ToolPRM: Fine-Grained Inference Scaling of Structured Outputs for Function Calling

Jianghao Lin[1]*, Yuanyuan Shi[1]*, Xin Peng[2], Renjie Ding[2], Hairui Wang[2], Yuxuan Peng[2], Bizhe Bai[3], Weixi Song[3], Fengshuo Bai[1], Huacan Chai[1], Weinan Zhang[1,3]†, Fei Huang[2]†, Ying Wen[1,3]†

[1]Shanghai Jiao Tong University, China;
[2]Longshine AI Research, China;
[3]Shanghai Innovation Institute, China
{chiangel, wnzhang, ying.wen}@sjtu.edu.cn, huangfei@longshine.com

Abstract

Large language models (LLMs) excel at function calling, but inference scaling has been explored mainly for unstructured generation. We propose an inference-scaling framework for structured outputs that combines fine-grained beam search with **ToolPRM**, a process reward model scoring each intra-call decision (function name and argument filling). We build the first fine-grained intra-call supervision dataset via function masking, rollout collection, and step-level annotation. ToolPRM outperforms outcome and coarse-grained reward models in predictive accuracy and yields consistent test-time gains on multiple function-calling benchmarks. We further show that structured generation follows "**explore more but retain less**", since early JSON errors are unrecoverable.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in a diverse range of tasks (Zhou et al., 2026; Xi et al., 2025). Subsequently, inference scaling has emerged as a critical technique for further enhancing their performance, where additional computational resources are allocated during the inference phase to explore a wider range of possibilities or to iteratively refine outputs, and to ultimately maximize the quality of the output. The core idea behind inference scaling strategies is to move beyond generating a single greedily decoded sequence of tokens. Instead, inference scaling approaches often intentionally increase the computational effort at inference time, to search for a wider range of possibilities and multiple potential reasoning paths, and select the best trajectories for better performance (Ke et al., 2025).

Existing research on inference scaling predominantly focuses on **unstructured output generation tasks**, such as mathematical problems (Puri et al., 2025) and other intricate reasoning tasks (Ma et al., 2023), where each sentence or discrete unit serves as a processing step. Novel methods often combine Tree of thoughts (ToT) (Yao et al., 2023) with tree-based search algorithms, generalizing the linear Chain-of-Thought (Wei et al., 2022) (CoT) by collecting multiple potential next thoughts from a given state. To search for the optimal output while condensing the possibility space, beam search and Monte Carlo Tree Search (MCTS) become the ideal choices, and several studies have adapted them for LLM inference scaling (Wu et al., 2024; Zhang et al., 2023; Liu et al., 2023; Choi et al., 2023; Zhou et al., 2023). Correspondingly, they involve either self-evaluation (Xie et al., 2023) or process reward models (Ma et al., 2023; Hung et al., 2025) (PRMs) to generate an evaluation score for each sub-sequence, as the reference for searching. However, the application of inference scaling techniques within the domain of **structured outputs**, particularly in the context of function calling[1], remains significantly underexplored.

Current studies that incorporate process reward mechanisms and inference scaling for function calling tasks operate at a coarse-grained level (Wang et al., 2024a; Nath et al., 2025), treating an entire round of function calling response as a singular, monolithic step. For example, Wang et al. (Wang et al., 2024a) employ a Best-of-N approach, where multiple generated function call candidates are scored and ranked by an outcome reward model (ORM). The final answer is selected by assessing the entire generation as a singular entity without applying distinct fine-grained rewards to its subcomponents within each function call. This overlooks the inherent multi-stage nature of the function calling process that involves solving a sequence of sub-questions, including the selection of function

*Equal contribution.
†Corresponding authors.

[1]In this paper, the term "function" is interchangeable with "tool" and "API".

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 18792–18804

name, the identification of relevant parameters, and the decision of parameter values. Consequently, the potential for optimization of fine-grained process rewards at the intra-call level is not adequately addressed by existing methodologies.

To this end, in this paper, we propose a novel fine-grained process reward mechanism specifically designed for structured function calling tasks (dubbed **ToolPRM**). In contrast to prior methods that treat function calls as indivisible units, our approach decomposes each call into semantically interpretable intermediate steps. We develop a dedicated process reward model, ToolPRM, supervised using meticulously curated fine-grained intra-call step labels constructed from both xlam-function-calling-60k (Liu et al., 2024b) and xlam-irrelevance-7.5k (Lin et al., 2024) datasets. Consequently, we develop fine-grained beam search with the intra-call process supervision from ToolPRM, which proves to outperform existing function-calling baselines in our experiments.

More critically, we highlight an important insight for the inference scaling on structured and unstructured output generation. In traditional inference scaling settings like math reasoning, the unstructured outputs usually enable more flexible inference scaling by simply maintaining a larger number of candidates. Even if we maintain the wrong intermediate steps, it could be further corrected at later steps via reflection and process supervision. However, in structured output generation, retaining multiple candidate trajectories often degrades performance, as errors in early steps cannot be corrected later, thereby wasting the subsequent computational budget. Therefore, we propose a tailored inference scaling principle for structured outputs: devote computation to exploring a wider range of decisions (i.e., increasing the beam width), while aggressively pruning the number of retained candidates (i.e., reducing the number of active beams). We summarize such a principle of inference scaling for structured function calling outputs as "**explore more, but retain less**".

The contributions of this paper are as follows:

- **Fine-Grained Intra-Call Process Supervision Dataset**: We construct a novel annotated dataset specifically designed for fine-grained intra-call reward modeling of structured function calling. This dataset, along with its annotation scripts, is given in the anonymous link and will be made publicly available, facilitating future research and benchmarking in fine-grained process supervision for structured output generation.
- **ToolPRM**: We introduce ToolPRM, a fine-grained process reward modeling framework specifically tailored for inference scaling in structured function calling tasks. ToolPRM could assist different backbone models in conducting test-time scaling for superior performance.
- **Inference Scaling Principle for Structured Outputs**: We identify and formalize a critical principle for inference scaling in structured output tasks: *Explore more but retain less*. It highlights the importance of widening exploration while aggressively eliminating incorrect, unrecoverable steps, leading to better performance.

