

FUSE: Failure-Guided Scenario Synthesis and Weighted Refinement for LLM Function Calling

Zhiguo Tao

University of Nottingham, Nottingham, UK
ssyzt3@outlook.com

Linhao Li

Charter Communications, Denver, CO, USA
kyson.li@spectrum.com

Lifan Sun

Independent Researcher, Seattle, WA, USA
lifansun1412@gmail.com

Zichen Yuan

Independent Researcher, Shanghai, China
yzc66633@gmail.com

Abstract

Large language model (LLM) agents increasingly rely on function calling to act across multi-turn interactions, but compact models remain brittle when tool availability, required parameters, and dialogue constraints change over time. Existing synthetic tool-use data often broadens coverage without targeting the recurring failures of a specific model. We present **FUSE**, a failure-guided refinement approach that uses the model’s own function-calling errors to synthesize harder training scenarios and then emphasizes difficult examples during supervised fine-tuning. This data-centric design turns diagnostic failures into targeted training signal while remaining simple to apply with standard offline fine-tuning. Experiments on BFCLv3 show that FUSE substantially improves a compact Qwen3-based model across standard and stress categories, including missing functions, missing parameters, and long-context interactions. Evaluation on BFCLv4 provides initial evidence that the same refinement strategy can transfer to selected agentic function-calling tasks.

Keywords: function calling, large language models, tool use, sequential decision-making, hard scenario synthesis, weighted refinement

1 Introduction

Large language models (LLMs) increasingly act through function calls: they select an external tool, fill its arguments, observe the result, and continue the interaction (Schick et al., 2023; Patil et al., 2023, 2025a; Yuan et al., 2025; Zhou et al., 2025). In multi-turn settings, this is not a single API-selection problem. At each assistant turn, the model must decide whether to call a function or respond in natural language, which visible function to use, which arguments are justified by the dialogue, and how to proceed when the requested tool or a required parameter is unavailable. A small mistake

can change the state of the interaction: an unavailable function may be hallucinated, a missing argument may be invented, or an earlier user constraint may be dropped several turns later.

These failures are especially important for compact LLM agents. Smaller open-weight models are attractive for deployment because they are cheaper and easier to serve, but empirical evidence shows that small models can struggle with strict function-call formatting (Kavathekar et al., 2025), while BFCL evaluations expose broader challenges in fallback behavior, schema-aware decisions, and stateful interaction (Patil et al., 2025a). For example, when a user asks for an operation whose function is absent from the visible tool list, a robust agent should ask for clarification or explain the limitation; a brittle model may instead emit a plausible-looking call to a tool that does not exist. Similar failures occur when the model fabricates optional parameters, omits required ones, or loses constraints accumulated across a dialogue. Such errors are not just formatting mistakes; they are action-selection failures in an agent interface.

Prior work has made substantial progress on tool-use learning and synthetic function-calling data (Qin et al., 2024; Liu et al., 2024b; Prabhakar et al., 2025; Liu et al., 2025; Song et al., 2024; Su et al., 2025; Sun et al., 2025). These methods broaden coverage by sampling tasks from tool schemas, simulated interactions, execution traces, or multi-agent data generation. However, coverage-oriented synthesis is not necessarily aligned with the recurring errors of a particular compact model. In multi-turn function calling, the hardest cases for a deployed model are often model-specific: one model may over-call unavailable tools, another may lose long-context constraints, and another may fail mostly at parameter grounding. This motivates a refinement strategy that starts from observed failures rather than from generic data expansion alone.

