

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

Anonymous ACL submission

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 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 on par with state-of-the-art giants like GPT-5.2 and DeepSeek-V3.2. 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., 2025; Singh et al., 2024). Supervised Fine-Tuning (SFT) has emerged as the standard paradigm to instill this capability, typically by training models on

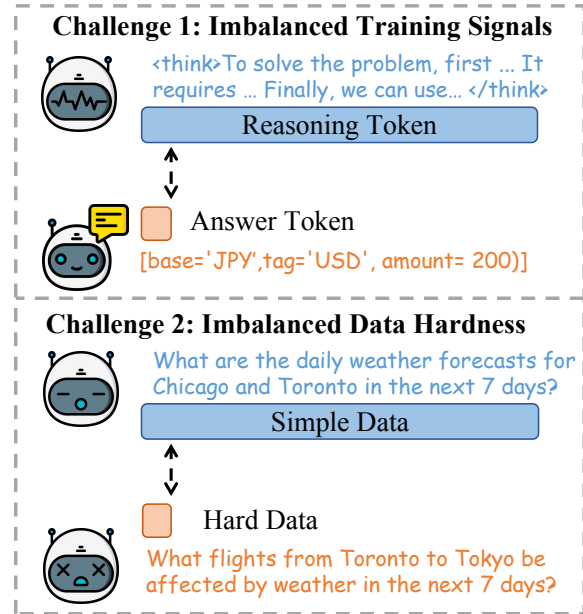


Figure 1: **Two challenges in LLM Function Calling.** (a) **Imbalanced Training Signals:** The lengthy Chain-of-Thought (CoT) tokens dominate the learning signal, overshadowing the concise but critical function call. (b) **Imbalanced Data Hardness:** The training dataset is dominated by simple examples,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

demonstrations that combine a Chain-of-Thought (CoT) reasoning process with the final function call (Lightman et al., 2023; Guo et al., 2025).

Despite the remarkable progress, conventional SFT approaches suffer from two inherent and critical imbalances that severely limit its 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 Imbalanced Training Signal causes the model to be optimized primarily for generating plausible reasoning, often at the expense of precision in the

059 final tool execution. On the other hand, the data
060 distribution is characterized by an Imbalanced Data
061 Hardness (Li et al., 2023; Zhang et al., 2024). Ex-
062 isting training datasets are naturally dominated by
063 simple examples, while complex and challenging
064 cases that are crucial for robust performance are
065 scarce. This scarcity prevents the model from effec-
066 tively learning to handle edge cases and intricate
067 scenarios (Zeng et al., 2025b).

068 To tackle the dual challenges, we propose **Bal-**
069 **anced Supervised Fine-Tuning (BalanceSFT)**, a
070 novel framework designed to recalibrate the SFT
071 process. BalanceSFT incorporates two simple yet
072 effective components: 1) A Self-adjusted Signal
073 Balancing (**SSB**) loss function that introduces a
074 learnable hyperparameter to dynamically balance
075 the contribution of CoT tokens and function call
076 tokens to the total loss, ensuring that the model
077 does not overlook the critical execution step; 2)
078 A Hard Data Re-sampling (**HDR**) strategy that es-
079 tablishes a feedback loop to generate new, high-
080 quality complex data guided by model errors. Ex-
081 tensive experiments validate the effectiveness of
082 our BalanceSFT framework. The results show
083 that a 7B model trained with BalanceSFT achieves
084 tool-calling performance competitive with state-of-
085 the-art giants like GPT-5.2 (OpenAI, 2025) and
086 Deepseek-V3.2 (Liu et al., 2025) on the compre-
087 hensive Berkeley Function Calling Leaderboard
088 (BFCL) (Patil et al., 2024) and other specialized
089 benchmarks (Chen et al., 2025a; Li et al., 2023).

090 Our main contributions are threefold:

- 091 • We identify two critical yet overlooked limi-
092 tations in standard SFT for tool learning: the
093 imbalance in training signals between lengthy
094 reasoning and concise function calls, and the
095 imbalance in data hardness.
- 096 • We propose BalanceSFT, a novel framework
097 that introduces a Self-adjusted Signal Balanc-
098 ing (SSB) loss and a Hard Data Re-sampling
099 (HDR) strategy to calibrate the learning focus.
- 100 • We train a 7B model with the proposed Bal-
101 anceSFT framework, achieving tool-calling
102 performance comparable to the state-of-the-
103 art model like GPT-5.2 (OpenAI, 2025) and
104 Deepseek-V3.2 (Liu et al., 2025).