2 Related Works

LLM for Function Calling. Recent research has increasingly demonstrated the significant potential of enabling large language models (LLMs) to interact with external systems through function calling (Liu et al., 2024b; Park et al., 2025; Srinivasan et al., 2023; Chai et al., 2025; Yang et al., 2025). For instance, IBM's Granite-20B-FunctionCalling (Abdelaziz et al., 2024) enhances performance through multi-task learning on seven core function-calling subtasks. ToolACE (Liu et al., 2024a) designs a self-evolution synthesis process for high-quality complex function calling data generation. To further enhance the LLM response to complex function calling, in this paper, we disassemble function calling into fine-grained basic steps, employ a process reward model to better guide the generation steps inside function calling, and finally implement inference scaling mechanisms with intra-call-level granularity.

Inference Scaling Strategies. A variety of inference scaling strategies have been developed based on existing sampling and searching algorithms. Straightforward methods include self-consistency (Wang et al., 2022) strategy, which samples N independent candidate reasoning paths through temperature sampling (Ackley et al., 1985) and decides the ultimate answer with majority voting. Best-of-N (BoN) (Brown et al., 2024) evaluates the N candidates with an outcome reward model to select the best one. In novel studies, tree-based methods have become the mainstream, and effective algorithms based on beam search or Monte Carlo Tree Search (MCTS) emerge (Wu

et al., 2024; Zhang et al., 2023; Liu et al., 2023; Choi et al., 2023; Zhou et al., 2023). While substantial studies have verified the effectiveness of inference scaling strategies on unstructured output generation tasks, especially on mathematical reasoning tasks, the application on structured output generation tasks remains underexplored. In this paper, we apply inference scaling on structured function call generation, analyze its promotion on final performance, and discuss the balance between exploration and retention in searching process.

Process Reward Models (PRMs). In complex multi-step reasoning tasks where errors can occur at any point, granular feedback is particularly valuable (Zhou et al., 2025; Zheng et al., 2025b). In contrast to the outcome reward models (ORMs) that just evaluate the final outcome, PRMs are trained to evaluate the correctness or quality of intermediate steps within an inference process (Uesato et al., 2022; Zheng et al., 2025a; Zhu et al., 2025). PRMs are initially applied in RLHF (Ouyang et al., 2022) training process to provide reward signals and to supervise the generation of reasoning steps (Piotrowski et al., 2025; Lightman et al., 2023). In this work, we curate the first fine-grained intra-call process supervision dataset with automatic annotation of granular process rewards, and finetune the ToolPRM that excels at verifying the intermediate steps in forming a full function call. Combined with well-configured searching strategies, our proposed ToolPRM helps the function-calling LLM reach the state-of-the-art performance.

3 Methodology

This section introduces the fine-grained process reward model for function calling (*i.e.*, ToolPRM). We first present fine-grained decomposition for each turn of function callings, and present the data collection and annotation for reward modeling training. Then, we design a state transition machenism for function calling, and apply beam search with fine-grained process supervision with ToolPRM.

3.1 Fine-Grained Decomposition of Function Calls

The essence of ToolPRM lies in its fine-grained decomposition of the function calling process. Traditional process supervision approaches for tool invocations primarily rely on coarse-grained response-level rewards, which evaluate the function call as a monolithic unit. However, such coarse granularity limits the interpretability, debuggability, and optimization potential.

To this end, ToolPRM decomposes each function call into fine-grained but semantically meaningful steps. Each LLM-generated response comprises a sequence of function calls tailored to a given user query. The construction of each function call is further broken down into (a) selecting the appropriate function name and (b) iteratively identifying parameter names and assigning corresponding values. Rather than evaluating a function call as a whole, ToolPRM provides fine-grained supervision that assesses the correctness of each constituent decision. As shown in Figure 1, this decomposition serves as the foundation for data collection and annotation.

3.2 Data Collection and Annotation

To enable robust reward modeling under our fine-grained process supervision paradigm, we construct a high-quality annotated dataset that reflects the intricacies of individual decision points.

Data Collection. We begin by collecting natural language queries paired with their corresponding structured function calls, using xlam-function-calling-60k (Liu et al., 2024b) and xlam-irrelevance-7.5k (Lin et al., 2024) datasets as the foundations. As shown in Figure 1, we apply function masking that selectively replaces function names and parameter identifiers with random strings. The masking mechanism introduces function ambiguity, encouraging the model to rely on contextual understanding of descriptions instead of simple memorization of the tool names, thus enhancing model robustness and generalization (Lin et al., 2024). We adopt Hammer2.1-3b and Hammer2.1-7b as policy models to perform the rollout for data collection. Given a user query and a set of masked function candidates, each rollout generates a sequence of function calls, which is subsequently annotated with fine-grained step labels.

Result Annotation. As shown in Figure 1, the structured function call generated by the policy model is usually in JSON format. We can provide fine-grained step labels with the following types:

- `<FUNC_NAME>`: Whether the selected function name at this round is correct or not.
- `<ARG_VALUE>`: Whether the one pair of parameter name and value is correctly filled in, which

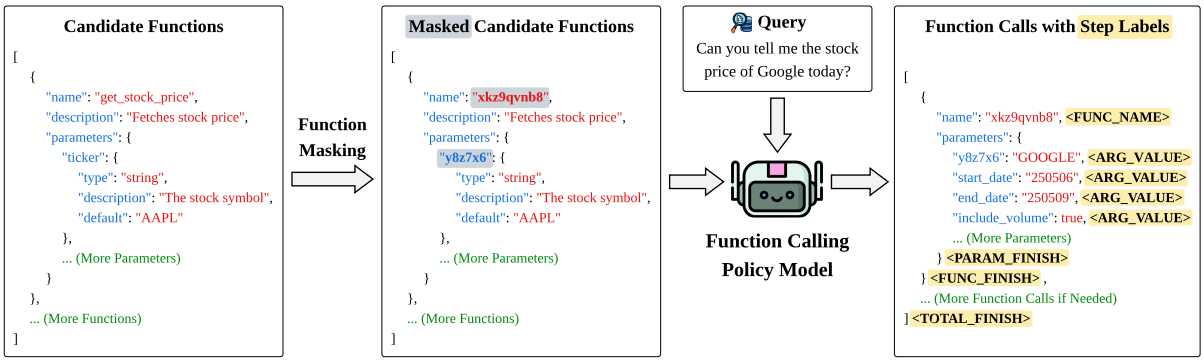

Figure 1: The illustration of data collection for ToolPRM.

can repeat multiple times.