Figure 1 illustrates this motivation. Generic syn-

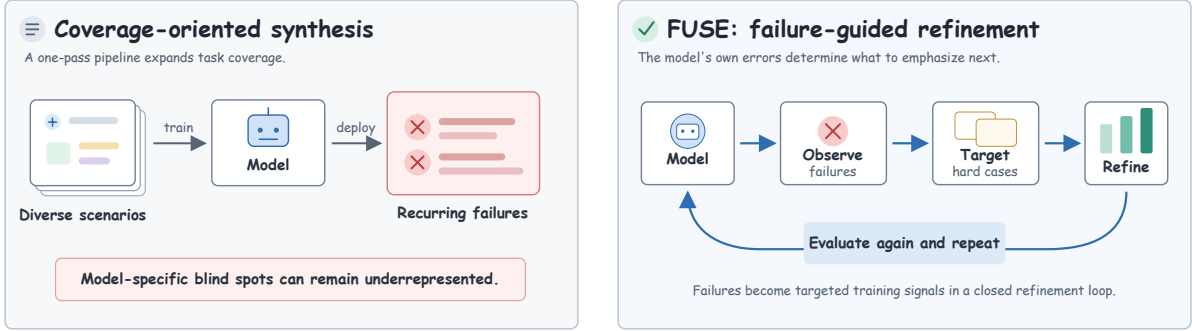


Figure 1: Motivation for FUSE. Coverage-oriented synthesis broadens tool-use data, whereas FUSE uses the current model’s failed turns to create targeted hard scenarios for the recurring failures that matter most.

thesis can broaden tool-use data, but it may leave recurring model-specific failures underemphasized. FUSE instead uses observed function-calling failures to guide hard scenario synthesis and supervised refinement.

We propose **FUSE**, a failure-guided scenario synthesis and weighted refinement approach for improving LLM function calling capability. FUSE first evaluates the current model on a diagnostic partition of multi-turn trajectories and logs failed assistant turns using rule-based checks. It then maps failures to a small set of predefined failure prototypes, uses those prototypes to synthesize targeted hard scenarios, applies quality filters to the generated scenarios, and trains the model with weighted supervised fine-tuning. The key idea is simple: the model’s own failures become data-construction signals, and difficult examples receive more emphasis during fine-tuning.

We evaluate FUSE on Berkeley Function-Calling Leaderboard version 3 (BFCLv3) and further examine transfer on selected Berkeley Function-Calling Leaderboard version 4 (BFCLv4) agentic tasks (Patil et al., 2025a,b). Our evaluation examines four complementary aspects: overall improvement on BFCLv3, performance on failure-aligned multi-turn stress cases, the contributions of synthesis, weighting, and iteration, and transfer to BFCLv4 agentic tasks. On BFCLv3, FUSE substantially improves a compact Qwen3-based model across both standard and stress categories, including missing functions, missing parameters, and long-context interactions. BFCLv4 results provide initial evidence of transfer beyond the refinement setting.

Our contributions are:

- We introduce a failure-guided view of multi-turn function-calling refinement, where the tar-

get model’s own failed turns are used to identify which tool-use situations should be emphasized during training.

- We present FUSE, a practical data-centric approach that turns failed turns into prototype-conditioned hard scenarios and uses quality-filtered, difficulty-weighted examples for supervised refinement.
- We evaluate FUSE on BFCLv3 and selected BFCLv4 tasks, showing large gains across standard and stress categories and providing analysis of where the refinement strategy transfers.

2 Related Work

2.1 LLM Function Calling and Agentic Tool Use

LLM tool use has developed from prompting-based API invocation toward dedicated training and evaluation for function calling. Toolformer (Schick et al., 2023) and ToolLLM (Qin et al., 2024) showed that language models can learn to select and call external tools, while Gorilla (Patil et al., 2023) demonstrated the value of API-aware training for reducing tool hallucination. Benchmarks such as API-Bank (Li et al., 2023b) and AgentBench (Liu et al., 2024a) broadened evaluation toward tool-augmented and interactive agent behavior.

