

BalanceSFT: Improving LLM Function Calling with Balanced Training Signals and Data Hardness

Bingguang Hao^{1,2*†}, ZengZhuang Xu^{2*}, Maolin Wang^{1*†}, Yuntao Wen^{2*†},
Yicheng Chen^{2†}, Cunyin Peng², Long Chen²,
Xiangyu Zhao¹, Jinjie Gu², Chenyi Zhuang^{2‡}, Ji Zhang^{3‡}

¹City University of Hong Kong, ²InclusionAI, Ant Group, ³Southwest Jiaotong University
{bingguanghao7, jizhang.jim}@gmail.com {chenyi.zcy}@antgroup.com

Abstract

While Supervised Fine-Tuning (SFT) is the prevailing method for equipping Large Language Models (LLMs) with function calling capabilities, its effectiveness is often compromised by two critical challenges: 1) **Imbalanced Training Signals**, where lengthy Chain-of-Thought (CoT) reasoning tokens dominate the training signals over concise function calls in the learning objective, and 2) **Imbalanced Data Hardness**, characterized by a scarcity of hard training examples. To overcome these limitations, we propose Balanced Supervised Fine-tuning (**BalanceSFT**), a novel framework that incorporates two key components: a Self-adjusted Signal Balancing (SSB) loss that employs a learnable hyperparameter to dynamically adjust the token contributions of CoT reasoning and function calls, together with a Hard Data Re-sampling (HDR) strategy that establishes a feedback loop to selectively generate new, high-quality complex data guided by model errors. Extensive experiments demonstrate the effectiveness of our proposed BalanceSFT framework. With BalanceSFT, a 7B model achieves function calling performance that surpasses state-of-the-art models like GPT-5. Our code, models, and dataset are open-sourced.

1 Introduction

The rapid advancement of Large Language Models (LLMs) has marked a transformative era in natural language processing (Ouyang et al., 2022; Achiam et al., 2023; Touvron et al., 2023; Bai et al., 2023), demonstrating exceptional capabilities in text understanding, generation, and complex reasoning (Zhong et al., 2024). A pivotal direction in enhancing their practicality is equipping them with function calling abilities, enabling LLMs to interact with external tools and APIs to execute concrete actions and access real-world data (Wang et al.,

*Equal contribution. †Work done at Ant Group.

‡Corresponding Author. Project Page: [BalanceSFT](#)



Figure 1: **Two challenges in LLM Function Calling.** (a) **Imbalanced Training Signals:** The lengthy Chain-of-Thought (CoT) tokens dominate the learning signal, overshadowing the concise but critical function call. (b) **Imbalanced Data Hardness:** The training dataset is dominated by simple examples, with a scarcity of hard cases necessary for robust performance.

2026; Singh et al., 2024). Supervised Fine-Tuning (SFT) has emerged as the standard paradigm to instill this capability, typically by training models on demonstrations that combine a Chain-of-Thought (CoT) reasoning process with the final function call (Lightman et al., 2023; Guo et al., 2025; Team et al., 2025).

Despite the remarkable progress, conventional SFT approaches suffer from two inherent and critical imbalances that severely limit their effectiveness (Ju et al., 2024). On the one hand, as illustrated in Figure 1, the training signals are skewed due to a token-level imbalance. In a typical function-calling sample, the lengthy CoT reasoning tokens numerically dominate the concise, yet

critical, function call tokens (Qian et al., 2025). This training-signal imbalance causes the model to be optimized primarily for generating plausible reasoning, often at the expense of precision in the final tool execution. On the other hand, the data distribution is characterized by an Imbalanced Data Hardness (Li et al., 2023; Zhang et al., 2025b). Existing training datasets are naturally dominated by simple examples, while complex and challenging cases that are crucial for robust performance are scarce. This scarcity prevents the model from effectively learning to handle edge cases and intricate scenarios (Zeng et al., 2025b).

To tackle the dual challenges, we propose **Balanced Supervised Fine-Tuning (BalanceSFT)**, a novel framework designed to recalibrate the SFT process. BalanceSFT incorporates two simple yet effective components: 1) A Self-adjusted Signal Balancing (**SSB**) loss function that introduces a learnable hyperparameter to dynamically balance the contribution of CoT tokens and function call tokens to the total loss, ensuring that the model does not overlook the critical execution steps; 2) A Hard Data Re-sampling (**HDR**) strategy that establishes a feedback loop to generate new, high-quality complex data guided by model errors. Extensive experiments validate the effectiveness of our BalanceSFT framework. The results show that a 7B model trained with BalanceSFT achieves function calling performance that surpasses state-of-the-art models like GPT-5 (Singh et al., 2025) on the comprehensive Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2024) and other specialized benchmarks (Chen et al., 2025a; Li et al., 2023).

Our main contributions are threefold:

- We identify two critical yet overlooked limitations in standard SFT for tool learning: the imbalance in training signals between lengthy reasoning and concise function calls, and the imbalance in data hardness.
- We propose BalanceSFT, a novel framework that introduces a Self-adjusted Signal Balancing (SSB) loss and a Hard Data Re-sampling (HDR) strategy to calibrate the learning focus.
- We train a 7B model with the proposed BalanceSFT framework, achieving function calling performance that surpasses state-of-the-art models like GPT-5.

2 Related Work

Function Calling for LLMs. The ability of LLMs to interact with external tools and APIs through function calling has emerged as a crucial aspect of their practical utility (Wang et al., 2024b; Singh et al., 2024; Patil et al., 2024). This capability allows LLMs to transcend the limitations of solely processing and generating text, enabling them to ground their responses in real-world data and automate complex tasks (Wang et al., 2024a; Kavathekar et al., 2025; Zhang et al., 2025a). Early explorations in this area focused on enabling LLMs to understand descriptions of functions and generate the necessary calls with appropriate parameters (Wang et al., 2023). By training LLMs on datasets specifically curated for function calling scenarios, these methods aim to instill a deeper understanding of function semantics and parameter requirements (Prabhakar et al., 2025; Zeng et al., 2025b). The quality and diversity of these fine-tuning datasets play a critical role in the resulting performance of the LLMs in function calling tasks. RL has also been explored as a means to optimize LLMs for function calling (Zhang et al., 2025c; Qian et al., 2025). RL-based approaches often involve defining reward functions that incentivize the generation of correct and executable function calls. While RL-based approaches offer the potential to optimize directly for task success, they still suffer from sparse rewards and a lack of effective data (Qu et al., 2025; Zeng et al., 2025a).

Improving Reasoning Capability via SFT. The integration of CoT with function calling mechanisms holds substantial promise for enhancing the accuracy and reliability of tool-use (Lu et al., 2025; Chen et al., 2024). By reasoning through the user’s request and identifying the most appropriate course of action, including the selection of relevant tools and their parameters, the model can make more informed decisions about function invocation (Li, 2025; Shen, 2024). The explicit reasoning process provided by CoT can also improve the transparency and interpretability of the model’s actions (Wu et al., 2025; Dong et al., 2025). However, the integration of CoT into the training process for function calling introduces a unique challenge related to the design of the loss function (Yang et al., 2025b). Traditional training methodologies, primarily SFT, typically treat the entire generated sequence (including the reasoning process and the final function call) uniformly when calculating the loss (Lobo

et al., 2025). This can lead to an imbalance, where the lengthy sequences of reasoning steps dominate the loss calculation, potentially overshadowing the importance of the final, often shorter, function call. Consequently, the model might be incentivized to generate elaborate and seemingly plausible reasoning chains (Agarwal et al., 2024; Chen et al., 2025b), even if they do not ultimately lead to a correct or executable function call. This inherent tension between the verbose reasoning process and the need for a succinct and precise function call necessitates a more nuanced approach to loss function design that can effectively balance these two critical aspects of the task. Our work addresses this challenge by introducing a Self-adjusted Signal Balancing approach that explicitly considers the different roles and importance of the reasoning process and the final function execution during training.