- <PARAM_FINISH>: Whether all the parameters and values are correctly assigned.
- <FUNC_FINISH>: Whether the single function call (one element of the list) is correct or not.
- <TOTAL_FINISH>: Whether the overall response (a list of function calls) is correct or not.

Each symbol is followed by a binary label annotated by exact match with any of the possible ground truths, to indicate the correctness of the corresponding fine-grained step. Some of the step labels seem redundant (*e.g.*, <ARG_VALUE> and <PARAM_FINISH>), but our experiments show that such a hierarchical step label redundancy can help the reward model generalize and perform better.

3.3 State Transition Mechanism and ToolPRM Training

Based on the fine-grained step labels above, we can further formalize the function calling generation process as a dynamic decision process with a series of state transitions. As illustrated in Figure 2, we define the following five fine-grained states:

- **State #0 (Initial State)**: Input context and masked function candidates are presented.
- **State #1 (To Select Function Name)**: The model is required to select the function name from the candidates that are aligned with the query intent.
- **State #2 (To Select Parameter Name)**: The model should choose one necessary parameter to be filled in for the selected function.
- **State #3 (To Fill in Parameter Value)**: The model has to assign the proper value to the previously selected function parameter.
- **State #4 (Terminated State)**: One LLM response for function calls completes.

Each transition between states can be explicitly supervised by the fine-grained step labels discussed in Section "Data Collection and Annotation for Reward Modeling".

To train ToolPRM under fine-grained process supervision, we represent the function calling generation as a trajectory of decision steps, each labeled with a binary process reward.

Let $\mathcal{T} = \{(s_t, a_t, r_t)\}_{t=1}^{T}$ denote a trajectory, where s_t is the state at step t, a_t is the action (*e.g.*, selecting a function or filling a parameter via language token generation), and $r_t \in \{0, 1\}$ indicates its correctness. Each state encodes the current decision context, including the input query, masked function candidates, as well as the partially generated function calls.

In this paper, since the backbone of ToolPRM is also a large language model, we adopt the tokens "+" and "-" for positive and negative reward labels, respectively. Given N annotated trajectories $\{\mathcal{T}_i\}_{i=1}^{N}$, ToolPRM is trained to predict r_t for each (s_t, a_t) via generative process reward modeling:

$$\mathcal{L}_{\text{ToolPRM}} = -\mathbb{E}_{\tau \in \mathcal{D}, (s_t, a_t, r_t) \in \tau} \log p_\theta(r_t | s_t, a_t), \quad (1)$$

where $r_t^{(i)} \in \{+, -\}$ is the binary label token, and θ is the parameters of backbone LLM.

3.4 Beam Search with Fine-Grained Process Supervision

As shown in Figure 2, we apply beam search guided by ToolPRM to generate high-quality structured outputs. At each step, ToolPRM assigns a fine-grained process reward to candidate actions based on their local state-action pair, allowing us to prune incorrect partial trajectories early. Specifically, we compute the score s for each beam candidate with

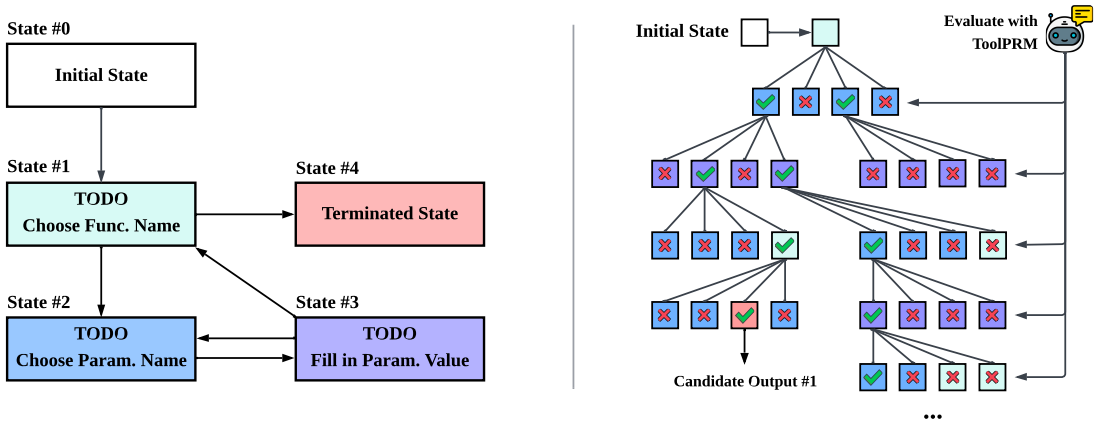

Figure 2: The state transition of function calling (left) and beam search with ToolPRM (right).

ToolPRM *i.e.*, $s = e^{s+}/(e^{s+} + e^{s-})$, where s_+ and s_- are the predictive logits on label tokens $\{+, -\}$. We maintain the top-N highest-scoring candidates, where N indicates the number of beams. At each step, every preserved candidate can generate M subsequent steps for ToolPRM evaluation, where M denotes the beam width.