Recent work further shifts function-calling evaluation from isolated API selection toward serial, parallel, multi-turn, and stateful agent behavior. The Berkeley Function-Calling Leaderboard (BFCL) formalizes broad executable evaluation for function calling (Patil et al., 2025a); its current suite includes missing-function, missing-parameter, and long-context stress cases as well as BFCLv4 web-search and memory tasks (Patil et al., 2025b). Specialized releases such as ToolACE (Liu et al.,

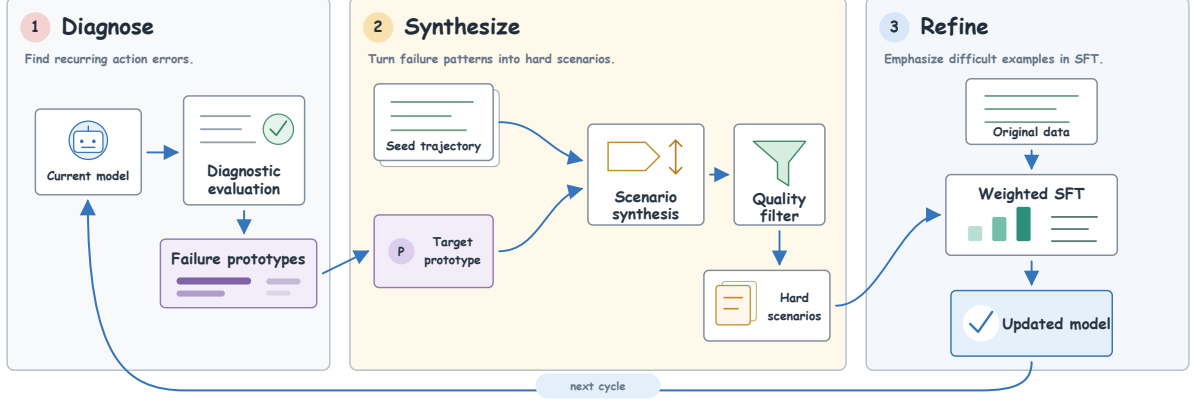


Figure 2: FUSE pipeline. Diagnostic evaluation exposes failed turns, their prototype distribution guides hard-scenario synthesis, quality filters keep usable cases, and weighted supervised fine-tuning emphasizes difficult examples before the next cycle.

2025), xLAM (Zhang et al., 2025), and BitAgent (BitAgent, 2025) provide open-weight checkpoints aimed at function calling across a range of model scales. Nevertheless, strict output formatting remains difficult for some small models (Kavathekar et al., 2025), and BFCL identifies stateful interaction and memory as continuing challenges (Patil et al., 2025a). FUSE focuses on this multi-turn agentic setting: it treats recurring function-calling failures as signals for where targeted refinement data should be created.

2.2 Synthetic Data for Tool-Use Agents

Synthetic data has been central to instruction tuning and tool-use learning. Self-Instruct bootstraps instruction data from model generations (Wang et al., 2023), while WizardLM evolves seed instructions into more complex variants (Xu et al., 2024). APIGen (Liu et al., 2024b) builds verifiable API-calling examples at scale, and APIGen-MT extends this direction to multi-turn trajectories generated through simulated agent-human interaction (Prabhakar et al., 2025). ToolAlpaca (Tang et al., 2023) uses multi-agent simulation to construct a diverse tool-use corpus. Role-playing frameworks such as CAMEL (Li et al., 2023a) generate conversational interactions, while ToolACE (Liu et al., 2025) uses multi-agent collaboration to improve function-calling data.

The closest line of work to FUSE is multi-turn and hard-case synthesis for tool-use agents. FunReason-MT (Xu et al., 2025) combines environment-API graph interactions, hard-query synthesis, and guided reasoning traces. ToolACE-MT (Zeng et al., 2026) instead constructs full di-

alogue skeletons and improves them through iterative mask-and-fill refinement. These approaches improve data coverage and trajectory complexity, but their generation process is primarily organized around tool dependencies, dialogue structure, simulated agents, or environment interactions. FUSE instead starts from the current model’s failed turns. This makes synthesis model-specific: generated scenarios are conditioned on the observed prototype distribution, filtered for quality, and then used as weighted supervised training examples.