3 Method

In this section, we elaborate on our proposed Balanced Supervised Fine-Tuning (BalanceSFT) framework, which is designed to address the dual challenges of Imbalanced Training Signals and Imbalanced Data Hardness.

3.1 Overview

The overall architecture of our BalanceSFT framework is illustrated in Figure 2. It consists of two core components that work in synergy: 1) A Self-adjusted Signal Balancing (SSB) loss that dynamically balances the learning signals between lengthy Chain-of-Thought reasoning tokens and concise function call tokens during training; and 2) a Hard Data Re-sampling (HDR) strategy that establishes a feedback loop to selectively generate and utilize high-quality complex data based on model errors. These two components address the signal imbalance and data hardness imbalance, respectively, forming a comprehensive solution for robust function calling capability.

3.2 Self-adjusted Signal Balancing (SSB) Loss

Motivation and Analysis. We conduct an intuitive analysis of a CoT dataset. As shown in Table 1, a CoT Dataset distilled from QwQ-32B (Qwen Team, 2025) based on the open-source function call dataset xlam-function-calling-60k (Zhang et al., 2025b) exhibits a significant token imbalance: Chain-of-Thought reasoning segments contain approximately 10 times more tokens (mean=350.74)

Dataset	CoT Token Len.		Result Token Len.	
	Mean	Median	Mean	Median
CoT Dataset	350.74	248.00	31.07	27.00

Table 1: Token Statistics of the CoT Dataset for function calling. Note the approximately 10x higher token count for the CoT compared to the final execution.

than function call results (mean=31.07). This substantial disparity highlights the need to rebalance the training focus toward the short but critical function execution components.

Formulation. The output of an LLM \mathcal{M} can be decomposed into two components: Chain-of-Thought t and the final function calling execution f . Let N_t and N_f denote the token counts for these components, respectively, with $N_{\text{all}} = N_t + N_f$ representing the total tokens, and V represents the size of the vocabulary. The standard SFT loss can be written as the average token-level cross-entropy:

$$L_{\text{SFT}} = \frac{\sum_{i=1}^{N_{\text{all}}} \sum_{j=1}^V -y_{ij} \log p_{ij}}{N_{\text{all}}} \quad (1)$$

We decompose the loss L_{SFT} as follows:

$$\begin{aligned} L_{\text{SFT}} &= \frac{N_t}{N_{\text{all}}} \cdot \frac{1}{N_t} \sum_{i=1}^{N_t} \sum_{j=1}^V -y_{ij} \log p_{ij} \\ &+ \frac{N_f}{N_{\text{all}}} \cdot \frac{1}{N_f} \sum_{i=N_t+1}^{N_t+N_f} \sum_{j=1}^V -y_{ij} \log p_{ij} \\ &= w_t \cdot L_{\text{think}} + w_f \cdot L_{\text{result}} \end{aligned} \quad (2)$$

where $w_t = \frac{N_t}{N_{\text{all}}}$, $w_f = \frac{N_f}{N_{\text{all}}}$, thus $w_t + w_f = 1$. Let L_{think} and L_{result} represent the loss for reasoning and function call components, respectively.

The statistical analysis from Table 1 reveals that $N_t \gg N_f$, which implies $w_t \gg w_f$ during standard training. This naturally leads to the model being optimized primarily for generating plausible reasoning at the expense of precise function execution. To address this inherent imbalance, we propose the Self-adjusted Signal Balancing (SSB) loss:

$$L_{\text{SSB}} = \alpha \cdot L_{\text{think}} + (1 - \alpha) \cdot L_{\text{result}} \quad (3)$$

where α is a learnable hyperparameter with $\alpha \in [0, 1]$. The introduction of α allows the model to dynamically adjust its focus between reasoning depth and execution accuracy based on performance requirements. In practice, we set α as a trainable parameter to avoid extensive hyperparameter tuning.

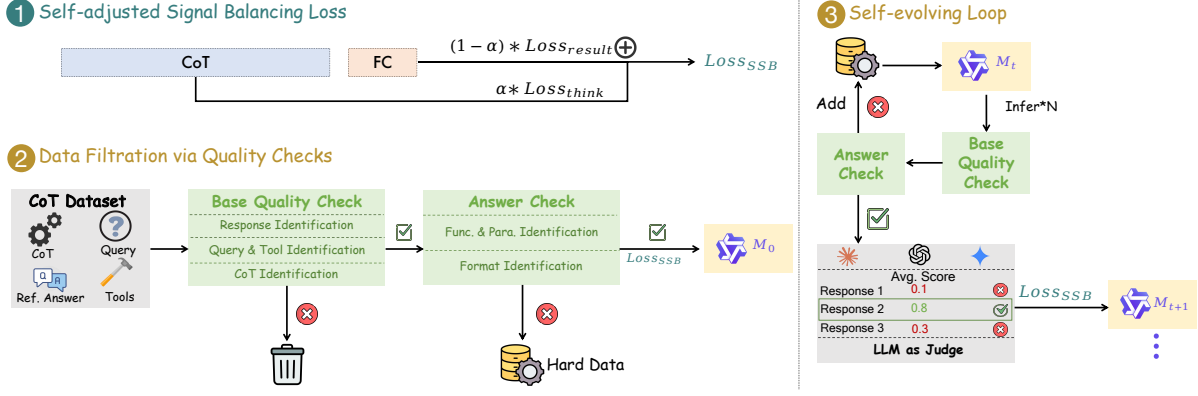


Figure 2: **Overview of the BalanceSFT framework.** It starts with a standard function call dataset, which is refined through a Base Quality Check and Answer Check to create initial training data and identify hard data. The model is first initialized via a Cold Start using the **Self-adjusted Signal Balancing (SSB) Loss**. Subsequently, the **Hard Data Re-sampling (HDR) strategy** creates a **Self-evolving Loop** where the model iteratively reasons on hard cases, generates new solutions, and undergoes quality-gated retraining.

3.3 Hard Data Re-sampling (HDR) Strategy

The core objective of the HDR strategy is to address the Imbalanced Data Hardness issue through a carefully designed self-evolving data refinement loop. This process can be formally described as a sequence of iterative data filtration and augmentation operations. Let $\mathcal{D}_{\text{initial}}$ denote the initial training dataset. Each data point $d_i = (q_i, c_i, f_i)$ consists of a user query q_i , a CoT reasoning sequence c_i , and a function call f_i .

Data Filtration via Quality Checks. The dataset undergoes a two-stage filtration process defined by boolean functions $\text{BQC}(\cdot)$ and $\text{AC}(\cdot)$ for Base Quality Check (including response, query, and CoT identification) and Answer Check (including function, parameter, and format verification), respectively:

$$\begin{aligned} \mathcal{D}_{\text{qualified}} &= \{d_i \in \mathcal{D}_{\text{initial}} \mid \text{BQC}(d_i) \wedge \text{AC}(d_i)\}, \\ \mathcal{D}_{\text{hard}} &= \{d_i \in \mathcal{D}_{\text{initial}} \mid \text{BQC}(d_i) \wedge \neg \text{AC}(d_i)\}. \end{aligned} \quad (4)$$

Here, $\mathcal{D}_{\text{qualified}}$ is used for the Cold Start phase to initialize the model \mathcal{M}_0 , while $\mathcal{D}_{\text{hard}}$ is reserved for iterative self-evolution.