We propose a core principle for inference scaling in structured output generation: **explore more but retain less**. Concretely, we increase beam width M to explore a wider range of candidate trajectories, but retain only a small number N of highly promising ones. This design reflects the unrecoverability of structured outputs, such as function calls in JSON format, where a single incorrect decision (e.g., a wrong function name or argument value) can invalidate the entire trajectory.

This stands in contrast to inference scaling in unstructured tasks such as math reasoning (Snell et al., 2024) or free-form text generation (Setlur et al., 2024; Zhang et al., 2025), where early errors can often be corrected or compensated for in later steps. In such settings, preserving a diverse set of candidates throughout decoding (i.e., larger N) is beneficial. However, in structured generation, most decision points admit only a single or very few valid actions. Retaining incorrect candidates leads to inefficient use of the generation budget, as future steps cannot recover from early structural mistakes.

Therefore, expanding the search space (larger beam width M) while aggressively pruning based on ToolPRM's step-wise supervision (smaller number of beams N) ensures that computational resources are concentrated on valid, high-quality structured outputs. This principle underpins the effectiveness of ToolPRM in scaling inference for structured generation tasks.

Table 1: The statistics of our ToolPRM dataset.

Sample Granularity (Data Split)	Positive	Negative	Total
Step (Train)	4,380,323	731,665	5,111,988
Trajectory (Train)	466,786	127,648	594,434
Step (Test)	488,611	81,366	569,977
Trajectory (Test)	52,030	14,019	66,049

4 Experiments

4.1 Experiment Setups

Datasets. To evaluate our proposed ToolPRM for function calling, we process and annotate the xlam-function-calling-60k (Liu et al., 2024b) and xlam-irrelevance-7.5k (Lin et al., 2024) datasets using a well designed pipeline, as detailed in Section "Methodology". We report the dataset statistics in Table 1. The resulting annotated dataset exhibits an average of 6.25 step labels per sample, culminating in a total of 192,061 samples. The integration of function masking techniques further expands the dataset size to 4 to 5 times, bringing it to a scale that surpasses typical datasets for Math PRM development, such as OpenAI's publicly available prm800k dataset (Lightman et al., 2023).

Note that our constructed dataset above is used to train and validate the predictive accuracy of reward models. To further validate the effectiveness of our inference scaling strategy using ToolPRM, we adopt two function-calling benchmarks: BFCL (Berkeley Function Calling Leaderboard) (Yan et al., 2024) and ToolAlpaca (Tang et al., 2023). Following previous works (Abdelaziz et al., 2024; Lin et al., 2024), we use abstract-syntax-tree-based (AST-based) accuracy as the metric for BFCL, and

employ F1 scores of both API selection and parameter value assignment for ToolAlpaca.

Baselines. We compare our proposed ToolPRM to different sets of baselines from two aspects.

To evaluate the predictive accuracy improvement brought by fine-grained intra-call process supervision, we train two distinct baseline models: outcome reward model (ORM) and coarse-grained process reward model (C-PRM). Compared with our ToolPRM, *i.e.*, fine-grained process reward model using all the step labels, these two baselines are trained on the same dataset with different choices of step labels:

- **ORM** is trained to discriminate the final result of the entire function-calling sequence with <TOTAL_FINISH>. All the other intermediate step labels are disregarded.
- **C-PRM** offers a more granular assessment of function call correctness than ORM by considering all the step labels except <ARG_VALUE>.

To evaluate the function-calling capabilities, we adopt the following three types of baselines:

- **General Purpose Models.** This group comprises large language models that have not undergone specific finetuning for function calling tasks, nor do they employ the inference scaling strategies. Their performance serves as a baseline representing general capabilities. The models include GPT-4o, GPT-4o-mini (Hurst et al., 2024), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Mistral-Nemo-Instruct (Jiang et al., 2023), and the Qwen2.5-Instruct series (72B, 32B, 7B, 3B, 1.5B) (Yang et al., 2024).
- **Function Calling Models**. This category includes models that have been specifically designed or finetuned for function calling tasks. These models include GRANITE (GRANITE-20B-FUNCTIONCALLING) (Abdelaziz et al., 2024), xLAM-fc-r series (7B, 1B) (Liu et al., 2024b), OpenFunctions-v2 (Patil et al., 2024), and Hammer2.1 series (7B, 3B, 1.5B, 0.5B) (Lin et al., 2024).
- **Inference Scaling Strategies.** We evaluate the performance enhancements afforded by different inference scaling methods. We choose the Hammer2.1 series (7B, 3B, and 1.5B variants) as base policy models. Three distinct inference scaling strategies are applied and compared against our ToolPRM: token-level beam search, majority voting, and best of N.

Table 2: Predicting accuracy of RMs of different granularities

Reward Model	Loss	Step Acc	Trajectory Acc
ORM	0.0536	98.39%	98.39%
C-PRM	0.0371	98.87%	99.06%
ToolPRM	**0.0286**	**99.11%**	**99.38%**

Implementation Details. All the experiments are conducted on NVIDIA 8xH100 GPU Clusters. We use Hammer2.1-3b as the reward model backbone and adopt SFT to train ToolPRM for 5 epochs with Adam optimizer. The batch size is 1024. The learning rate is 1e-3 with a warmup ratio of 0.008 followed by linear learning rate decay. The weight decay is 1e-5. As for the fine-grained beam search with ToolPRM, we set the temperature as 0.8, and select the number of beams N and the beam width M from $\{1, 2, 4, 8, 16\}$.

4.2 Predictive Accuracy of Reward Model

High predictive accuracy of the reward model is vital for increasing the probability of selecting and retaining correct function call sequences during inference scaling, ultimately leading to superior overall inference outcomes. Hence, this section investigates the performance of reward models for function calls with varying granularities.

