so you need to pay $133.
3. Rerank Candidate Responses
Evaluate the candidates and select the best response based on a ranker.
[0.3, 0.6, 0.1]
Similarity
Trans Block
Projection
Listwise Ranker
0.4
Similarity
MLP Block
Projection
Pointwise Ranker

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.

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 and selects the final result.

A detailed description of dataset construction and ranker training is provided in Appendix D, and a fine-grained analysis of each component of the Language System is presented in Section 4 and Appendix A.

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 C.

Baselines For each task, we train two reward models of different scales for comparison. The first is based on GPT-2 (Radford et al., 2019). The second reward model is trained from the corresponding base model using LoRA. 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.

Datasets For the mathematics task, we uniformly sample 1,000 problems each for training and testing across different topics and difficulty levels in MATH dataset (Hendrycks et al., 2021). For the coding task, we use the full MBPP dataset (Austin et al., 2021), which consists of Python programming problems, 374 for training and 500 for testing. For the function calling task, we adopt the xlam-function-calling-60k dataset (Liu et al., 2024). We randomly sample 1,500 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,

Table 1: 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

we extract the final answers 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."

4. Analysis and Ablation Study

Ranker Scaling Law Figure 4 shows that the Language System's performance improves as the number of candidate responses increases, demonstrating the Ranker Scaling Law. A key challenge in LLM research is scaling inference-time computation to boost performance. Most prior work focuses on optimizing sampling strategies, often with traditional reward models for reranking (Wu et al., 2024; Snell et al., 2024; Zhang et al., 2025). In contrast, our method targets the ranking stage, offering a scalable and efficient alternative. This highlights the complementarity of our approach with sampling-based techniques, which can be combined for further gains.

CPU Trainability As shown in Table 2, 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.

Further analysis are provided in Appendix A.

5. Conclusion and Discussion

We introduce the Language System, a lightweight ranking framework for enhancing LLMs inspired by recommender systems. Our method addresses the limitations of existing methods by reranking responses based on features extracted from the base model, with minimal overhead. The ranker can be decoupled from the base model, allowing flexible pairing and independent optimization across different domains. We hope this work offers new perspectives on inference-time computation and contributes to more resource-efficient LLM systems.

References

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Conover, M., Hayes, M., Mathur, A., Xie, J., Wan, J., Shah, S., Ghodsi, A., Wendell, P., Zaharia, M., and Xin, R. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023.

Cui, G., Yuan, L., Wang, Z., Wang, H., Li, W., He, B., Fan, Y., Yu, T., Xu, Q., Chen, W., et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dubois, Y., Galambosi, B., Liang, P., and Hashimoto, T. B. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024a.

Dubois, Y., Li, C. X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P. S., and Hashimoto, T. B. Alpacafarm: A simulation framework for methods that learn from human feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2024b.

Fan, A., Lewis, M., and Dauphin, Y. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018.

Ficler, J. and Goldberg, Y. Controlling linguistic style aspects in neural language generation. In *Proceedings of the Workshop on Stylistic Variation*, 2017.

Gao, Z., Wang, Q., Chen, A., Liu, Z., Wu, B., Chen, L., and Li, J. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*, 2024.

Guan, X., Zhang, L. L., Liu, Y., Shang, N., Sun, Y., Zhu, Y., Yang, F., and Yang, M. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems*, 2021.

Holtzman, A., Buys, J., Forbes, M., Bosselut, A., Golub, D., and Choi, Y. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Li, X. L., Holtzman, A., Fried, D., Liang, P., Eisner, J., Hashimoto, T., Zettlemoyer, L., and Lewis, M. 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.

Liu, Z., Hoang, T., Zhang, J., Zhu, M., Lan, T., Kokane, S., Tan, J., Yao, W., Liu, Z., Feng, Y., et al. API-Gen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*, 2024.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. In *Advances in Neural Information Processing Systems*, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Rashid, A., Wu, R., Fan, R., Li, H., Kristiadi, A., and Poupart, P. Towards cost-effective reward guided text generation. *arXiv preprint arXiv:2502.04517*, 2025.