Self-evolving Loop. The model at iteration t , denoted as \mathcal{M}_t , is used to reason on the current hard dataset $\mathcal{D}_{\text{hard}}^t$, generating a new set of candidate responses \mathcal{R}^{t+1} :

$$\mathcal{R}^{t+1} = \{\mathcal{M}_t(q_i) \mid \forall q_i \in \mathcal{D}_{\text{hard}}^t\} \quad (5)$$

Each candidate response $r_j \in \mathcal{R}^{t+1}$ is evaluated by an ensemble of LLM judges $\mathcal{J} = \{J_1, J_2, \dots, J_k\}$ (e.g., Gemini-2.5-Pro (Comanici et al., 2025), GPT-

4o (Hurst et al., 2024), Claude-3.5-Sonnet (Anthropic, 2024)) via ensemble voting. The scoring function $\text{Score}(r_j)$ aggregates their judgments:

$$\text{Score}(r_j) = \mathbb{I} \left(\frac{1}{k} \sum_{m=1}^k J_m(r_j) \geq \tau \right) \quad (6)$$

where $\mathbb{I}(\cdot)$ is the indicator function and τ is a passing threshold. Responses meeting the quality standard form the new high-quality dataset:

$$\mathcal{D}_{\text{new}}^t = \{r_j \in \mathcal{R}^{t+1} \mid \text{Score}(r_j) = 1\} \quad (7)$$

The model is then fine-tuned on this newly generated data:

$$\mathcal{M}_{t+1} = \text{SFT}(\mathcal{M}_t, \mathcal{D}_{\text{new}}^t) \quad (8)$$

Simultaneously, the hard dataset is updated by retaining failed samples:

$$\mathcal{D}_{\text{hard}}^{t+1} = \{d_i \in \mathcal{D}_{\text{hard}}^t \mid \text{Score}(\mathcal{M}_t(q_i)) < 1\} \quad (9)$$

This iterative process continues until convergence criteria are met: $\mathcal{D}_{\text{hard}}^{t+1} = \emptyset$ or $t = T_{\text{max}}$.

The entire HDR strategy establishes a virtuous cycle where model improvement and data quality enhancement reinforce each other, effectively resolving the data hardness imbalance through the iterative refinement process:

$$\mathcal{M}_{t+1}, \mathcal{D}_{\text{hard}}^{t+1} = \text{HDR-Iteration}(\mathcal{M}_t, \mathcal{D}_{\text{hard}}^t) \quad (10)$$

4 Experiments

The experiments in this paper are organized into five parts: experimental settings, data preparation, comparative evaluations of BalanceSFT, an ablation study, and further analysis of the proposed method.

4.1 Experiment Settings

Dataset. As shown in Table 2, our training dataset comprises two components: xlam-function-calling-60k (Zhang et al., 2025b) and Open-Agentic-tool-use (Aworld team, 2025). xlam-function-calling-60k contains 60,000 samples from 3,673 APIs across 21 categories, while Open-Agentic-tool-use includes 30,000 samples from 129 APIs across 8 categories. The former focuses on single-turn tool-use to strengthen the model’s basic capabilities, while the latter focuses on multi-turn tool-use to address complex scenario requirements. The distribution of the diverse and comprehensive API resources is shown in Appendix A, Figure 5.

Benchmarks. To assess our methodology, we choose a highly representative benchmark, the Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2024). For BFCL, we conduct evaluations on BFCLv3, which contains single-turn and multi-turn function calling scenarios. Performance across all subsets of BFCL is reported in terms of accuracy (%). Beyond BFCL, we further evaluate our model on two specialized tool-use benchmarks, ACEBench (Chen et al., 2025a) and APiBank (Li et al., 2023).

Backbones. To demonstrate the effectiveness and general applicability of our BalanceSFT framework, we select Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) as our primary base model. To demonstrate the effectiveness of our method, we also conduct additional experiments using Llama-3.2-3B-Instruct (Grattafiori et al., 2024).

Baselines. For comprehensive comparison, we include both open-source and closed-source models: open-source models include Moonshotai-Kimi-K2-Instruct (Team et al., 2025), DeepSeek-R1-0528 (Guo et al., 2025), Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025a), ToolACE-MT (8B) (Zeng et al., 2025b), and BitAgent-8B (BitAgent, 2025); closed-source models include GPT-5 (Singh et al., 2025), GPT-4o (Hurst et al., 2024), Gemini-2.5-Pro (Comanici et al., 2025), and Grok-4 (xAI, 2025).

Implementation Details. Our implementations are based on the training framework LLaMA Factory (Zheng et al., 2024). To ensure fair comparison, we maintain consistent hyperparameters across all experiments: global batch size of 512, learning rate of 4×10^{-5} , warm-up rate of 0.05. The SSB loss hyperparameter α is initialized to 0.7 and optimized as a learnable parameter during train-

Dataset	Samples	APIs	Categories
xlam-function-calling-60k	60000	3673	21
Open-Agentic-tool-use	30000	129	8

Table 2: Statistics of the xlam-function-calling-60k and Open-Agentic-tool-use datasets.

Models	Non-Live	Live	Overall
Qwen2.5-Coder-7B-Instruct	83.88	69.75	76.82
w/ Strategy CoT	87.00	66.81	76.91
w/ Natural CoT	85.67	71.73	78.70

Table 3: Performance of the prescribed strategy mode and the natural reasoning mode on the single-turn subset of BFCL. The two models are trained on Qwen2.5-Coder-7B-Instruct with the same training hyperparameters.

ing. During inference, we set the number of sampled generations to 3 with a temperature of 0.7. In HDR, for the LLM-as-judge phase, we use Gemini-2.5-Pro (Comanici et al., 2025), GPT-4o (Hurst et al., 2024), Claude-3.5-Sonnet (Anthropic, 2024) as judges with the same temperature setting at 0.7, threshold $\tau = 0.5$, and scoring range $[0, 1]$. The maximum number of iteration T_{max} is 10. All training runs were conducted on 8 NVIDIA H20 GPUs. To further assess the generalizability of our framework, we additionally report results using an open-source LLM judge, QwQ-32B (Qwen Team, 2025), in Appendix C.4.

4.2 Data Preparation

We construct two CoT datasets: (1) Strategy CoT, generated by GPT-4o following prescribed reasoning strategies, and (2) Natural CoT, generated by QwQ-32B (Qwen Team, 2025) based on its natural reasoning process (implemented with vLLM (Kwon et al., 2023), temperature = 0.1). We conduct standard SFT on Qwen2.5-Coder-7B-Instruct using each dataset. As shown in Table 3, we observe that Natural CoT generally outperforms Strategy CoT. Therefore, we adopt Natural CoT for cold-start data generation and employ self-inference in the Hard Data Re-sampling (HDR) stage to maintain model-consistent reasoning patterns.