We use three metrics for this evaluation: model loss, step-level accuracy (Step Acc), and trajectory-level accuracy (Trajectory Acc). Step Acc quantifies the correctness at each discrete process supervision step. It is worth noting that since each reward model (ORM, C-PRM, and ToolPRM) is supervised at its distinct level of granularity, the inherent difficulty of accurately predicting each step differs. Hence, we introduce trajectory-level accuracy to facilitate an apple-to-apple comparison of models trained with differing supervisory step labels. The trajectory accuracy evaluates the correctness of the entire function call sequence. A trajectory is deemed accurate if the final assessment of the entire sequence is correct. This means that for C-PRM and ToolPRM, even if their evaluations of some intermediate steps are incorrect, the trajectory can still be classified accurately if their overall judgment of the function call's success or failure is correct.

We report the predictive accuracy results in Ta-

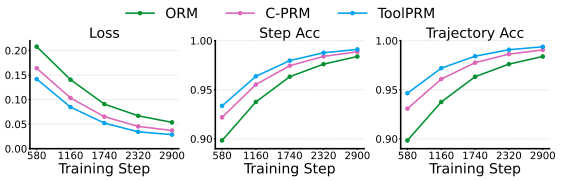

Figure 3: The 5-epoch learning curves of different RMs (*i.e.*, ORM, C-PRM, ToolPRM) in terms of loss, step-level accuracy, and trajectory-level accuracy.

ble 2 and illustrate the learning curves over the five-epoch training in Figure 4. We can observe that reward models with finer granularity consistently achieve higher predictive accuracy across all three metrics. Specifically, our proposed ToolPRM outperforms both C-PRM and ORM in this regard. This suggests that fine-grained process reward modeling, such as ToolPRM, offers superior performance not only in terms of the precision of step-level supervision but also in their ultimate effectiveness in judging overall sequence correctness compared to their coarser-grained counterparts.

4.3 Inference Scaling Performance

We conduct a comprehensive examination and comparative analysis of our ToolPRM, as well as various function calling baseline models and inference scaling strategies.

As evidenced in Table 3, we can observe that our proposed ToolPRM can generally achieve the best performance compared with other inference scaling strategies given base policy models of different sizes. Besides, the performance of other inference scaling strategies is fairly unstable and can sometimes perform worse than the base model. The potential reason is that the non-greedy sampling might cause minor errors during the function calling generation, which could directly ruin the entire trajectory. This suggests the necessity of intra-call process supervision through the fine-grained beam search for structured function calling generation.

Notably, the performance uplift brought by our ToolPRM is more pronounced for smaller policy models. This characteristic renders ToolPRM particularly advantageous for on-device inference scenarios, which is of considerable importance for applications such as function calling that are frequently deployed in edge environments and are critical for real-world utility.

Illustratively, the Hammer2.1-1.5B model, when augmented with our ToolPRM method, achieves performance comparable to the baseline 3B model. Similarly, applying ToolPRM to the Hammer2.1-3B model elevates its performance to a level on par with the baseline 7B model. Furthermore, the integration of ToolPRM with the Hammer2.1-7B model enables it to outperform several significantly larger models that previously held a performance advantage, including state-of-the-art models such as Qwen2.5-32B-Instruct.

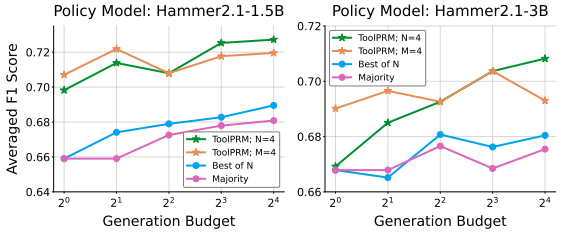

Figure 4: The F1 performance on ToolAlpaca w.r.t. different generation budgets and inference scaling strategies. We conduct experiments on Hammer2.1-1.5B (left) and Hammer2.1-3B (right) policy models.

These empirical results validate the effectiveness of our proposed inference scaling framework for structured function calling generation. Applying fine-grained beam search with ToolPRM can substantially improve the capability of base models at inference time through computational scaling.

4.4 In-Depth Analysis

In this section, we investigate the performance increase brought by the scaling of generation budget. We choose Hammer2.1-1.5B and Hammer2.1-3B as the base policy models, and conduct experiments on ToolAlpaca dataset. We report the averaged F1 score over both function selection and parameter value assignment in Figure 4.

For majority voting and best of N approaches, the generation budget refers to the number of candidate trajectories to be sampled. For our ToolPRM, as discussed in Section "Beam Search with Fine-Grained Process Supervision via ToolPRM", there are two key hyperparameters that can determine the generation budget, *i.e.*, the number of beams N and the beam width M. Hence, we develop two ToolPRM variants by fixing the one hyperparameter as 4 and increasing the other for a larger budget.

We first compare ToolPRM variants with $N = 4$ and $M = 4$. When fixing the number of candidates to be retrained at 4 and scaling the beam width, it generally demonstrates a consistent improvement in accuracy as the generation budget increases. Conversely, when maintaining a fixed beam width of 4 and scaling the number of candidates, less performance gain is exhibited, and in

Table 3: Performance comparison of the general purpose models, function calling models, and different inference scaling techniques applied on Hammer2.1 series of models, evaluated on BFCL and ToolAlpaca. Multi., Paral., Mul.P. represents the Multiple split, Parallel split, and Multiple Parallel split of BFCL, separately. And Avg. represents an unweighted average of all the sub-categories of that benchmark. The best results of each model type (*i.e.*, general purpose, function calling, and inference scaling) is given in bold and the second best is underlined.