Setlur, A., Nagpal, C., Fisch, A., Geng, X., Eisenstein, J., Agarwal, R., Agarwal, A., Berant, J., and Kumar, A. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

Skean, O., Arefin, M. R., LeCun, Y., and Shwartz-Ziv, R. Does representation matter? exploring intermediate layers in large language models. *arXiv preprint arXiv:2412.09563*, 2024.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Sun, H., Shen, Y., Ton, J.-F., and van der Schaar, M. Reusing embeddings: Reproducible reward model research in large language model alignment without gpus. *arXiv preprint arXiv:2502.04357*, 2025.

Trung, L., Zhang, X., Jie, Z., Sun, P., Jin, X., and Li, H. Reft: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Wang, H., Xiong, W., Xie, T., Zhao, H., and Zhang, T. Interpretable preferences via multi-objective reward modeling and mixture-of-experts, 2024.

Wang, X. and Zhou, D. Chain-of-thought reasoning without prompting. *arXiv preprint arXiv:2402.10200*, 2024.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Wu, Y., Sun, Z., Li, S., Welleck, S., and Yang, Y. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2024.

Xu, Y., Sehwag, U. M., Koppel, A., Zhu, S., An, B., Huang, F., and Ganesh, S. Genarm: Reward guided generation with autoregressive reward model for test-time alignment. *arXiv preprint arXiv:2410.08193*, 2024a.

Xu, Z., Jiang, F., Niu, L., Jia, J., Lin, B. Y., and Poovendran, R. 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)*, 2024b.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

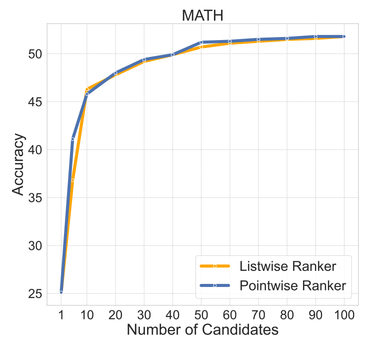
Zhang, K., Zhou, S., Wang, D., Wang, W. Y., and Li, L. Scaling llm inference efficiently with optimized sample compute allocation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7959–7973, 2025.

Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., and Agarwal, R. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024a.

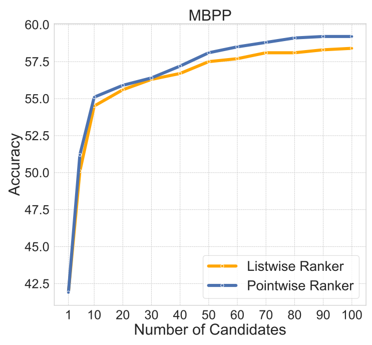
Zhang, S., Yao, L., Sun, A., and Tay, Y. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 2020.

Zhang, Y., Cui, L., Bi, W., and Shi, S. Alleviating hallucinations of large language models through induced hallucinations. *arXiv preprint arXiv:2312.15710*, 2024b.

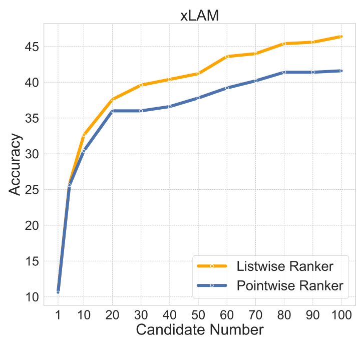
Zhang, Y., Zhang, G., Wu, Y., Xu, K., and Gu, Q. General preference modeling with preference representations for aligning language models. *arXiv preprint arXiv:2410.02197*, 2024c.



(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 2: 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 3: 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 instruction	44.1	0.28M
– remove MLP block	42.5	0.25M

A. Detailed Analysis

Ranker Scaling Law Figure 4 shows that the Language System's performance improves as the number of candidate responses increases, demonstrating the Ranker Scaling Law. A key challenge in LLM research is scaling inference-time computation to boost performance. Most prior work focuses on optimizing sampling strategies, often with traditional reward models for reranking (Wu et al., 2024; Snell et al., 2024; Zhang et al., 2025). In contrast, our method targets the ranking stage, offering a scalable and efficient alternative. This highlights the complementarity of our approach with sampling-based techniques, which can be combined for further gains.