4.3 Main Results

The performance of BalanceSFT-7B (built upon Qwen2.5-Coder-7B-Instruct) on the BFCL benchmark is summarized in Table 4. We report accuracy across the main subsets using the official BFCL

Models	Parameter Counts	Multi-Turn					Single-Turn		
		Overall	Base	Miss Func	Miss Param	Long Context	Overall	Non-Live	Live
Closed-Source Models									
GPT-5-2025-08-07	/	28.50	33.50	29.50	23.00	28.00	65.59	72.92	58.25
GPT-4o-2024-11-20	/	42.50	<u>55.50</u>	34.50	29.00	51.00	77.21	83.88	70.54
Gemini-2.5-Pro	/	25.00	25.50	26.00	24.50	24.00	74.50	85.04	63.95
Gemini-2.5-Flash	/	33.62	36.00	32.50	32.50	33.50	77.78	83.10	72.46
o3-2025-04-16	/	38.38	44.00	40.50	31.50	37.50	53.01	39.98	66.03
Amazon-Nova-Pro-v1:0	/	34.75	42.50	24.50	27.50	44.50	81.78	85.25	78.31
Grok-4-0709	/	36.12	44.00	31.00	26.00	43.50	79.80	85.21	74.39
Open-Source Models									
Moonshotai-Kimi-K2-Instruct	1043B	41.25	51.00	<u>43.00</u>	31.00	40.00	80.80	84.02	77.57
DeepSeek-R1-0528	671B	<u>44.50</u>	54.50	41.00	<u>36.50</u>	46.00	78.22	75.73	80.90
Qwen3-235B-A22B-Inst-2507	235B	39.62	53.50	34.50	27.50	43.00	<u>83.37</u>	90.12	76.61
Llama-4-Maverick	400B	17.88	23.50	18.00	14.00	16.00	80.90	88.15	73.65
Qwen3-8B	8B	24.12	26.00	29.00	20.50	21.00	83.20	<u>88.60</u>	77.79
BitAgent-8B	8B	37.75	46.50	37.50	24.00	43.00	81.71	87.33	76.09
watt-tool-8B	8B	37.88	45.50	39.00	24.00	43.00	81.71	87.54	75.87
ToolACE-MT	8B	40.25	57.50	31.50	34.00	38.00	78.23	84.94	71.52
Qwen2.5-Coder-7B-Instruct	7B	3.88	6.00	3.50	2.00	4.00	76.82	83.88	69.75
Ours									
BalanceSFT-7B	7B	47.00	53.50	47.00	41.00	<u>46.50</u>	84.00	88.29	<u>79.70</u>

Table 4: Performance on BFCL (last updated August 26, 2025), with all metrics calculated using the official script. The best result within each category is highlighted in **bold**. The second-best results are underlined.

evaluation script.

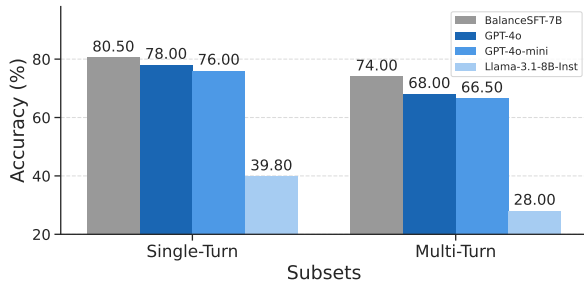
Multi-Turn Evaluation. The Multi-Turn subset evaluates a model’s ability to handle complex, continuous conversations. As shown in Table 4, BalanceSFT-7B demonstrates the strongest overall capability, achieving the highest Overall score of 47.00. It also achieves optimal performance in the key failure mode categories: Miss Function at 47.00 and Miss Parameter at 41.00, indicating its ability to effectively detect the absence of functions and required parameters in the dialogue flow. In contrast, the Long Context subset, which tests memory and coherence over conversation history, is led by the closed-source model GPT-4o-2024-11-20 (Hurst et al., 2024) at 51.00, with BalanceSFT-7B being the second best at 46.50.

Single-Turn Evaluation. The Single-Turn subset focuses on isolated prompts under Non-Live and Live settings. As reported in Table 4, BalanceSFT-7B maintains its competitive edge by achieving the highest Overall score of 84.00. The Non-Live subset is led by Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025a) with the highest score of 90.12. For the Live subset, simulating the real-world interactions, DeepSeek-R1-0528 (Guo et al., 2025) performs the best at 80.90, with BalanceSFT-

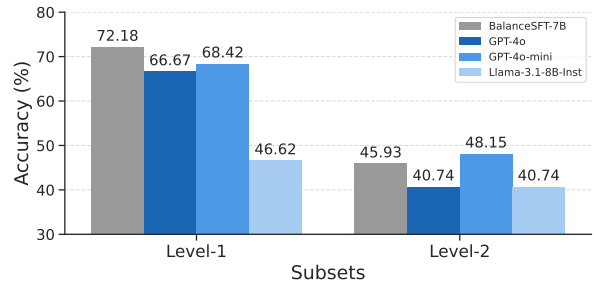
7B coming in a close second at 79.70.

Despite being a small model, BalanceSFT-7B outperforms larger proprietary and specialized models, revealing that our BalanceSFT strategy achieves superior performance, particularly in the multi-turn function calling scenario. These findings demonstrate that BalanceSFT-7B can accurately analyze system status and remains highly sensitive to key functions and parameters within context.

Results on Other Benchmarks. Figure 3 presents the performance of BalanceSFT-7B on single-turn and multi-turn tasks from ACEBench (Chen et al., 2025a), the Level-1 (Call) and Level-2 (Retrieve+Call) subsets from APiBank (Li et al., 2023). On ACEBench, BalanceSFT-7B achieves the highest accuracy of 80.50 in the single-turn task, outperforming GPT-4o, which scores 78.00. In the multi-turn task, it also attains the best result with an accuracy of 74.00, which is 6 percentage points higher than GPT-4o (68.00). On APiBank, BalanceSFT-7B maintains strong generalization, ranking first and second on the Level-1 and Level-2 subsets. These results confirm that the model trained with BalanceSFT exhibits robust and consistent performance across diverse tool-calling benchmarks.



(a) Performance on ACEBench.



(b) Performance on APIBank.

Figure 3: Performance comparison on ACEBench and APIBank benchmarks using official evaluation scripts, reported as accuracy (%).

4.4 Ablation Study

In the ablation study, all experiments are conducted under consistent configurations to ensure fair and controlled comparisons. The basic SFT method serves as the baseline to assess the contribution of each proposed component. To further examine the generalization of our method, we conduct experiments on two base models: Qwen2.5-Coder-7B-Instruct and Llama-3.2-3B-Instruct.

Effectiveness of the Designed Components. Our ablation analysis, presented in Table 5, systematically investigates the effect of the two core modules—Self-adjusted Signal Balancing (SSB) loss and Hard Data Re-sampling (HDR)—on both single-turn and multi-turn performance within the BFCL benchmark.

For Qwen2.5-Coder-7B-Instruct, the SFT trained model achieves 81.73 and 38.25 in Single-Turn and Multi-Turn, respectively. Individual activation of SSB improves performance to 82.20 (Single-Turn) and 41.62 (Multi-Turn). In contrast, applying SFT to HDR resampled data provides larger gains, boosting Single-Turn by 2.12 and dramatically increasing Multi-Turn by 4.87, suggesting that rare edge samples are critical. Crucially, the combination of both SSB and HDR resulted in synergistic gains, achieving the highest performance of 84.00 (Single-Turn) and 47.00 (Multi-Turn). These findings highlight the significant contribution of both modules to improving model robustness and accuracy.

A similar pattern can be observed for Llama-3.2-3B-Instruct, showing consistent trends, which further validates the generality of our approach. These results again validate that both SSB and HDR individually enhance performance, especially in the Multi-Turn scenario. Notably, their combination provides a complementary and superior method

Models	Modules		Single Turn	Multi Turn
	SSB	HDR		
Qwen2.5-Coder-7B-Instruct+SFT	×	×	81.73	38.25
	✓	×	82.20	41.62
	×	✓	83.85	43.12
	✓	✓	84.00	47.00
Llama-3.2-3B-Instruct+SFT	×	×	73.67	33.25
	✓	×	76.82	34.62
	×	✓	78.28	35.75
	✓	✓	78.99	36.12

Table 5: Performance comparison of Qwen2.5-Coder-7B-Instruct and Llama-3.2-3B-Instruct under different module combinations (SSB and HDR) on Single-Turn and Multi-Turn BFCL tasks.

for optimizing tool-use capabilities across different model architectures, i.e., yielding harder edge samples (through HDR) and enabling the model to learn more effectively (through SSB).