105 2 Related Work

106 **Function Calling for LLMs.** The ability of LLMs
107 to interact with external tools and APIs through

108 function calling has emerged as a crucial aspect of
109 their practical utility (Wang et al., 2024b; Singh
110 et al., 2024; Patil et al., 2024). This capability al-
111 lows LLMs to transcend the limitations of solely
112 processing and generating text, enabling them to
113 ground their responses in real-world data and au-
114 tomate complex tasks (Wang et al., 2024a; Ka-
115 vathekar et al., 2025; Zhang et al., 2025a). Early
116 explorations in this area focused on enabling LLMs
117 to understand descriptions of functions and gener-
118 ate the necessary calls with appropriate param-
119 eters (Wang et al., 2023; Zheng et al., 2023). By
120 training LLMs on datasets specifically curated for
121 function calling scenarios, these methods aim to
122 instill a deeper understanding of function seman-
123 tics and parameter requirements (Prabhakar et al.,
124 2025; Zeng et al., 2025b). The quality and diversity
125 of these fine-tuning datasets play a critical role in
126 the resulting performance of the LLMs in function
127 calling tasks. RL has also been explored as a means
128 to optimize LLMs for function calling (Zhang et al.,
129 2025b; Qian et al., 2025). RL-based approaches
130 often involve defining reward functions that incen-
131 tivize the generation of correct and executable func-
132 tion calls. While RL offers the potential to directly
133 optimize for task success, it still suffers from sparse
134 rewards and the lack of effective data (Qu et al.,
135 2025; Zeng et al., 2025a).

136 **Improving Reasoning Capability via SFT.** The
137 integration of CoT with function calling mech-
138 anisms holds substantial promise for enhancing
139 the accuracy and reliability of tool-use (Lu et al.,
140 2024; Kokane et al.; Chen et al., 2024). By reason-
141 ing through the user’s request and identifying the
142 most appropriate course of action, including the
143 selection of relevant tools and their parameters, the
144 model can make more informed decisions about
145 function invocation (Li, 2025; Shen, 2024). The
146 explicit reasoning process provided by CoT can
147 also improve the transparency and interpretability
148 of the model’s actions (Wu et al., 2025; Dong et al.,
149 2025). However, the integration of CoT into the
150 training process for function calling introduces a
151 unique challenge related to the design of the loss
152 function (Yang et al., 2025b). Traditional training
153 methodologies, primarily SFT, typically treat the
154 entire generated sequence (including the reasoning
155 process and the final function call) uniformly when
156 calculating the loss (Lobo et al., 2025). This can
157 lead to an imbalance, where the potentially lengthy
158 sequences of reasoning steps dominate the loss cal-
159 culation, 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

This section introduces our Balanced Supervised Fine-Tuning (BalanceSFT) framework, designed to address imbalanced training signals and 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 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 (Team, 2024) based on the open-sourced function call dataset xlam-function-calling-60k (Zhang et al., 2024), which exhibits a significant token imbalance: Chain-of-Thought reasoning segments contain approximately 10 times more tokens (mean=350.74) 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.

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 call. Note the approximately 10x higher token count for the CoT compared to the final execution.

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 expressed as a weighted sum:

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

We decompose the loss L_{SFT} as following:

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

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 parameter 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.

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.

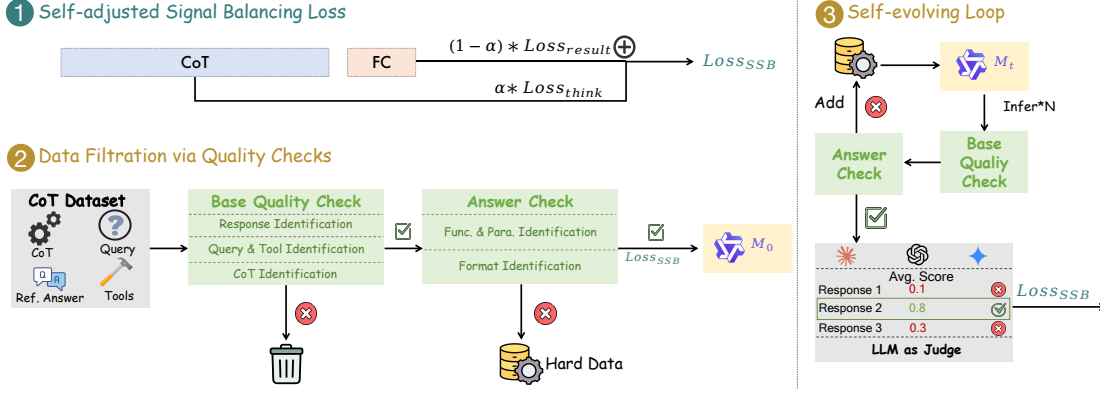


Figure 2: **Overview of the BalanceSFT framework.** It starts with a standard function call dataset, which is refined through a Base Quality Check and Answer Check to create initial training data and identify hard data. The model is first initialized via a Cold Start using the **Self-adjusted Signal 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.

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 (Claude)) via majority voting. The scoring function $\text{Score}(r_j)$

aggregates their judgments:

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

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 Setting

Dataset. As summarized in Table 4, our training data consists of two components: xlam-function-calling-60k (Zhang et al., 2024) and Open-Agentic-tool-use (Aworld team, inclusionAI, 2025). The xlam-function-calling-60k dataset includes 60,000 samples spanning 3,673 APIs across 21 categories and primarily targets single-turn tool use to strengthen foundational capabilities. In contrast, Open-Agentic-tool-use contains 30,000 samples covering 129 APIs across 8 categories and emphasizes multi-turn interactions to support complex, agentic scenarios. Figure 5 illustrates the distribution of the diverse and comprehensive API resources, with additional dataset analysis provided in Appendix A.