CPU Trainability As shown in Table 2, 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 3, 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

Table 4: 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 5: 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)

perspective of recommender system.

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 (Skean et al., 2024). As shown in the Hidden States Layer part of Table 4, 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 4, 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 5 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 6). 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.

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 (Dubey et al., 2024) and Qwen2.5-7B-Base (Yang et al., 2024). For fairness, all models are evaluated using zero-shot prompts, as shown in Appendix G.

Datasets We use the first 1,000 queries from the Databricks-Dolly-15k dataset (Conover et al., 2023) for training. For evaluation, we adopt AlpacaEval (Dubois et al., 2024b), 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 (Dubois et al., 2024b) 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

Table 6: 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**

to train both our ranker and reward models. For evaluation, we adopt the official AlpacaEval evaluator to compute the Length-Controlled Win Rate metric (Dubois et al., 2024a), 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.

Results As shown in Table 1, 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. 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.

D. Dataset construction and Ranker Training

The training dataset for our ranker is constructed in a manner similar to that of reward model datasets (Trung et al., 2024), 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.

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

Table 7: 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

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:

$$\pi_y^{(n)} = \frac{y_k^{(n)}}{\sum\limits_{k=1}^{K} y_k^{(n)}}, \quad \pi_s^{(n)} = \frac{\exp(s_k^{(n)})}{\sum\limits_{k=1}^{K} \exp(s_k^{(n)})}, \tag{5}$$

$$\mathcal{J}_{cls}^{list} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{D}_{KL}\left(\pi_y^{(n)} \| \pi_s^{(n)}\right). \tag{6}$$

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:

$$\mathcal{J}_{reg}^{list} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{K} \sum_{k=1}^{K} \left(s_k^{(n)} - y_k^{(n)}\right)^2. \tag{7}$$

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:

$$p_k^{(n)} = \frac{\exp(s^{(n)})}{1 + \exp(s^{(n)})}, \tag{8}$$

$$\mathcal{J}_{cls}^{point} = -\frac{1}{N} \sum_{n=1}^{N} y^{(n)} \log p^{(n)} + (1 - y^{(n)}) \log(1 - p^{(n)}). \tag{9}$$

$$\mathcal{J}_{reg}^{point} = \frac{1}{N} \sum_{n=1}^{N} \left(s^{(n)} - y^{(n)}\right)^2. \tag{10}$$

E. Related Work

Decoding methods A variety of rule-based decoding methods have been proposed to improve language model performance, including top-k sampling (Fan et al., 2018; Holtzman et al., 2018), temperature-based sampling (Ficler & Goldberg, 2017), and nucleus sampling (Holtzman et al., 2020). Beyond these, more refined algorithms have been developed to focus specific task. Wang et al. (2023); Wang & Zhou (2024) introduce self-consistency as a method to improve Chain-of-Thought (CoT) reasoning by majority voting. (Xu et al., 2024b; Li et al., 2023; Zhang et al., 2024b) select or even finetune another

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]

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 (Ouyang et al., 2022; Zhang et al., 2024c), and have also been applied to guide multi-step reasoning processes (Guan et al., 2025; Cui et al., 2025). 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 Xu et al. (2024a); Zhang et al. (2024a), but their performance remains suboptimal. Sun et al. (2025) proposes an embedding-based alternative to simplify reward model training, while it focuses on RLHF settings and still requires an additional forward pass during both training and inference to extract embeddings. Rashid et al. (2025) 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 (Setlur et al., 2024; Wang et al., 2024; Zhang et al., 2025; Trung et al., 2024). 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.

F. 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 2. When scaling to significantly larger datasets, this I/O overhead may become a limiting factor and should be taken into consideration.

G. 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.