4.5 Further Analysis

Comparison with Reinforcement Learning. To further validate the effectiveness of the proposed BalanceSFT method, we compare it with Group Relative Policy Optimization (GRPO) (Guo et al., 2025) on BFCL, as shown in Figure 4(a). The 7B and 3B models are built on Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) and Llama-3.2-3B-Instruct (Grattafiori et al., 2024), respectively. We use the hyperparameter settings and reward design from a GRPO baseline detailed in Appendix B. For Qwen2.5-Coder-7B-Instruct, the GRPO method achieves a higher score in the Single-Turn task (84.60 vs. 84.00), but the BalanceSFT approach outperforms it significantly in the Multi-Turn task (47.00 vs. 40.75). This suggests that while GRPO maintains an edge in simpler, Single-Turn interactions, the BalanceSFT method better enhances the model’s capability to handle more complex, Multi-Turn dialogues. A similar trend is observed with

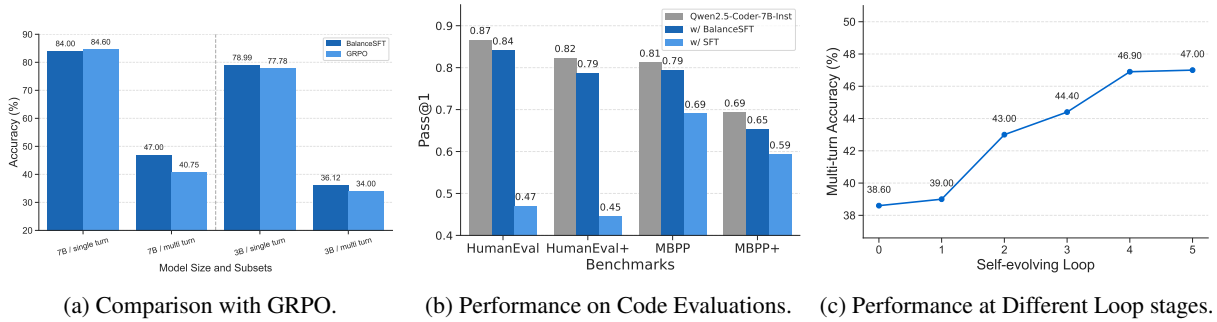


Figure 4: **More experiments.** (a) Performance of two series models trained by GRPO and BalanceSFT on BFCL. (b) Performance of BalanceSFT and SFT models on HumanEval and MBPP (including HumanEval+ and MBPP+) compared with that of the code pre-trained model (Qwen2.5-Coder-7B-Instruct). (c) Multi-Turn performance of BalanceSFT-7B at different stages in Self-evolving.

Llama-3.2-3B-Instruct, where BalanceSFT outperforms GRPO in both tasks, marginally in Single-Turn (78.99 vs. 77.78) and more notably in Multi-Turn (36.12 vs. 34.00). The results indicate that the proposed BalanceSFT approach provides a more balanced performance boost, particularly excelling in the more challenging Multi-Turn setting compared to the GRPO method, thereby validating its advantage in enhancing model robustness across different interaction complexities.

Effectiveness of Alleviating Catastrophic Forgetting. To assess how BalanceSFT impacts the coder model’s original coding capabilities, we evaluate our model on widely recognized code generation benchmarks in Figure 4(b), HumanEval and MBPP (including their variants HumanEval+ and MBPP+) (Austin et al., 2021; Chen et al., 2021; Yu et al., 2025). The results demonstrate a crucial advantage of our method. Specifically, BalanceSFT-7B maintains strong performance across all benchmarks, with pass@1 scores only marginally lower (within 4%) compared to the base Qwen2.5-Coder-7B-Instruct model. For instance, on HumanEval, BalanceSFT achieves 0.84 compared to the base model’s 0.87, and on MBPP, it reaches 0.79 versus 0.81. This minimal performance degradation stands in stark contrast to models fine-tuned using standard SFT. The SFT method shows dramatic drops in pass@1 scores on all the benchmarks, notably falling from 0.87 to 0.47 on HumanEval and from 0.82 to 0.45 on HumanEval+.

These results indicate that BalanceSFT effectively mitigates catastrophic forgetting, preserving the model’s original code capabilities across benchmarks. This is because the SSB loss dynamically rebalances the training signal, focusing updates on the concise, yet crucial function call tokens. By

concentrating learning on these key tokens, the model avoids overfitting to lengthy reasoning and maintains its original coding capabilities.

Effectiveness of HDR Self-evolving Loop. As shown in Figure 4(c), the model’s ability in Multi-Turn function call grows with the Self-evolving Loop until convergence, providing strong evidence that the proposed HDR can effectively and continuously identify more complex edge cases, driving model performance toward higher accuracy.

This improvement arises from the HDR Self-evolving Loop, which forms a self-reinforcing training cycle. HDR leverages model errors to create harder edge samples that target its weak points, forming a focused feedback loop for continual improvement. Through iterative refinement, the model progressively improves its accuracy and robustness on complex, multi-step tasks.

Hyperparameter Sensitivity. To assess the robustness of BalanceSFT to hyperparameter initialization, we conduct a sensitivity analysis by varying the initial α value from 0.5 to 0.9, as shown in Table 6. Despite different starting points, the final converged α values cluster within a narrow range of 0.68–0.72, demonstrating the stability of our optimization process. Specifically, the best performance is achieved when initializing α at 0.7, yielding the highest Single-Turn score of 84.00 and Multi-Turn score of 47.00. Other initializations result in slightly lower performance, with only minor fluctuations around this optimum. This suggests that while BalanceSFT is generally robust to the choice of initialization, starting from a moderate value (e.g., 0.7) can lead to slightly better outcomes. Overall, the adaptive balancing mechanism consistently drives α toward a similar converged range, ensuring stable training behavior.

Initial α	Final α	Single-Turn	Multi-Turn
0.5	0.68	83.42	45.12
0.6	0.68	83.75	45.80
0.7	0.69	84.00	47.00
0.8	0.70	83.88	46.10
0.9	0.72	83.51	45.67

Table 6: Performance and α convergence across different initializations.

5 Conclusion

In this work, we identify two critical yet overlooked challenges in standard SFT for function calling: the imbalance in training signals between lengthy reasoning and concise function calls, and the scarcity of hard training data. To address these issues, we propose BalanceSFT, a novel framework that integrates a Self-adjusted Signal Balancing (SSB) loss and a Hard Data Re-sampling (HDR) strategy. The SSB loss dynamically recalibrates the learning focus between reasoning and execution, while HDR iteratively enriches the training set with high-quality complex examples. Extensive experiments demonstrate that BalanceSFT enables a 7B model to achieve function calling performance that surpasses state-of-the-art models like GPT-5 on BFCL and other benchmarks. Our work provides a simple yet effective pathway toward more robust and generalizable LLM-based tool-use capabilities.

Limitations

Despite its strong performance, BalanceSFT has two major limitations. First, the effectiveness of the SSB loss depends on the initial quality and structure of the CoT data, which may limit its applicability in domains where high-quality reasoning annotations are scarce. Second, while HDR improves data hardness, it relies on an ensemble of LLM judges for quality control, which introduces computational cost and potential biases from the judge models.

Ethical Considerations

This work aims to improve function calling in large language models, which can increase the usefulness of LLMs in real-world applications but may also introduce risks such as unintended tool execution, privacy leakage, and misuse in sensitive scenarios. All data used in our experiments are publicly available or synthetically generated, and we do not use any personal information.