Benchmarks. To assess our method, we adopt a highly representative benchmark, the Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2024). We evaluate our models on BFCLv3, which covers both single-turn and multi-turn function-calling scenarios, and report performance on all subsets 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 the BalanceSFT framework, we adopt Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) as our primary base model. In addition, to further validate the robustness and transferability of our approach, we conduct supplementary experiments on Llama-3.2-3B-Instruct (Touvron et al., 2023) and Qwen3-4B-Instruct-2507 (Yang et al., 2025a).

Implementations. Our implementation is built upon the LLaMA Factory training framework (Zheng et al., 2024). We adopt consistent hyperparameters across all experiments, including a batch size of 512, a learning rate of 4×10^{-5} , and a warm-up ratio of 0.05. The SSB loss hyperparameter α is initialized to 0.7 and treated as a learnable parameter. During inference, we perform three sampling runs with a temperature of 0.7. For the LLM-as-judge stage in HDR, we employ Gemini-2.5-Pro (Comanici et al., 2025), GPT-4o (Hurst et al., 2024), and Claude-3.5 Sonnet (Claude) as judges, using a unified temperature of 0.7, a threshold $\tau = 0.5$, and a scoring range of $[0, 1]$. The maximum number of iterations T_{\max} is set to 10. All training experiments are conducted on 8 NVIDIA

A100 GPUs. To further assess the generalizability of our framework, we additionally report results using an open-source LLM judge, QwQ-32B (Team, 2024), 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 (Team, 2024) 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 consistently 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

Results on BFCLv3. The performance of BalanceSFT-7B (built upon Qwen2.5-Coder-7B-Instruct) on the BFCL benchmark is summarized in Table 2. BalanceSFT-7B achieves 74.02 Overall performance and establishes a new state-of-the-art results for 7B parameter models. The Multi-Turn subset evaluates a model’s ability to handle complex, continuous conversations. BalanceSFT-7B shows exceptional gains in multi-turn scenarios (+48.37), particularly for missing parameters (+46.00) and long context (+53.00). The Single-Turn subset focuses on isolated prompts under Non-Live and Live settings. As reported in Table 2, BalanceSFT-7B maintains its competitive edge by achieving the subset Overall score of 84.90 (+8.08 over base model). Although closed-source models like GPT-4o (65.64) demonstrate competitive performance, our results suggest that BalanceSFT can enable smaller open-source models to achieve comparable or superior function calling capabilities. These findings demonstrate that BalanceSFT-7B can accurately analyze system status and remains highly sensitive to key functions and parameters within context. To further validate the generalizability of our framework, we apply BalanceSFT to Qwen3-4B-Instruct-2507 and report the results in Appendix C.2.

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

Model	Parameter Counts	Single-Turn			Multi-Turn					Overall
		Non-Live	Live	Subset Overall	Base	Miss Func	Miss Param	Long Context	Subset Overall	
Close Source Model										
Gemini-2.5-Pro	-	85.04	63.95	74.50	25.50	26.00	24.50	24.00	25.00	58.00
Gemini-2.5-Flash	-	84.96	74.39	79.68	41.50	36.00	32.00	35.50	36.25	65.20
Grok-4-1-fast-reasoning	-	88.27	78.46	83.37	70.50	59.50	43.00	62.50	58.88	75.20
Grok-4-1-fast-non-reasoning	-	88.13	77.94	83.04	58.00	39.50	37.50	52.00	46.75	70.94
GPT-5.2-2025-12-11	-	81.85	70.39	76.12	36.50	18.00	27.50	30.50	28.13	60.12
GPT-4o-2024-11-20	-	83.88	70.54	77.21	55.50	34.50	29.00	51.00	42.50	65.64
Open Source Model										
Kimi-K2-Instruct	1043B	81.60	78.68	80.14	62.00	41.00	44.50	55.00	50.63	70.30
DeepSeek-V3.2-Exp	671B	85.52	76.02	80.77	55.00	49.00	27.00	48.50	44.88	68.81
Llama-4-Maverick	400B	88.65	73.65	81.15	27.00	22.00	14.00	18.00	20.25	60.85
GLM-4.6	355B	87.56	80.90	84.23	74.50	68.00	63.00	66.50	68.00	78.82
Qwen3-235B-A22B-Instruct	235B	90.33	78.68	84.51	54.00	42.50	31.50	50.50	44.63	71.21
Qwen3-32B	32B	88.77	82.01	85.39	56.00	52.50	40.00	43.00	47.88	72.88
Qwen3-30B-A3B-Thinking	30B	85.77	77.94	81.86	43.50	10.50	25.00	41.00	30.00	64.57
ToolACE-2-8B	8B	87.10	77.42	82.26	49.00	28.00	30.50	46.00	38.38	67.63
ToolACE-MT	8B	84.94	71.52	78.23	57.50	31.50	34.00	38.00	40.25	65.57
xLAM-2-3b-fc-r	3B	82.96	62.92	72.94	71.50	59.00	57.50	45.50	58.38	68.09
Qwen2.5-Coder-7B-Inst	7B	83.88	69.75	76.82	6.00	3.50	2.00	4.00	3.88	52.50
BalanceSFT-7B	7B	88.60	81.20	84.90	51.00	53.00	48.00	57.00	52.25	74.02
Δ		+4.72	+11.45	+8.08	+45.00	+49.50	+46.00	+53.00	+48.37	+21.52

Table 2: **Performance on BFCLv3** (last updated on 2025-12-16). All metrics are calculated using the official script and reported in terms of Accuracy (%).