Type	Model	Size	BFCL					ToolAlpaca		
			Simple	Multi.	Paral.	Mul.P.	Avg.	F1-API	F1-Args	Avg.
General Purpose	GPT-4o	-	77.17	95.00	**93.50**	85.00	87.67	**88.64**	**66.67**	**77.66**
	GPT-4o-mini	-	80.08	90.50	89.50	87.00	86.77	64.34	54.69	59.52
	Llama-3.1-8B-Instruct	8B	72.83	93.50	87.00	83.50	84.21	75.64	55.12	65.38
	Mistral-Nemo-Instruct	12B	77.00	93.50	89.50	84.50	86.13	87.31	66.18	76.75
	Qwen2.5-72B-Instruct	72B	**80.25**	**97.50**	**93.50**	**92.00**	**90.81**	83.21	65.56	74.39
	Qwen2.5-32B-Instruct	32B	72.83	94.00	**93.50**	88.50	87.21	85.82	61.11	73.47
	Qwen2.5-7B-Instruct	7B	75.33	94.50	91.50	84.50	86.46	83.70	60.00	71.85
	Qwen2.5-3B-Instruct	3B	74.17	90.50	79.50	79.00	80.79	70.80	51.63	61.22
	Qwen2.5-1.5B-Instruct	1.5B	72.42	87.00	81.50	75.50	79.11	62.07	43.23	52.65
Function Calling	GRANITE	20B	72.83	91.50	84.00	81.50	82.46	77.27	58.00	67.64
	xLAM-7b-fc-r	7B	73.08	93.50	87.00	84.00	84.40	67.26	58.96	63.11
	xLAM-1b-fc-r	1.3B	69.67	89.50	79.00	66.50	76.17	64.86	50.58	57.72
	OpenFunctions-v2	7B	**83.27**	93.00	85.50	66.00	81.94	72.93	51.26	62.10
	Hammer2.1-7B	7B	78.08	**95.00**	**93.50**	**88.00**	**88.65**	**80.93**	**64.60**	**72.77**
	Hammer2.1-3B	3B	81.42	**95.00**	89.50	81.50	86.86	80.31	62.83	71.57
	Hammer2.1-1.5B	1.5B	74.67	92.00	84.50	80.00	82.79	77.42	61.17	69.30
	Hammer2.1-0.5B	0.5B	68.00	83.00	71.50	54.00	69.13	77.10	60.67	68.89
Inference Scaling	Hammer2.1-7B (Base)	7B	78.08	95.00	93.50	88.00	88.65	80.93	64.60	72.77
	+ Token-level Beam Search		74.58	93.50	91.50	82.50	85.52	79.69	62.37	71.03
	+ Marjority		**79.58**	95.00	93.50	85.00	88.27	79.03	65.26	72.15
	+ Best of N (ORM)		76.58	94.50	91.50	87.00	87.40	78.86	**67.73**	73.30
	+ ToolPRM (Ours)		79.08	**95.50**	**94.50**	**89.00**	**89.52**	**81.42**	65.30	**73.36**
	Hammer2.1-3B (Base)	3B	**81.42**	95.00	89.50	81.50	86.86	80.31	62.83	71.57
	+ Token-level Beam Search		74.50	91.00	87.50	77.00	82.50	78.29	59.32	68.81
	+ Marjority		79.50	95.00	89.00	81.00	86.13	76.00	58.27	67.14
	+ Best of N (ORM)		77.08	93.50	89.50	84.00	86.02	77.29	60.16	68.73
	+ ToolPRM (Ours)		80.50	**95.50**	**91.50**	**88.00**	**88.88**	**80.78**	**63.13**	**71.96**
	Hammer2.1-1.5B (Base)	1.5B	74.67	**92.00**	84.50	80.00	82.79	77.42	61.17	69.30
	+ Token-level Beam Search		72.33	91.50	81.00	73.50	79.58	73.03	56.68	64.86
	+ Marjority		77.50	91.50	84.50	80.00	83.38	72.88	**63.61**	68.25
	+ Best of N (ORM)		75.25	90.00	86.50	84.00	83.94	77.42	61.66	69.54
	+ ToolPRM (Ours)		**78.42**	**92.00**	**87.50**	**84.50**	**85.61**	**82.68**	63.18	**72.93**

some instances, increased N even leads to a noticeable degradation in accuracy. This phenomenon validates our core insights that increased retention of wrong intermediate steps (*i.e.*, larger N) can misguide subsequent generation steps. With higher generation budgets, the benefits derived from expanded exploration (scaling beam width) tend to outweigh those derived from increased retention (scaling N), *i.e.*, explore more but retain less.

Moreover, our ToolPRM approaches generally demonstrate superior performance when compared to the baseline strategies "Best of N" and "Majority Voting". The "Best of N" strategy shows a modest increase in averaged F1 score with a larger budget, while the 'Majority' strategy generally lags behind. Overall, fine-grained beam search with ToolPRM appears to be a robust approach, particularly when a larger generation budget is available and the "explore more but retain less" principle is followed, highlighting the importance of exploration guided by a fine-grained process reward model for structured function calling generation.

5 Conclusion

In this paper, we propose a fine-grained inference scaling framework to enhance LLM performance on structured function calling tasks. We construct an intra-call step-annotated dataset to train ToolPRM, a process reward model that supervises each intermediate step. Integrated with beam search, ToolPRM achieves the highest supervision accuracy and enables base LLMs to attain state-of-the-art results. We also identify a key principle for structured inference scaling: "explore more but retain less" based on the unrecoverability of structured output generation. However, the optimal trade-off in this principle is not yet dynamically adjustable. Future work could explore adaptive

strategies that calibrate exploration and retention based on input complexity or ToolPRM-derived confidence.

Limitations

While ToolPRM improves fine-grained guidance for tool use, it still assumes a discretized, step-wise view of decision making, which may not capture all forms of implicit reasoning or latent uncertainty. The approach focuses on rewarding intermediate structure and consistency, and thus cannot guarantee global optimality of the final tool choice or argument specification in every case. In addition, the framework introduces additional modeling components (e.g., masking design and state definitions), whose choices may affect behavior and require careful implementation.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by the National Key R&D Program of China (2024YFC3505402) and the National Natural Science Foundation of China (624B2096,62322603,U2244217,72542012,72595872).

References

Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, GP Bhargav, Maxwell Crouse, Chulaka Gunasekara, and 1 others. 2024. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks. *arXiv preprint arXiv:2407.00121*.

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. 1985. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169.