3 Method

3.1 Problem Formulation

We consider multi-turn function calling, where an assistant observes a dialogue prefix, a set of visible function schemas, and any previous tool outputs, then selects the next assistant action. At turn t , the model input is

$$s_t = (h_t, \mathcal{F}_t, o_{<t}), \quad (1)$$

where h_t is the dialogue history, $\mathcal{F}_t$ is the set of currently visible function schemas, and $o_{<t}$ denotes previous tool outputs. The target action a_t^* is either a function call (f, args) or a natural-language response such as an answer, clarification, or fallback. A multi-turn trajectory is a sequence of such decision points:

$$\tau = \{(s_t, a_t^*)\}_{t=1}^T. \quad (2)$$

FUSE targets recurring action errors made by a compact model in this setting, including unavailable-function calls, wrong function selection, missing required arguments, hallucinated parameters, and long-context constraint loss.

3.2 Architecture Overview

Figure 2 shows the FUSE pipeline. FUSE is a closed-loop, data-centric refinement procedure: it uses a fixed diagnostic partition to identify what the current model gets wrong, mutates separate seed trajectories into examples of those errors, and re-trains on the resulting data mixture with greater exposure to examples the model finds difficult. FUSE uses APIGen-MT-5k as the multi-turn trajectory pool (Prabhakar et al., 2025). A fixed 20% partition is reserved for diagnostic evaluation and is not directly used for SFT; the remaining 80% serves two roles: it provides seed trajectories for hard-scenario synthesis and remains part of the supervised refinement data. Auxiliary APIGen function-calling data adds generic supervised examples for schema-following coverage (Liu et al., 2024b). BFCLv3 and BFCLv4 are used for evaluation only.

At each refinement cycle, the current model is evaluated on the diagnostic partition. Rule-based checks identify failed assistant turns, which are assigned to predefined failure prototypes by a closed-label LLM classifier. The observed proportions of the synthesis-target prototypes determine how the generation budget is distributed across them. A generator LLM then synthesizes hard multi-turn scenarios from seed trajectories conditioned on the selected prototypes. Quality filters remove malformed, inconsistent, or duplicate generations. Finally, accepted scenarios are segmented into assistant-turn examples and used in difficulty-weighted supervised fine-tuning.

3.3 Failure Logging and Prototype Assignment

We log failures with rule-based checks. The diagnostic protocol is teacher-forced and turn-local: each assistant decision is evaluated from the reference dialogue prefix and previous reference tool outputs, so an earlier model error does not change later states. For each assistant turn in the diagnostic partition, FUSE records the dialogue prefix, visible tools, model output, reference action, turn index, and low-level error type. The checker verifies parseability, tool availability, schema validity, required arguments, reference-action matching, and fallback behavior. For example, if the reference action is a clarification because a required parameter is absent, a generated function call that invents the parameter is marked as a failure.

Failed turns are then mapped to high-level pro-

Table 1: Failure prototypes used by FUSE. Prototype labels guide scenario synthesis, while rule-based checks determine which diagnostic turns are failures.

ID	Prototype
P1	Missing-tool fallback failure: calls an unavailable function instead of falling back or asking for clarification.
P2	Underspecified-parameter handling failure: invents or omits a required argument when the user has not provided enough information.
P3	Argument grounding error: chooses the right function but uses an argument value not supported by the dialogue or tool output.
P4	Wrong function selection: selects a semantically related but incorrect function.
P5	Long-context constraint loss: drops a constraint introduced in earlier turns.
P6	Other: failure does not fit the synthesis-target prototypes above.

totypes with a closed-label LLM classifier using gpt-5-mini. The classifier operates after rule-based failure detection and assigns each failed turn to one of the predefined categories in Table 1. P1–P5 are used as synthesis targets, with generation allocated in proportion to their observed frequencies in the diagnostic partition. Other failures are retained in the diagnostic logs but do not condition hard-scenario generation.

3.4 Prototype-Conditioned Scenario Synthesis

FUSE synthesizes new hard scenarios from the 80% seed split. Operationally, this is a prototype-guided mutation process: FUSE preserves a seed trajectory’s domain and tool schemas while changing the request, context, tool availability, or reference action to instantiate a diagnosed failure mode. The generator is prompted with a target prototype, a seed trajectory, the visible function schemas, and a prototype-specific transformation instruction. We use gpt-5-mini as the generator. The output is a candidate multi-turn scenario containing the visible tools, dialogue turns, reference assistant actions, target prototype, and a short rationale linking the case to the prototype.