Acknowledgments

We would like to thank all reviewers for constructive comments that helped us improve the paper. This research was partially supported by National Natural Science Foundation of China (No.62502404), Hong Kong Research Grants Council (Research Impact Fund No.R1015-23, Collaborative Research Fund No.C1043-24GF, General Research Fund No. 11218325), Institute of Digital Medicine of City University of Hong Kong (No.9229503), and Ant Group Research Internship Program.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Chirag Agarwal, Sree Harsha Tanneru, and Himabindu Lakkaraju. 2024. Faithfulness vs. plausibility: On the (un) reliability of explanations from large language models. *arXiv preprint arXiv:2402.04614*.
- Anthropic. 2024. [Claude 3.5](#). Accessed on June 21, 2025.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- InclusionAI Aworld team. 2025. [Open-agentic-tool-use](#). Accessed on September 26, 2025.
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.
- BitAgent. 2025. [Bitagent-8b](#). Accessed on September 27, 2025.
- Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, and 1 others. 2025a. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman,

- Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, and 1 others. 2025b. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*.
- Zhi-Yuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. 2024. Towards tool use alignment of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1382–1400.
- Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.
- Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*.
- 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*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- 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*.
- Yiming Ju, Ziyi Ni, Xingrun Xing, Zhixiong Zeng, Hanyu Zhao, Siqi Fan, and Zheng Zhang. 2024. Mitigating training imbalance in llm fine-tuning via selective parameter merging. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15952–15959.
- Ishan Kavathekar, Raghav Donakanti, Ponnurangam Kumaraguru, and Karthik Vaidhyanathan. 2025. Small models, big tasks: An exploratory empirical study on small language models for function calling. In *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, pages 1117–1126.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- 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. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116.
- Xinzhe Li. 2025. A review of prominent paradigms for llm-based agents: Tool use, planning (including rag), and feedback learning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9760–9779.
- 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*.
- Elita Lobo, Chirag Agarwal, and Himabindu Lakkaraju. 2025. On the impact of fine-tuning on chain-of-thought reasoning. In *Proceedings of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11679–11698.
- Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, and 1 others. 2025. Tool-sandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183.
- 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.
- 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.
- Akshara Prabhakar, Zuxin Liu, Weiran Yao, Jianguo Zhang, Ming Zhu, Shiyu Wang, Zhiwei Liu, Tulika Awalganekar, Haolin Chen, Thai Hoang, and 1 others. 2025. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiushi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

- Yun Qu, Yuhang Jiang, Boyuan Wang, Yixiu Mao, Cheems Wang, Chang Liu, and Xiangyang Ji. 2025. Latent reward: Llm-empowered credit assignment in episodic reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 20095–20103.
- Qwen Team. 2025. [Qwq-32b: Embracing the power of reinforcement learning](#). Accessed on September 27, 2025.
- Zhuocheng Shen. 2024. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*.
- Guangming Sheng, Chi Zhang, Zilinfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.
- Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, and 1 others. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*.
- Simranjit Singh, Andreas Karatzas, Michael Fore, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. 2024. An llm-tool compiler for fused parallel function calling. *arXiv preprint arXiv:2405.17438*.
- Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Jize Wang, Ma Zerun, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. 2024a. Gta: a benchmark for general tool agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Maolin Wang, Yingyi Zhang, Bowen Yu, Bingguang Hao, Cunyin Peng, Yicheng Chen, Wei Zhou, Jinjie Gu, Chenyi Zhuang, Ruocheng Guo, Wanyu Wang, and Xiangyu Zhao. 2026. Function calling in large language models: Industrial practices, challenges, and future directions. *ACM Comput. Surv.*, 58(9).
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*.
- Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. 2023. Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents. *Advances in Neural Information Processing Systems*, 36:34153–34189.
- Bin Wu, Edgar Meij, and Emine Yilmaz. 2025. A joint optimization framework for enhancing efficiency of tool utilization in llm agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22361–22373.
- xAI. 2025. [Grok 4](#). Accessed on September 27, 2025.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Kang Yang, Guanhong Tao, Xun Chen, and Jun Xu. 2025b. Alleviating the fear of losing alignment in llm fine-tuning. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2152–2170. IEEE.
- Zhaojian Yu, Yilun Zhao, Arman Cohan, and Xiaoping Zhang. 2025. HumanEval pro and MBPP pro: Evaluating large language models on self-invoking code generation task. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13253–13279.
- Xingshan Zeng, Weiwen Liu, Xu Huang, Zezhong Wang, Lingzhi Wang, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruiming Tang, and 1 others. 2025a. Toolace-r: Tool learning with adaptive self-refinement. *arXiv preprint arXiv:2504.01400*.
- Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2025b. Toolace-mt: Non-autoregressive generation for agentic multi-turn interaction. *arXiv preprint arXiv:2508.12685*.
- Ji Zhang, Shihan Wu, Xu Luo, Hao Wu, Lianli Gao, Heng Tao Shen, and Jingkuan Song. 2025a. Inspire: Vision-language-action models with intrinsic spatial reasoning. *arXiv preprint arXiv:2505.13888*.
- Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, and 1 others. 2025b. xlam: A family of large action models to empower ai agent systems. 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 11583–11597.
- Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025c. Nemotron-research-tool-n1: Tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyang Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pages 400–410.

Tianyang Zhong, Zhengliang Liu, Yi Pan, Yutong Zhang, Yifan Zhou, Shizhe Liang, Zihao Wu, Yanjun Lyu, Peng Shu, Xiaowei Yu, and 1 others. 2024. Evaluation of openai o1: Opportunities and challenges of agi. *arXiv preprint arXiv:2409.18486*.

A Dataset

A.1 Data Analysis

Here, we split each turn of multi-turn conversations and count them together with Single-Turn. By using Qwen2.5-Coder-7B-Instruct’s tokenizer for token calculation, the average function-call token length across the entire dataset (including rejections) is 31.07, the average CoT token length is 350.74. In the first data filtering, we obtain 85,014 qualified data and 3,178 hard samples for self-evolving, dropping 1,808 samples.

A.2 Errors in Dataset

As illustrated in Figure 6, the xlam dataset contains an error in the formatting of function calls. Specifically, an extra space appears between the opening bracket and the function name ("[forecast_weather_api(...)"), which causes the Abstract Syntax Tree (AST) parser to fail. This type of issue highlights the importance of performing a base quality check on the dataset to ensure correct syntax and reliable downstream processing. Ensuring proper formatting is essential for both automated function call execution and model evaluation.

B RL Training Details

To further compare the methods, we select the reinforcement learning baseline method GRPO and conduct experiments using both Qwen2.5-Coder-7B-Instruct and Llama-3.2-3B-Instruct. We use the open-source reinforcement learning library Verl (Sheng et al., 2025) for training; the key hyperparameters for training are shown in Table 7.

C More Results

C.1 Full BFCL Single-Turn Results

We show the full single-turn results on the BFCL benchmark, where BalanceSFT attains superior performance with scores of 88.29 and 79.70 on Non-Live and Live sections. The results surpass most

similar scale models and achieve great enhancement on all the categories of live section, which indicates that our model has achieved excellent tool-calling capabilities under complex queries; the evaluation details are shown in Table 9.

C.2 Results on Agentic Evaluation

In addition to single-turn and multi-turn function call capabilities, BFCLv4 further evaluates the model’s agentic capability, specifically, Web Search and Memory. We test our model on the corresponding test sets in Table 10. Even though our base model has almost no agentic capability and no Web Search or Memory-related data is included in the BalanceSFT training, our model still shows improvement in these two evaluation subsets, which indicates that our training and data selection are beneficial to the model’s agentic ability.