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

Huacan Chai, Zijie Cao, Maolin Ran, Yingxuan Yang, Jianghao Lin, Xin Peng, Hairui Wang, Renjie Ding, Ziyu Wan, Muning Wen, and 1 others. 2025. Parl-mt: Learning to call functions in multi-turn conversation with progress awareness. *arXiv preprint arXiv:2509.23206*.

Sehyun Choi, Tianqing Fang, Zhaowei Wang, and Yangqiu Song. 2023. Kcts: knowledge-constrained tree search decoding with token-level hallucination detection. *arXiv preprint arXiv:2310.09044*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Chia-Yu Hung, Navonil Majumder, Ambuj Mehrish, and Soujanya Poria. 2025. Reward-guided tree search for inference time alignment of large language models. 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 12575–12593.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, and 1 others. 2025. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv preprint arXiv:2504.09037*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, and 1 others. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*.

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. 2023. Don't throw away your value model! generating more preferable text with value-guided monte-carlo tree search decoding. *arXiv preprint arXiv:2309.15028*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024a. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, and 1 others. 2024b. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. 2023. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*.

Vaskar Nath, Pranav Raja, Claire Yoon, and Sean Hendryx. 2025. Toolcomp: A multi-tool reasoning & process supervision benchmark. *arXiv preprint arXiv:2501.01290*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Kanghee Park, Timothy Zhou, and Loris D'Antoni. 2025. Flexible and efficient grammar-constrained decoding. *arXiv preprint arXiv:2502.05111*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Bartosz Piotrowski, Witold Drzewakowski, Konrad Staniszewski, and Piotr Miłoś. 2025. Lightweight latent verifiers for efficient meta-generation strategies. *arXiv preprint arXiv:2504.16760*.

Isha Puri, Shivchander Sudalairaj, Guangxuan Xu, Kai Xu, and Akash Srivastava. 2025. A probabilistic inference approach to inference-time scaling of llms using particle-based monte carlo methods. *arXiv preprint arXiv:2502.01618*.

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

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

Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. 2023. Nexusraven: a commercially-permissive language model for function calling. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zhuoer Wang, Leonardo FR Ribeiro, Alexandros Papangelis, Rohan Mukherjee, Tzu-Yen Wang, Xinyan Zhao, Arijit Biswas, James Caverlee, and Angeliki Metallinou. 2024a. Fantastic sequences and where to find them: Faithful and efficient api call generation through state-tracked constrained decoding and reranking. *arXiv preprint arXiv:2407.13945*.

Zhuoer Wang, Leonardo FR Ribeiro, Alexandros Papangelis, Rohan Mukherjee, Tzu-Yen Wang, Xinyan Zhao, Arijit Biswas, James Caverlee, and Angeliki Metallinou. 2024b. Fantastic sequences and where to find them: Faithful and efficient api call generation through state-tracked constrained decoding and reranking. *arXiv preprint arXiv:2407.13945*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Yunjia Xi, Jianghao Lin, Yongzhao Xiao, Zheli Zhou, Rong Shan, Te Gao, Jiachen Zhu, Weiwen Liu, Yong Yu, and Weinan Zhang. 2025. A survey of llm-based deep search agents: Paradigm, optimization, evaluation, and challenges. *arXiv preprint arXiv:2508.05668*.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. 2023. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36:41618–41650.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, and 1 others. 2025. A survey of ai agent protocols. *arXiv preprint arXiv:2504.16736*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan. 2023. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*.

Zhuosheng Zhang, Yao Yao, Aston Zhang, Xiangru Tang, Xinbei Ma, Zhiwei He, Yiming Wang, Mark Gerstein, Rui Wang, Gongshen Liu, and 1 others. 2025. Igniting language intelligence: The hitchhiker's guide from chain-of-thought reasoning to language agents. *ACM Computing Surveys*, 57(8):1–39.

Congmin Zheng, Jiachen Zhu, Jianghao Lin, Xinyi Dai, Yong Yu, Weinan Zhang, and Mengyue Yang. 2025a. Cold: Counterfactually-guided length debiasing for process reward models. *arXiv preprint arXiv:2507.15698*.

Congming Zheng, Jiachen Zhu, Zhuoying Ou, Yuxiang Chen, Kangning Zhang, Rong Shan, Zeyu Zheng, Mengyue Yang, Jianghao Lin, Yong Yu, and 1 others. 2025b. A survey of process reward models: From outcome signals to process supervisions for large language models. *arXiv preprint arXiv:2510.08049*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, and 1 others. 2026. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. 2025. Steporlm: A self-evolving framework with generative process supervision for operations research language models. *arXiv preprint arXiv:2509.22558*.

Jiachen Zhu, Congmin Zheng, Jianghao Lin, Kounianhua Du, Ying Wen, Yong Yu, Jun Wang, and Weinan Zhang. 2025. Retrieval-augmented process reward model for generalizable mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8453–8468.

A Prompts for Function Call Generation

```
You are a helpful assistant.<|im_end|>

<|im_start|>user
[BEGIN OF TASK INSTRUCTION]
You are a tool calling assistant. In
    order to complete the user's
    request, you need to select one or
    more appropriate tools from the
    following tools and fill in the
    correct values for the tool
    parameters. Your specific tasks are:
1. Make one or more function/tool calls
    to meet the request based on the
    question.
2. If none of the function can be used,
    point it out and refuse to answer.
3. If the given question lacks the
    parameters required by the
    function, also point it out.

The following are characters that may
    interact with you
1. user: Provides query or additional
    information.
2. tool: Returns the results of the
    tool calling.
[END OF TASK INSTRUCTION]

[BEGIN OF AVAILABLE_TOOLS]
[{"name": "get_weather", "description":
    "get information about the
    weather", "parameters": {...}}]
[END OF AVAILABLE_TOOLS]

[BEGIN OF TASK INSTRUCTION]
The output MUST strictly adhere to the
    following JSON format, and NO other
    text MUST be included.
The example format is as follows.
    Please make sure the parameter type
    is correct. If no function call is
    needed, please directly output an
    empty list '[]'

[
{"name": "func_name1", "arguments":
    {"argument1": "value1",
    "argument2": "value2"}},
... (more tool calls as required)
]

[END OF TASK INSTRUCTION]

<|im_end|>

<|im_start|>user
Please help me check the weather
    <|im_end|>

<|im_start|>assistant
```

Listing 1: Prompt for generating a function call.