Models	Non-Live	Live	Overall
Qwen2.5-Coder-7B-Inst	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 single-turn subset of BFCL. The two models are trained on Qwen2.5-Coder-7B-Instruct with the same training hyperparameters.

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

Table 4: Statistics of the xlam-function-calling-60k and Open-Agentic-tool-use dataset.

(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. To further demonstrate the transferability of our model’s function-calling capabilities

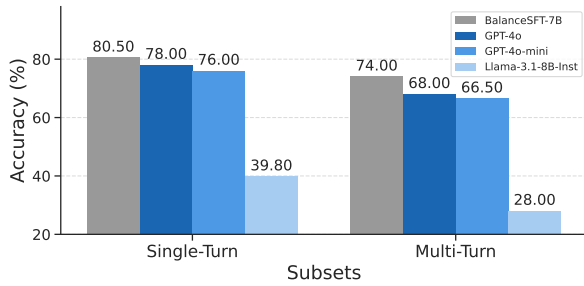
in concrete scenarios, we additionally present its performance in the Search and Memory scenarios of BFCLv4 (Patil et al., 2024) in Appendix C.1.

4.4 Ablation Study

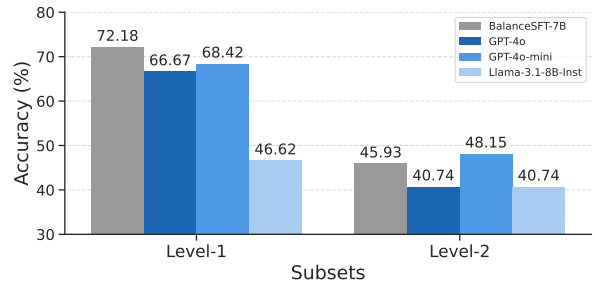
In the ablation study, all experiments are conducted under identical settings to ensure fair comparisons. We use basic SFT as the baseline to isolate the contribution of each component. To assess generalization, we evaluate our framework on Qwen2.5-Coder-Instruct and Llama3.2-3B-Instruct.

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-Instruct, the SFT trained model achieves 81.92 and 40.17 in Single-Turn and Multi-Turn, respectively. Individual activation of SSB yields improvements up to 82.25 (Single-Turn) and 42.72 (Multi-Turn). In contrast, applying SFT to HDR resampled data provides larger gains, boosting Single-Turn by 2.00 and dramatically increasing Multi-Turn by 6.08, suggesting that rare edge samples are critical. Crucially, the combination of both SSB and HDR resulted in synergistic gains, achieving the highest performance of 84.90 (Single-Turn) and 52.25 (Multi-Turn). These



(a) Performance on ACEBench.



(b) Performance on APIBank.

Figure 3: Performance comparison on ACEBench and APIBank.

Models	Modules		Single Turn	Multi Turn
	SSB	HDR		
Qwen2.5-Coder-Inst+SFT	×	×	81.92	40.17
	✓	×	82.25	42.72
	×	✓	83.92	46.25
	✓	✓	84.90	52.25
Llama-3.2-3B-Inst+SFT	×	×	75.25	35.28
	✓	×	77.82	36.62
	×	✓	79.28	37.75
	✓	✓	80.97	38.26

Table 5: Performance comparison of Qwen2.5-Coder-Instruct and Llama-3.2-3B-Instruct, across different components of modules (SSB and HDR) in Single-Turn and Multi-Turn on BFCL.

findings underscore the substantial contributions of both modules to enhancing the model’s 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 for optimizing tool-use capabilities across different model architectures, i.e., yielding more hard 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) method on BFCL shown in Table 4a, the 7B and 3B models are trained upon Qwen2.5-Coder-7B-Instruct and Llama-3.2-3B-Instruct (Shao et al., 2024). We use the hyperparameter settings and reward design from the official GRPO baseline de-

scribed in **Appendix B**. Our proposed BalanceSFT method demonstrates superior performance over GRPO across multiple model architectures and task settings. For Qwen2.5-Coder-7B-Instruct, BalanceSFT achieves a higher score in the Single-Turn task (84.90 vs. 84.00), and outperforms GRPO in the Multi-Turn task (52.25 vs. 47.85). A similar trend is observed with Llama-3.2-3B-Instruct, where BalanceSFT outperforms GRPO in both tasks, marginally in Single-Turn (80.99 vs. 78.82) and more notably in Multi-Turn (38.26 vs. 35.12).

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 HDR Self-Evolving Loop. As shown in Figure 4c, 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 sample complex edge samples, driving the model performance toward a high-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 enhances its accuracy, robustness across complex, multi-step tasks.