C.3 Results on Qwen3

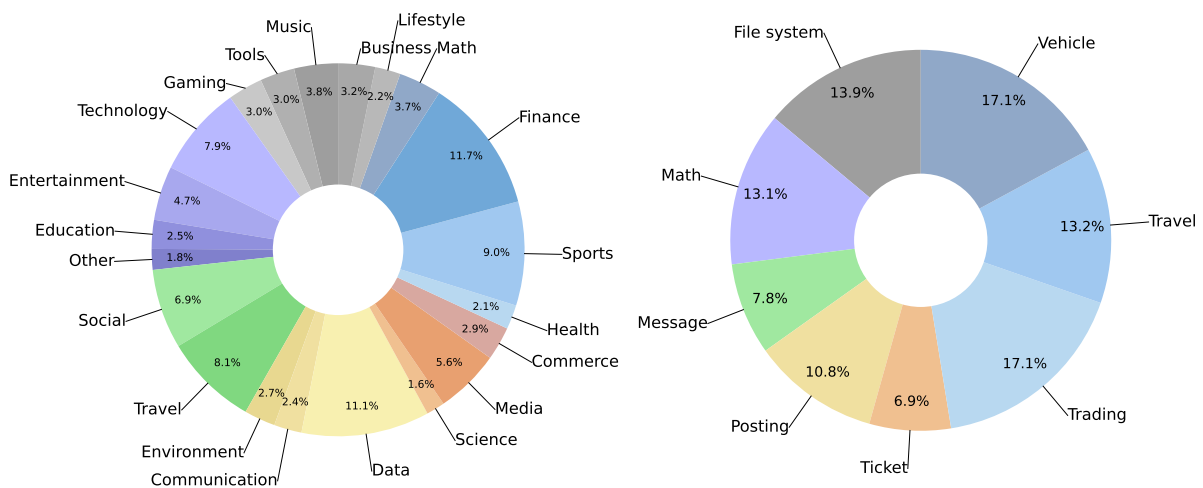
To further verify the effectiveness of BalanceSFT, we conduct experiments by using the more recently open-sourced Qwen3-4B-Instruct-2507. The detailed results are shown in Table 11. Compared with the standard SFT method, the proposed BalanceSFT still performs well on the Qwen3 model.

C.4 Open-source LLM Judge

To ensure our results are not contingent on proprietary models, we conduct experiments using QwQ-32B, a freely available open-source alternative that can be deployed locally as a lightweight evaluator. Table 8 presents a comparison between QwQ-32B and proprietary model judgments: BalanceSFT-7B scores 83.21/45.12 versus 84.00/47.00 on Single-Turn/Multi-Turn tasks respectively. Although QwQ-32B is demonstrably weaker than commercial evaluators like GPT-4o, Gemini, and Claude, the resulting performance metrics remain highly competitive, with differences of only 0.79 points (Single-Turn) and 1.88 points (Multi-Turn). This narrow gap suggests that BalanceSFT’s effectiveness is evaluator-agnostic and that our method produces robust improvements that are consistently recognized across different judgment paradigms, including those based on more accessible open-source models.

D Details of Hard Data Re-sampling Strategy

In the data refinement pipeline, we leverage QwQ-32B to conduct hierarchical identification and cor-



(a) The proportion of various APIs in xlam-function-calling-60k

(b) The proportion of various APIs in Open-Agentic-tool-use

Figure 5: Distribution of API categories in xlam-function-calling-60k and Open-Agentic-tool-use. Different colors represent different API categories.

Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
Global Batch Size	1024	Max Prompt Length	8192	Max Response Length	20480
Learning Rate	1e-6	Temperature	0.7	Epoch Number	5
KL Coefficient	1e-3	Entropy Coefficient	1e-3	Rollout Number	8

Table 7: The detailed hyperparameters used for RL training.

Judge	Single-Turn	Multi-Turn
QwQ-32B	83.21	45.12
proprietary models	84.00	47.00

Table 8: Performance across different judges.

rection across multiple dimensions of data quality. To support this process, we design a set of structured prompts that decompose the refinement procedure into several stages: response identification (Figure 7), query and tool consistency verification (Figure 8), Chain-of-Thought validation (Figure 9), function and parameter verification (Figure 10), and format validation (Figure 11). To further illustrate the overall process, we present a case study of function call generation with Chain-of-Thought reasoning in Figure 12.

Models	Non-Live					Live				
	Overall	Simple	Multiple	Parallel	Parallel Multiple	Overall	Simple	Multiple	Parallel	Parallel Multiple
Close-Source Models										
GPT-4o-2024-11-20	83.88	76.50	91.00	90.00	78.00	70.54	70.54	70.75	62.50	66.67
Gemini-2.5-Pro	85.04	68.67	91.00	91.50	89.00	63.95	75.58	60.40	81.25	83.33
o3-2025-04-16	39.98	71.42	88.50	0.00	0.00	66.03	69.77	67.62	0.00	0.00
Amazon-Nova-Pro-v1:0	85.25	71.50	93.50	92.00	84.00	78.31	80.62	77.87	87.50	66.67
Grok-4-0709	85.21	73.33	92.50	89.50	85.50	74.39	77.91	73.69	68.75	70.83
Open-Source Models										
Moonshotai-Kimi-K2-Inst	84.02	79.08	93.00	76.00	88.00	77.57	86.05	75.78	81.25	62.50
DeepSeek-R1-0528	75.73	60.42	96.50	84.50	61.50	80.90	84.50	80.53	81.25	58.33
Qwen3-235B-A22B-Inst-2507	90.12	80.50	95.00	95.50	89.50	76.61	83.72	75.02	75.00	70.83
Llama-4-Maverick	88.15	76.08	95.00	92.50	89.00	73.65	84.50	71.04	75.00	70.83
Qwen3-8B	88.60	78.92	95.00	91.50	89.00	77.79	80.23	77.21	81.25	75.00
BitAgent-8B	87.33	76.33	95.00	93.00	85.00	76.09	78.29	75.59	87.50	66.67
watt-tool-8B	87.54	76.17	94.50	94.00	85.50	75.87	77.52	75.59	87.50	62.50
Base Models										
Qwen2.5-Coder-7B-Instruct	83.88	76.00	95.00	81.00	83.50	69.75	74.81	75.31	68.75	62.50
Llama-3.2-3B-Instruct	81.94	71.25	92.00	87.00	77.50	57.74	63.57	57.26	25.00	37.50
Ours										
BalanceSFT-7B	88.29	78.17	94.00	91.50	89.50	79.70	82.56	77.87	87.50	83.33
BalanceSFT-3B	83.83	73.33	90.50	88.00	83.50	74.14	74.42	71.70	75.00	79.17

Table 9: The evaluation details on BFCL benchmark (last updated August 26, 2025), with all metrics calculated using the official script.

Models	Web Search			Memory			
	Overall	Base	No Snippet	Overall	KV	Vector	Recursive Sum
Qwen2.5-Coder-7B-Instruct	4.50	4.00	5.00	2.37	0.00	0.00	7.10
BalanceSFT-7B	7.00	9.00	5.00	8.17	0.00	0.00	24.52

Table 10: Performance of the model on agentic evaluation tasks.

Models	Parameter Counts	Multi-Turn					Single-Turn		
		Overall	Base	Miss Func	Miss Param	Long Context	Overall	Non-Live	Live
Qwen3-4B-Inst-2507	4B	15.75	19.00	15.50	12.50	16.00	78.19	86.35	70.02
w/ SFT	4B	41.88	47.00	50.50	36.00	34.00	82.20	85.15	79.25
w/ BalanceSFT	4B	47.12	53.00	46.00	41.50	48.00	83.88	87.48	80.28

Table 11: Performance of Qwen3-4B-Instruct-2507 on BFCL.