Each transformation keeps the domain and interaction style of the seed trajectory and adds a prototype-specific stressor. Missing-tool cases remove or withhold a needed function, and the reference action is a fallback or clarification. Underspecified-parameter cases remove a required user detail, making clarification the correct next action rather than argument invention. Argument-grounding cases preserve the correct function but

make one value depend on explicit dialogue evidence or a prior tool output. Wrong-function cases introduce semantically similar tools whose schemas differ in subtle but task-relevant ways. Long-context cases place key constraints in earlier turns and insert intervening distractor turns. This conditioning keeps synthesis tied to observed failure patterns rather than to generic task expansion alone.

3.5 Quality Filtering and Example Construction

Generated scenarios are not used directly. FUSE applies rule-based quality filters to reject malformed or inconsistent candidates and to enforce structural validity under the scenario’s visible tools and schemas. The filters check whether reference calls are parseable, whether function names and argument names match visible schemas, whether reference calls use only available tools, whether required parameters are present when a function call is expected, and whether prototype-specific structural conditions are satisfied. Exact and near-duplicate scenarios are removed. These checks constrain structural validity but do not independently certify every semantic detail of a generated dialogue.

Each accepted multi-turn scenario is segmented at assistant turns. For an accepted turn, the SFT input is the dialogue prefix plus visible tool schemas, and the target is the reference assistant action. Each example keeps metadata indicating its source, prototype, diagnostic cycle, and filter status. The final supervised refinement set has three sources: original examples segmented from the 80% APIGen-MT seed partition, accepted FUSE hard scenarios generated from those seeds, and auxiliary APIGen function-calling data. The 20% diagnostic partition contributes failure statistics and prototype labels but is not included as raw SFT data.

3.6 Difficulty-Weighted Supervised Refinement

Prototype labels guide which hard scenarios are generated, whereas difficulty weights are computed over the entire final SFT mixture. Thus, synthetic examples are not automatically assigned higher weight: all examples are emphasized according to how difficult their reference action is for the current model. For each training example $e = (s_e, a_e^*)$, we compute the current model’s average negative log-

likelihood on the reference output:

$$\ell(e) = -\frac{1}{|a_e^*|} \sum_i \log \pi_{\theta_{\text{old}}}(a_i^* \mid s_e, a_{<i}^*). \quad (3)$$

We convert $\ell(e)$ into a percentile score $q(e) \in [0, 1]$, where larger values indicate examples that the current model finds more difficult. The example weight is

$$w(e) = \frac{\text{clip}(1 + \lambda q(e), w_{\min}, w_{\max})}{\mathbb{E}_{e'}[\text{clip}(1 + \lambda q(e'), w_{\min}, w_{\max})]}. \quad (4)$$

In all reported runs, we use $\lambda = 1.0$, $w_{\min} = 0.5$, and $w_{\max} = 3.0$, with weights normalized to have mean 1. The clipping bounds do not bind under $\lambda = 1.0$ because $q(e) \in [0, 1]$, but they are kept in the implementation for larger weighting strengths. The corresponding weighted SFT objective is:

$$\mathcal{L}(\theta) = - \sum_{e \in \mathcal{D}_{\text{train}}} w(e) \log \pi_{\theta}(a_e^* \mid s_e). \quad (5)$$

We implement this objective by sampling the final SFT set with probability proportional to $w(e)$ and then applying standard language-model training. Thus, FUSE is a supervised refinement procedure implemented through data construction and resampling.

4 Experiments

4.1 Experimental Setup

Benchmarks. The primary evaluation uses BFCLv3, which includes Single-Turn and Multi-Turn function-calling tasks (Patil et al., 2025a). We additionally evaluate on BFCLv4 Web Search and Memory tasks (Patil et al., 2025b) to examine transfer to separate agentic task formats. Single-Turn (ST) denotes one user request with one tool-use decision context; Multi-Turn (MT) denotes dialogues requiring tool-use decisions across multiple turns.