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 4b, HumanEval and MBPP (including their plus variants) (Austin et al., 2021;

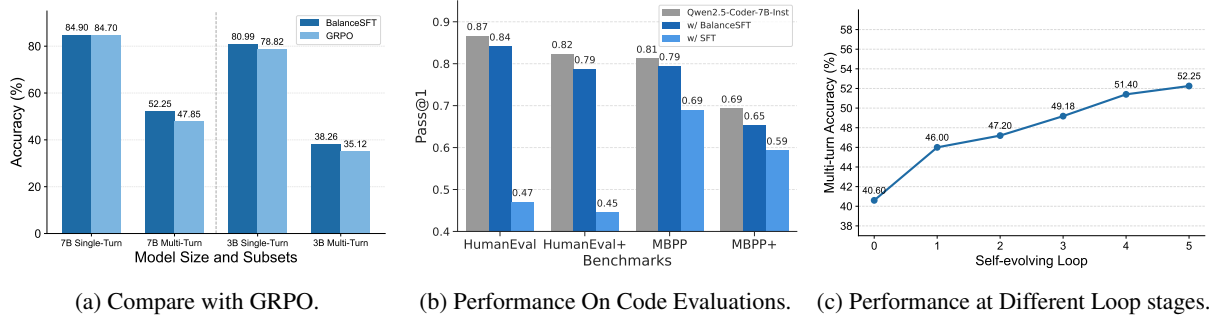


Figure 4: **More experiments.** (a) Performance of two series models trained by GRPO and BalanceSFT on BFCL. (b) Performance of BalanceSFT and SFT models on HumanEval and MBPP (including HumanEval+ and MBPP+) compared with that of the code pre-trained model (Qwen2.5-Coder-7B-Instruct). (c) Multi-Turn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

Chen et al., 2021; Yu et al., 2024). The results, presented in Table 4c, 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-Instruct model. For instance, on HumanEval, BalanceSFT achieves 0.841 compared to the base model’s 0.866, and on MBPP, it reaches 0.794 versus 0.812. 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 0.866 to 0.470 on HumanEval and from 0.823 to 0.445 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.

Initial α	Final α	Single-Turn	Multi-Turn
0.5	0.68	84.12	50.97
0.6	0.68	84.49	51.28
0.7	0.69	84.90	52.25
0.8	0.70	84.84	52.21
0.9	0.72	84.12	51.12

Table 6: Performance and α convergence across different initializations.

Sensitivity of Hyperparameter. To assess the robustness of BalanceSFT to hyperparameter initialization, we conduct sensitivity analysis by varying the initial α value from 0.5 to 0.9, as shown in

Table 6. Notably, 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. Performance remains consistently strong across initializations, with Single-Turn scores ranging from 84.12 to 84.90 and Multi-Turn scores from 50.97 to 52.25. These results indicate that BalanceSFT is relatively insensitive to initial hyperparameter selection, as the adaptive balancing mechanism effectively guides α toward similar optimal values regardless of initialization, ensuring stable training outcomes. Our results in Appendix C.3 further indicate that the learnable α can effectively compensate for inherent data imbalances without manual intervention.

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 competitive with state-of-the-art giants like GPT-5.2, Gemini-2.5-Pro and DeepSeek-V3.2 on BFCL and other benchmarks. Our work provides a simple yet effective pathway toward more robust and generalizable LLM-based tool use.

565
566
567
568
569
570
571
572
573
574
575

576

577
578
579
580
581

582
583
584
585

586
587
588
589
590

591
592

593
594
595
596

597
598
599
600
601

602
603
604
605
606
607

608
609
610
611
612
613

614
615

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 bias from the judge models.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, 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*.

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*.

Aworld team, inclusionAI. 2025. [Open-agentic-tool-use](#).

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*.

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 learning? *arXiv e-prints*, pages arXiv–2501.

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. 616
617
618

Claude. The claude 3 model family: Opus, sonnet, haiku. 619
620

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*. 621
622
623
624
625
626
627

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*. 628
629
630
631
632
633

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*. 634
635
636
637
638
639

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*. 640
641
642
643
644

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*. 645
646
647
648
649

Yiming Ju, Ziyi Ni, Xingrun Xing, Zhixiong Zeng, Siqi Fan, Zheng Zhang, and 1 others. 2024. Mitigating training imbalance in llm fine-tuning via selective parameter merging. *arXiv preprint arXiv:2410.03743*. 650
651
652
653

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. *arXiv preprint arXiv:2504.19277*. 654
655
656
657
658

Shirley Kokane, Ming Zhu, Tulika Manoj Awalgaonkar, Jianguo Zhang, Akshara Prabhakar, Thai Quoc Hoang, Zuxin Liu, Rithesh RN, Liangwei Yang, Weiran Yao, and 1 others. Toolscan: A benchmark for characterizing errors in tool-use llms. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*. 659
660
661
662
663
664
665

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*. 666
667
668
669
670
671
672