Table 4: BFCL results of self-reflection baselines. Avg. is the unweighted average over the four splits.

Model	Simple	Multi.	Paral.	Mul.P.	Avg.
Qwen2.5-32B	72.83	94.00	93.50	88.50	87.21
Qwen2.5-32B + Refl.	74.17	**95.50**	93.50	**89.00**	88.04
Qwen2.5-7B	75.33	94.50	91.50	84.50	86.46
Qwen2.5-7B + Refl.	77.17	**95.50**	91.50	83.50	86.92
Hammer2.1-7B + ToolPRM	**79.08**	**95.50**	**94.50**	**89.00**	**89.52**

B Additional Experimental Results

B.1 Can Advanced Reasoning Models Self-Correct Structural Errors?

An important question is whether stronger reasoning models can implicitly recover from early structural errors through self-reflection, which would challenge our assumption that such errors are difficult to repair once they occur. To investigate this possibility, we additionally evaluate Qwen2.5-Instruct models with a self-reflection strategy at inference time on BFCL. Specifically, we compare the original Qwen2.5-7B-Instruct and Qwen2.5-32B-Instruct models against their self-reflection variants, and further contrast them with our Hammer2.1-7B model guided by ToolPRM.

As shown in Table 4, self-reflection yields only minor gains for Qwen2.5-Instruct models and still consistently underperforms our 7B model with ToolPRM. This result suggests that post-hoc reflection is limited for structured outputs such as JSON function calls. Unlike open-ended reasoning tasks, where a model may revise its line of thought in subsequent sentences, structured generation is inherently brittle: a hallucinated argument value or a subtle early syntax error can invalidate the entire output while remaining difficult for the model itself to detect and repair afterward. Therefore, these results further support our premise that early errors in structured outputs are functionally unrecoverable in practice, motivating fine-grained intra-call pruning rather than relying on delayed self-correction.

B.2 Sensitivity to Function Masking

We further clarify the mechanics and robustness of the function masking strategy used in ToolPRM training. Function masking is only applied to a subset of the training samples, rather than the entire dataset. Its role is not merely to prevent memorization of tool names, but also to act as a regularization mechanism that encourages the model to rely on contextual understanding of tool descriptions and argument semantics instead of surface-form cues.

Table 5: Sensitivity of ToolPRM to function masking on BFCL. FM denotes function masking.

Model	Simple	Multi.	Paral.	Mul.P.	Avg.
Hammer2.1-1.5B	74.67	92.00	84.50	80.00	82.79
ToolPRM w/o FM	78.08	91.50	86.50	83.50	84.90
ToolPRM with FM	**78.42**	**92.00**	**87.50**	**84.50**	**85.61**

Table 6: Comparison between ToolPRM and the constrained decoding baseline FANTASE (Wang et al., 2024b) on BFCL.

Model	Simple	Multi.	Paral.	Mul.P.	Avg.
Hammer2.1-1.5B	74.67	92.00	84.50	80.00	82.79
+FANTASE	76.58	91.50	86.50	82.50	84.27
+ToolPRM (Ours)	**78.42**	**92.00**	**87.50**	**84.50**	**85.61**

To further validate this design, we conduct an additional ablation study on BFCL by training ToolPRM with and without function masking. Table 5 shows that removing function masking leads to consistently worse overall performance, and more importantly, weakens the efficacy of inference-time guidance. These results suggest that partial function masking improves the robustness of ToolPRM and helps it provide more reliable guidance during structured decoding.

B.3 Comparison with Constrained Decoding Baselines

We further compare ToolPRM against constrained decoding baselines by adding FANTASE (Wang et al., 2024b), a state-tracked constrained decoding framework paired with reranking. For a fair comparison, we equip FANTASE with a RoBERTa reranker that acts as an outcome reward model trained on our constructed ToolPRM dataset. As shown in Table 6, ToolPRM consistently outperforms the constrained decoding baseline across all splits on BFCL. This result suggests that enforcing state-level constraints and reranking alone is insufficient to match the benefits of fine-grained process supervision during structured generation.

B.4 Generalization Beyond BFCL and ToolAlpaca

To evaluate whether ToolPRM generalizes beyond BFCL and ToolAlpaca, we conduct additional experiments on the API-Bank benchmark (Li et al., 2023). Following prior work on Hammer (Lin et al., 2024), we use the cleaned version of API-Bank, which contains 314 tool-use dialogues and 753 API calls. This benchmark evaluates a model's ability to correctly invoke a known API from a query (L1), as well as to select and call APIs from a candidate list (L2).

Table 7: Generalization results on API-Bank (Li et al., 2023). Following Hammer (Lin et al., 2024), we report results on the cleaned test set.

API-Bank (L1)			
Model	F1-API	F1-Args	Avg.
Hammer2.1-1.5B	96.55	89.96	93.26
+Token-level Beam Search	91.41	84.78	88.10
+Majority	93.93	85.71	89.82
+Best of N	97.29	90.25	93.77
+ToolPRM (Ours)	**97.82**	**90.88**	**94.35**
API-Bank (L2)			
Model	F1-API	F1-Args	Avg.
Hammer2.1-1.5B	85.09	67.82	76.46
+Token-level Beam Search	81.50	63.84	72.67
+Majority	85.71	68.57	77.14
+Best of N	86.98	68.49	77.74
+ToolPRM (Ours)	**87.32**	**68.89**	**78.11**

As shown in Table 7, ToolPRM consistently outperforms the base model and other inference-time baselines on both L1 and L2. These results validate that the fine-grained supervision learned by ToolPRM generalizes to more complex and out-of-domain tool-use environments, beyond the benchmark settings used in our main experiments.