Baselines. We compare against proprietary models, including GPT-5-2025-08-07, GPT-4o-2024-11-20, Claude-Sonnet-4-20250514, Gemini-2.5-Pro, o3-2025-04-16, and Grok-4-0709, as well as open-source and specialized tool-use models including Kimi-K2-Instruct (Kimi Team et al., 2025), DeepSeek-R1-0528 (DeepSeek-AI et al., 2025), Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025), ToolACE-2-8B (Liu et al., 2025), BitAgent-8B (BitAgent, 2025), watt-tool-8B (watt-ai, 2024; Shi et al., 2024), xLAM-2-3b-fc-r (Zhang et al., 2025), ToolACE-MT (Zeng et al., 2026), and

Table 2: BFCLv3 Multi-Turn accuracy (%). Model identifiers are abbreviated.

Model	Overall	Base	Miss Func	Miss Param	Long Context
GPT-5	28.50	33.50	29.50	23.00	28.00
GPT-4o	42.50	55.50	34.50	29.00	51.00
Claude-Sonnet-4	54.75	64.00	54.00	47.50	53.50
Gemini-2.5-Pro	25.00	25.50	26.00	24.50	24.00
o3	38.38	44.00	40.50	31.50	37.50
Grok-4	36.12	44.00	31.00	26.00	43.50
Kimi-K2	41.25	51.00	43.00	31.00	40.00
DeepSeek-R1-0528	44.50	54.50	41.00	36.50	46.00
Qwen3-235B	39.62	53.50	34.50	27.50	43.00
ToolACE-2-8B	37.00	47.00	31.00	28.00	42.00
BitAgent-8B	37.75	46.50	37.50	24.00	43.00
watt-tool-8B	37.88	45.50	39.00	24.00	43.00
xLAM-2-3B	57.12	73.50	55.00	54.50	45.50
ToolACE-MT	40.25	57.50	31.50	34.00	38.00
Qwen3-4B	15.75	19.00	15.50	12.50	16.00
FunReason-MT SFT	46.90	53.20	47.10	40.40	46.90
FunReason-MT RL	57.75	62.50	60.50	48.50	59.50
FUSE (Ours)	60.38	74.25	62.00	55.75	61.00

FunReason-MT (Xu et al., 2025). FUSE is built on Qwen3-4B-Instruct-2507 (Yang et al., 2025). Scores for FUSE and its ablations are produced with the BFCL evaluator. External baseline scores are reported for the listed model identifiers from BFCL evaluator/leaderboard entries when available; Inst and Instruct are model-name variants denoting instruction-tuned releases.

Metric. All results are reported as accuracy (%). Overall denotes the aggregate score reported for a benchmark split, Base denotes the standard non-stress subset in BFCL multi-turn evaluation, Miss Func denotes missing-function cases, Miss Param denotes missing-parameter cases, Long Context denotes long-context multi-turn cases, KV denotes key-value memory tasks, and Vector denotes vector-memory tasks.

Implementation details. Full implementation settings are provided in Appendix A; Appendix B gives prompt examples.

4.2 BFCLv3 Main Results

Table 2 reports the primary BFCLv3 Multi-Turn results. FUSE reaches 60.38% Multi-Turn Overall accuracy, compared with 15.75% for the Qwen3-4B-Instruct base model. Large absolute gains also appear in the stress categories that match the failure-guided design: Miss Func, Miss Param, and Long Context.

Table 3 reports complementary BFCLv3 results. Although FUSE is motivated by multi-turn decision-making, it also improves Single-Turn Overall accuracy over the base model, suggesting that the refinement procedure preserves standard function-calling accuracy in this evaluation.

Table 3: Complementary BFCLv3 accuracy (%).