673	Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song,	Yun Qu, Yuhang Jiang, Boyuan Wang, Yixiu Mao,	728
674	Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang,	Cheems Wang, Chang Liu, and Xiangyang Ji. 2025.	729
675	and Yongbin Li. 2023. Api-bank: A comprehensive	Latent reward: Llm-empowered credit assignment	730
676	benchmark for tool-augmented llms. <i>arXiv preprint</i>	in episodic reinforcement learning. In <i>Proceedings</i>	731
677	<i>arXiv:2304.08244</i> .	of the <i>AAAI Conference on Artificial Intelligence</i> ,	732
		volume 39, pages 20095–20103.	733
678	Xinzhe Li. 2025. A review of prominent paradigms for	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu,	734
679	llm-based agents: Tool use, planning (including rag),	Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan	735
680	and feedback learning. In <i>Proceedings of the 31st</i>	Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseek-	736
681	<i>International Conference on Computational Linguistics</i> ,	math: Pushing the limits of mathematical reason-	737
682	pages 9760–9779.	ing in open language models. <i>arXiv preprint</i>	738
683	Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harri-	<i>arXiv:2402.03300</i> .	739
684	son Edwards, Bowen Baker, Teddy Lee, Jan Leike,	Zhuocheng Shen. 2024. Llm with tools: A survey.	740
685	John Schulman, Ilya Sutskever, and Karl Cobbe.	<i>arXiv preprint arXiv:2409.18807</i> .	741
686	2023. Let’s verify step by step. In <i>The Twelfth Inter-</i>	Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin	742
687	<i>national Conference on Learning Representations</i> .	Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin	743
688	Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingx-	Lin, and Chuan Wu. 2024. Hybridflow: A flexible	744
689	uan Wang, Bingzheng Xu, Bochao Wu, Bowei	and efficient rlhf framework. <i>arXiv preprint arXiv:</i>	745
690	Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025.	<i>2409.19256</i> .	746
691	Deepseek-v3. 2: Pushing the frontier of open large	Simranjit Singh, Andreas Karatzas, Michael Fore, Irak-	747
692	language models. <i>arXiv preprint arXiv:2512.02556</i> .	lis Anagnostopoulos, and Dimitrios Stamoulis. 2024.	748
693	Elita Lobo, Chirag Agarwal, and Himabindu Lakkaraju.	An llm-tool compiler for fused parallel function call-	749
694	2025. On the impact of fine-tuning on chain-of-	ing. <i>arXiv preprint arXiv:2405.17438</i> .	750
695	thought reasoning. In <i>Proceedings of the 2025 Con-</i>	Qwen Team. 2024. Qwq: Reflect deeply on the bound-	751
696	<i>ference of the Nations of the Americas Chapter of the</i>	aries of the unknown.	752
697	<i>Association for Computational Linguistics: Human</i>	Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier	753
698	<i>Language Technologies (Volume 1: Long Papers)</i> ,	Martinet, Marie-Anne Lachaux, Timothée Lacroix,	754
699	pages 11679–11698.	Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal	755
700	Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Au-	Azhar, and 1 others. 2023. Llama: Open and effi-	756
701	mayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma,	cient foundation language models. <i>arXiv preprint</i>	757
702	Mengyu Li, Guoli Yin, and 1 others. 2024. Toolsand-	<i>arXiv:2302.13971</i> .	758
703	box: A stateful, conversational, interactive evalua-	Jize Wang, Ma Zerun, Yining Li, Songyang Zhang,	759
704	tion benchmark for llm tool use capabilities. <i>arXiv</i>	Cailian Chen, Kai Chen, and Xinyi Le. 2024a. Gta:	760
705	<i>preprint arXiv:2408.04682</i> .	a benchmark for general tool agents. In <i>The Thirty-</i>	761
706	OpenAI. 2025. Gpt-5 system card. Technical report,	<i>eight Conference on Neural Information Processing</i>	762
707	OpenAI.	<i>Systems Datasets and Benchmarks Track</i> .	763
708	Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida,	Maolin Wang, Yingyi Zhang, Cunyin Peng, Yicheng	764
709	Carroll Wainwright, Pamela Mishkin, Chong Zhang,	Chen, Wei Zhou, Jinjie Gu, Chenyi Zhuang,	765
710	Sandhini Agarwal, Katarina Slama, Alex Ray, and 1	Ruocheng Guo, Bowen Yu, Wanyu Wang, and 1 oth-	766
711	others. 2022. Training language models to follow in-	ers. 2025. Function calling in large language models:	767
712	structions with human feedback. <i>Advances in neural</i>	Industrial practices, challenges, and future directions.	768
713	<i>information processing systems</i> , 35:27730–27744.	Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang,	769
714	Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E	Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Exe-	770
715	Gonzalez. 2024. Gorilla: Large language model	cutable code actions elicit better llm agents. In <i>Forty-</i>	771
716	connected with massive apis. <i>Advances in Neural</i>	<i>first International Conference on Machine Learning</i> .	772
717	<i>Information Processing Systems</i> , 37:126544–126565.	Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu,	773
718	Akshara Prabhakar, Zuxin Liu, Weiran Yao, Jianguo	Xiaojian Shawn Ma, and Yitao Liang. 2023. De-	774
719	Zhang, Ming Zhu, Shiyu Wang, Zhiwei Liu, Tulika	scribe, explain, plan and select: interactive planning	775
720	Awalgaonkar, Haolin Chen, Thai Hoang, and 1 others.	with llms enables open-world multi-task agents. <i>Ad-</i>	776
721	2025. Apigen-mt: Agentic pipeline for multi-turn	<i>Advances in Neural Information Processing Systems</i> ,	777
722	data generation via simulated agent-human interplay.	36:34153–34189.	778
723	<i>arXiv preprint arXiv:2504.03601</i> .	Bin Wu, Edgar Meij, and Emine Yilmaz. 2025. A joint	779
724	Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang,	optimization framework for enhancing efficiency of	780
725	Xiuxi Chen, Dilek Hakkani-Tür, Gokhan Tur, and	tool utilization in llm agents. In <i>Findings of the As-</i>	781
726	Heng Ji. 2025. Toolrl: Reward is all tool learning	<i>sociation for Computational Linguistics: ACL 2025</i> ,	782
727	needs. <i>arXiv preprint arXiv:2504.13958</i> .	pages 22361–22373.	783