Error of xlam Dataset

Query: What are the daily weather forecasts for Chicago and Toronto for the next 7 days?

Tools:

```
[
  {
    "name": "get_forecastdata_by_lat_lon",
    "description": "Fetches a 14-day weather forecast for a given location specified by latitude and longitude.",
    "parameters": {"lat": {"type": "int", "default": "51.5"}, "lon": {"type": "int", "default": "-0.6"}, "lang": {"type": "str", "default": "en"}}
  },
  {
    "name": "location_info",
    "description": "Retrieve metadata for a specified location.",
    "parameters": {"location": {"type": "str", "default": "101275339"}}
  },
  {
    "name": "forecast_weather_api",
    "description": "Fetches weather forecast and alerts from API.",
    "parameters": {"q": {"type": "str", "default": "London"}, "days": {"type": "int", "default": "3"}, "dt": {"type": "str", "default": ""}, "lang": {"type": "str", "default": ""}}
  },
  {
    "name": "daily",
    "description": "Fetches daily weather forecast for a given location.",
    "parameters": {"location": {"type": "str", "default": "102643743"}, "alt": {"type": "int", "default": "0"}, "dataset": {"type": "str", "default": "full"}, "tempunit": {"type": "str", "default": "C"}, "windunit": {"type": "str", "default": "MS"}, "periods": {"type": "int", "default": "8"}}
  }
]
```

Answer:

```
[
  {
    "name": "forecast_weather_api",
    "arguments": {
      "q": "Chicago",
      "days": 7
    }
  },
  {
    "name": "forecast_weather_api",
    "arguments": {
      "q": "Toronto",
      "days": 7
    }
  }
]
```

Figure 6: An Example of an Error in the Xlam Dataset.

Response Identification

Workflow
 1) Determine whether the Reference Answer is a function_tool call or a response statement. The function call may not be in the standard function_tool call format. If it is a function_tool call, output <judge>True</judge>; if it is a response, output <judge>False</judge>.
Hint: Common function_tool call formats include the following, where function_name is generally from candidate_function_tools:
 [func_name1(params_name1=params_value1, params_name2=params_value2...),
 func_name2(params_name3=params_value3, params_name4=params_value4...)]

Output format: <think>thought process</think><judge>True/False</judge>

Input
 Reference Answer: <refANS>

Figure 7: Response Identification Prompt.

Query and Tool Identification

Workflow
 Judge whether the User Query and Candidate Function Tools meet the following requirements:
 1) Determine whether the parameter values of the function call can be analyzed from the User Query and the function name can be analyzed from the Candidate Function Tools. If parameter values and function names can be analyzed, output <judge>True</judge>; otherwise, output <judge>False</judge>.

Output format: <think>thought process</think><judge>True/False</judge>

Input
 User Query: <query>
 Candidate Function Tools: <tools>

Figure 8: Query and Tool Identification Prompt.

CoT Identification

Workflow
 Judge whether the Chain-of-Thought leads to the Reference Function Call by evaluating the following requirements:
 1) Determine whether the Chain-of-Thought starts from a proper position.
 2) Determine whether every step tightly follows the above step and makes correct inference.
 3) Determine whether the last step of reasoning points to the correct answer.
 If the Chain-of-Thought meets all the requirements, output <judge>True</judge>; otherwise, output <judge>False</judge>.

Output format: <think>thought process</think><judge>True/False</judge>

Input
 Chain-of-Thought: <CoT process>
 Reference Function Call: <refFC>

Figure 9: Chain-of-Thought Identification Prompt.

Function and Parameter Identification

Workflow

Judge whether the function names and parameters in the Reference Function Call meet the following requirements:

1) Determine whether the function names and parameter values of the function call are correct. If the function names and parameters are valid, output `<judge>True</judge>`; otherwise, output `<judge>False</judge>`.

2) When the function names and parameters requirements are met, output the new function call result according to the Reference Function Call. When the function names and parameters requirements are not met, modify the Reference Function Call format to obtain a new function call result that meets the name and parameters requirements.

Output format:

`<think>thought process</think><judge>True/False</judge>`

`<NewFC>`

`[func_name1(params_name1=params_value1, params_name2=params_value2...),
func_name2(params_name3=params_value3, params_name4=params_value4...)]`

`</NewFC>`

Input

User query: `<query>`

Candidate function tools: `<tools>`

Reference Function Call: `<refFC>`

Figure 10: Function and Parameter Identification Prompt.

Format Identification

Workflow

Judge whether the answer meets the following requirements:

1) The output function_tool format must satisfy the format:

```
[func_name1(params_name1=params_value1, params_name2=params_value2...),  
func_name2(params_name3=params_value3, params_name4=params_value4...)]
```

Note: The parameter name param_name should not be enclosed in " or ', for example:

Incorrect:

- [getPrivacyViolationRisk(data="paramvalue1", purpose="paramvalue2")]
- [getPrivacyViolationRisk(data='paramvalue1', purpose='paramvalue2')]

The paramValue parameter value, based on its value type, needs to be enclosed in quotes if it is a string.

2) When the output meets the format requirements, output <judge>True</judge>; otherwise, output <judge>False</judge>.

3) When the format requirements are met, output the new function call result according to the Reference Function Call. When the format requirements are not met, modify the Reference Function Call format to obtain a new function call result that meets the format requirements. Do not add or modify parameters and parameter values; only modify the output format.

Output format:

```
<think>thought process</think><judge>True/False</judge>
```

```
<NewFC>
```

```
[func_name1(params_name1=params_value1, params_name2=params_value2...),  
func_name2(params_name3=params_value3, params_name4=params_value4...)]
```

```
</NewFC>
```

Input

User query: <query>

Candidate function tools: <tools>

Reference Function Call: <refFC>

Figure 11: Format Identification Prompt.

Case Study: Ice Hockey Match Schedule Query

System: You are an expert in composing functions. You are given a question and a set of possible functions. Based on the question, you will need to make one or more function/tool calls to achieve the purpose. If none of the functions can be used, point it out. If the given question lacks the parameters required by the function, also point it out.

You should only return the function calls in your response. If you decide to invoke any of the function(s), you MUST put it in the format of [func_name1(params_name1=params_value1, params_name2=params_value2...), func_name2(params)]. You SHOULD NOT include any other text in the response.

Here is a list of functions in JSON format that you can invoke:

```
[{"name": "matchschedules",
  "description": "Fetches the schedules of ice hockey matches for
  a specified date, including match timings, teams, and other
  relevant information.",
  "parameters": {
    "day": {"description": "The day of the month for which you
    want to retrieve the schedules (1-31).", "type": "int",
    "default": 28},
    "year": {"description": "The year for which you want to
    retrieve the schedules (e.g., 2022).", "type": "int",
    "default": 2022},
    "month": {"description": "The month for which you want to
    retrieve the schedules (1-12).", "type": "int",
    "default": 11}}}]
```

User: I need the schedules of matches happening on February 28, 2024.

Assistant: <think> Okay, let's see. The user is asking for the match schedules on February 28, 2024. The function provided is called matchschedules. The parameters it takes are day, month, and year, all integers. The defaults are day 28, month 11, year 2022. The user's date is February 28, 2024. So February is month 2. The day is 28, which is valid. The year is 2024. Since the function parameters allow specifying these, I need to set each parameter to the user's requested values. The defaults are for November 28, 2022, which isn't what the user wants. So I should override the defaults by setting day=28, month=2, year=2024. That should fetch the correct schedules. The function seems appropriate here. No missing parameters, so the function call is possible. </think>
[matchschedules(day=28, month=2, year=2024)]

Figure 12: Example of Function Call Generation with Chain-of-Thought Reasoning.