(a) Single-Turn			
Model	Overall	Non-Live	Live
Claude-Sonnet-4	84.72	88.38	81.05
Qwen3-235B	83.37	90.12	76.61
ToolACE-2-8B	82.54	87.87	77.20
Qwen3-4B	78.19	86.35	70.02
FunReason-MT SFT	81.97	83.36	80.57
FunReason-MT RL	85.47	89.58	81.35
FUSE (Ours)	86.33	90.85	81.80
(b) Targeted Multi-Turn stress subsets			
Model	MT Miss Func	MT Long Context	
Claude-Sonnet-4	54.00	53.50	
xLAM-2-3b	55.00	45.50	
FunReason-MT RL	60.50	59.50	
FUSE (Ours)	62.00	61.00	

4.3 Failure-Aligned BFCLv3 Stress Subsets

Miss Func stresses unavailable-tool handling and fallback behavior, while Long Context stresses the retention of constraints across turns. Table 3(b) shows that FUSE improves over the compact base model by 46.50 points on Miss Func and 45.00 points on Long Context. Compared with the closest listed specialized baseline, FUSE is ahead by 1.50 points on both subsets, indicating competitive performance on the targeted stress cases. The two subsets also probe different behaviors: fallback under missing tools and constraint retention across long contexts. FUSE is balanced across the two, whereas xLAM-2-3b is stronger on Miss Func than on Long Context.

4.4 Component Analysis

Table 4 reports component removals and intermediate variants. Hard Synth. denotes prototype-conditioned hard scenario synthesis, Weight denotes difficulty-weighted resampling, Iter. denotes multiple refinement cycles, and Δ_{Full} denotes the score difference from the full FUSE model on BFCLv3 and BFCLv4 respectively.

Removing hard scenario synthesis lowers BFCLv3 Overall from 60.38% to 53.12% and BFCLv4 Web Search Overall from 16.85% to 14.40%. The unweighted single-pass variant reaches 48.25% on BFCLv3 and 13.20% on BFCLv4; this variant removes both weighting and later iteration, so the comparison reflects the combined effect of those two design choices. Comparing the unweighted single-pass variant with the weighted single-pass variant suggests that weighting adds 6.85 points on BFCLv3, while comparing the weighted single-pass variant with the full model suggests that later

Table 4: FUSE ablations. BFCLv3 reports Multi-Turn Overall; BFCLv4 reports Web Search Overall. All values are accuracy (%).

(a) Overall ablation						
Variant	Hard Synth.	Weight	Iter.	Overall (%)		Δ_{Full}
				BFCLv3	BFCLv4	
Base (Qwen3-4B)	–	–	–	15.75	8.85	-44.63 / -8.00
w/o Hard Synth.	–	✓	✓	53.12	14.40	-7.26 / -2.45
w/o Weight + Iter.	✓	–	–	48.25	13.20	-12.13 / -3.65
w/o Iteration	✓	✓	–	55.10	15.40	-5.28 / -1.45
FUSE (Full)	✓	✓	✓	60.38	16.85	0.00 / 0.00

(b) Targeted stress-subset ablation				
Model Variant	MT Miss Func	MT Long Context		
FUSE w/o Hard Synthesis	56.50	55.00		
FUSE w/o Weight + Iter.	50.25	49.50		
FUSE w/o Iteration	58.50	58.00		
FUSE (Full)	62.00	61.00		

Table 5: BFCLv4 Web Search transfer results. All values are accuracy (%).

Model	Overall	Base	No Snippet	Recency
ToolACE-2-8B	14.83	9.00	13.00	5.00
Qwen3-4B	8.85	5.00	4.00	6.00
FunReason-MT SFT	9.99	9.00	12.00	6.00
FunReason-MT RL	<u>15.10</u>	<u>16.00</u>	<u>20.00</u>	<u>12.00</u>
FUSE (Ours)	16.85	18.00	21.50	13.00

iteration adds 5.28 points. Overall, the ablations indicate that prototype-conditioned synthesis, weighting, and repeated refinement each contribute under the evaluated setup.