784 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang,
785 Binyuan Hui, Bo Zheng, Bowen Yu, Chang
786 Gao, Chengen Huang, Chenxu Lv, and 1 others.
787 2025a. Qwen3 technical report. *arXiv preprint*
788 *arXiv:2505.09388*.

789 Kang Yang, Guan hong Tao, Xun Chen, and Jun Xu.
790 2025b. Alleviating the fear of losing alignment in llm
791 fine-tuning. In *2025 IEEE Symposium on Security*
792 *and Privacy (SP)*, pages 2152–2170. IEEE.

793 Zhaojian Yu, Yilun Zhao, Arman Cohan, and Xiao-
794 Ping Zhang. 2024. Humaneval pro and mbpp pro:
795 Evaluating large language models on self-invoking
796 code generation. *arXiv preprint arXiv:2412.21199*.

797 Kingshan Zeng, Weiwen Liu, Xu Huang, Zezhong
798 Wang, Lingzhi Wang, Liangyou Li, Yasheng Wang,
799 Lifeng Shang, Xin Jiang, Ruiming Tang, and 1 oth-
800 ers. 2025a. Toolace-r: Tool learning with adaptive
801 self-refinement. *arXiv preprint arXiv:2504.01400*.

802 Kingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou
803 Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin
804 Jiang, and Qun Liu. 2025b. Toolace-mt: Non-
805 autoregressive generation for agentic multi-turn inter-
806 action. *arXiv preprint arXiv:2508.12685*.

807 Ji Zhang, Shihan Wu, Xu Luo, Hao Wu, Lianli Gao,
808 Heng Tao Shen, and Jingkuan Song. 2025a. Inspire:
809 Vision-language-action models with intrinsic spatial
810 reasoning. *arXiv preprint arXiv:2505.13888*.

811 Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai
812 Hoang, Shirley Kokane, Weiran Yao, Juntao Tan,
813 Akshara Prabhakar, Haolin Chen, and 1 others. 2024.
814 xlam: A family of large action models to empower ai
815 agent systems. *arXiv preprint arXiv:2409.03215*.

816 Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz,
817 Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhid-
818 ing Yu, and Guilin Liu. 2025b. Nemotron-research-
819 tool-n1: Tool-using language models with reinforced
820 reasoning. *arXiv preprint arXiv:2505.00024*.

821 Yaowei Zheng, Richong Zhang, Junhao Zhang, Yan-
822 han Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang
823 Ma. 2024. Llamafactory: Unified efficient fine-
824 tuning of 100+ language models. *arXiv preprint*
825 *arXiv:2403.13372*.

826 Zhe Zheng, Ke-Yin Chen, Xin-Yu Cao, Xin-Zheng
827 Lu, and Jia-Rui Lin. 2023. Llm-funcmapper:
828 Function identification for interpreting complex
829 clauses in building codes via llm. *arXiv preprint*
830 *arXiv:2308.08728*.

831 Tianyang Zhong, Zhengliang Liu, Yi Pan, Yutong
832 Zhang, Yifan Zhou, Shizhe Liang, Zihao Wu, Yanjun
833 Lyu, Peng Shu, Xiaowei Yu, and 1 others. 2024. Eval-
834 uation of openai o1: Opportunities and challenges of
835 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 token length of function call of the entire dataset (including rejections) is 31.07, the average cot token length is 350.74. In the first data filtering, we obtain 85014 qualified simple data and 3178 difficult data for self-evolving, dropping 1808 samples.

A.2 Error 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 by 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., 2024) for training; the key hyperparameters for training are shown in Table 7.

C More Results

C.1 Results of 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 8, even though our base model has almost no agentic capability and we do not include any Web Search or Memory related data 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.2 Results on Qwen3

To further verify the effectiveness of BalanceSFT, we conduct experiments by using the more recently

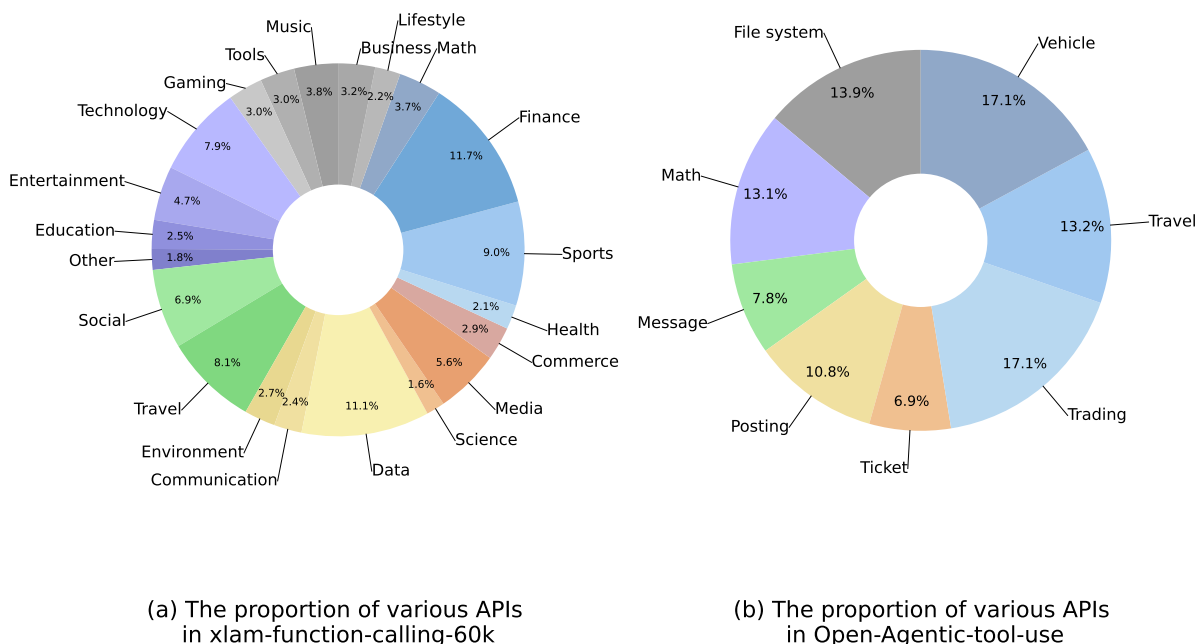


Figure 5: API categories distribution of xlam-function-calling-60k and Open-Agentic-tool-use. Different colors represent the distribution of different API categories.

Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
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.

open-sourced Qwen3-4B-Instruct-2507. The detailed results are shown in Table 9. Compared with the standard SFT method, the proposed BalanceSFT still performs well on the Qwen3 model.

C.3 Relationship between α and token ratios

To investigate the relationship between α and token distribution characteristics, we conduct controlled experiments across datasets with varying CoT-to-function-call (CoT/FC) ratios: 5:1, 10:1, 15:1, and 20:1. Each dataset contains 2,000 samples, and models are trained with learnable α initialized at 0.7. The results in table 10 reveal a clear inverse relationship: as the CoT/FC Ratio increases from 5:1 to 20:1, the final converged α decreases monotonically from 0.75 to 0.58. This adaptive behavior demonstrates that BalanceSFT automatically adjusts the balance parameter in response to token distribution—when reasoning tokens dominate (higher CoT/FC ratios), α decreases to allocate relatively more learning signal to function calling

tokens, thereby maintaining balanced optimization across both token types. This self-regulating mechanism validates our hypothesis that learnable α can effectively compensate for inherent data imbalances without manual intervention.

C.4 Open-sourced 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 11 presents a comparison between QwQ-32B and proprietary model judgments: BalanceSFT-7B scores 84.12/51.12 versus 84.90/52.25 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 0.69 points (Single-Turn) and 1.13 points (Multi-Turn). This narrow gap suggests that BalanceSFT’s effective-

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 8: Performance of model’s agentic ability.

Models	Overall	Multi-Turn					Single-Turn		
		Overall	Base	Miss Func	Miss Param	Long Context	Overall	Non-Live	Live
Qwen3-4B-Inst-2507	62.13	22.12	26.50	21.00	15.50	25.50	82.14	87.88	76.39
w/ SFT	71.50	47.13	55.00	52.00	42.50	39.00	83.69	87.23	80.14
w/ BalanceSFT	74.50	52.63	58.50	56.00	44.50	51.50	85.44	89.56	81.32

Table 9: Performance of Qwen3-4B-Instruct-2507 on BFCL.

CoT/FC Ratio	Final α
5:1	0.75
10:1	0.68
15:1	0.62
20:1	0.58

Table 10: Relationship between α and token ratios.

Judge	Single-Turn	Multi-Turn
QwQ-32B	84.21	51.12
proprietary models	84.90	52.25

Table 11: Performance across different judges.

924 ness is evaluator-agnostic and that our method pro-
925 duces robust improvements that are consistently
926 recognized across different judgment paradigms,
927 including those based on more accessible open-
928 source models.

929 **D Details of Hard Data Re-sampling** 930 **Strategy**

931 In the pipeline of data refinement, we utilize QwQ-
932 32B to assist in different levels of identification and
933 correction. The prompts are shown in the following
934 figures.

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: A Example for error in 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