The stress-subset ablation in Table 4(b) further tests the same variants on Miss Func and Long Context. Removing hard synthesis lowers both stress categories. The unweighted single-pass variant lowers them further, matching the main ablation. Removing later iteration has the smallest effect, indicating that a single weighted refinement pass captures much of the targeted-subset gain but does not match the final model.

4.5 Transfer to BFCLv4

Tables 5 and 6 show mixed results on BFCLv4 agentic tasks: FUSE improves on Web Search, but Memory performance depends strongly on the memory type.

FUSE achieves the best Web Search scores among the listed models and the best KV and Recursive Sum memory scores. On Memory Overall and Memory Vector, ToolACE and BitAgent remain stronger. These results indicate partial, backend-dependent transfer rather than a uniform

Table 6: BFCLv4 Memory transfer results. All values are accuracy (%).

Model	Overall	KV	Vector	Recursive Sum
ToolACE-2-8B	<u>20.65</u>	7.74	<u>18.06</u>	36.13
BitAgent-8B	12.47	4.52	20.65	12.26
Qwen3-4B	12.69	<u>11.61</u>	11.61	14.84
FunReason-MT RL	14.19	5.81	0.00	<u>36.77</u>
FUSE (Ours)	<u>16.05</u>	12.90	2.42	38.71

improvement across memory settings. In particular, FUSE’s Vector score drops from the base model’s 11.61% to 2.42%, a clear regression that suppresses Memory Overall to 16.05%.

5 Conclusion

We showed that compact function-calling models can be strengthened by treating their own failed turns as data-design signals. FUSE logs rule-detected failures, maps them to failure prototypes, synthesizes targeted hard scenarios according to the observed prototype distribution, and uses difficulty-weighted supervised refinement to emphasize examples that expose recurring weaknesses. On BFCLv3, this strategy substantially improves a compact Qwen3-4B model across standard and stress categories; BFCLv4 results show partial transfer, with clear gains on Web Search and selected memory tasks but a regression on vector-memory behavior. The broader lesson is that targeted failure analysis can make synthetic data generation more model-aware. Future work should extend this idea to adaptive prototype discovery, richer memory failures, and more interactive planning environments.

6 Limitations

FUSE relies on gpt-5-mini for prototype assignment and scenario generation, so the refinement data may inherit priors from this stronger model even though structural validity is checked by rules. The current prototypes also do not cover all agentic behaviors: FUSE improves several BFCLv4 transfer settings but regresses on Vector memory.

7 Ethics Statement

This work uses public or synthetic function-calling datasets and does not involve private user data or human-subject annotation. Because stronger function-calling agents may increase deployment risk, FUSE should be paired with application-specific tool permissions, safety checks, and monitoring in real systems.

References

- BitAgent. 2025. [BitAgent-8B model card](#). Hugging Face model card.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, et al. 2025. [DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning](#). *Preprint*, arXiv:2501.12948.
- Ishan Kavathekar, Raghav Donakanti, Ponnuram Kumaraguru, and Karthik Vaidhyanathan. 2025. [Small models, big tasks: An exploratory empirical study on small language models for function calling](#). *Preprint*, arXiv:2504.19277.
- Kimi Team, Yifan Bai, Yiping Bao, Y. Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. 2025. [Kimi K2: Open agentic intelligence](#). *Preprint*, arXiv:2507.20534.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. [CAMEL: Communicative agents for Mind exploration of large language model society](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 51991–52008.
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. [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. Association for Computational Linguistics.
- Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2025. [ToolACE: Winning the points of LLM function calling](#). In *International Conference on Learning Representations*.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2024a. [AgentBench: Evaluating LLMs as agents](#). In *International Conference on Learning Representations*.
- Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024b. [APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 54463–54482. Datasets and Benchmarks Track.
- Ilya Loshchilov and Frank Hutter. 2019. [Decoupled weight decay regularization](#). In *International Conference on Learning Representations*.
- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025a. [The berkeley function calling leaderboard \(BFCL\): From tool use to agentic evaluation of large language models](#). In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR.
- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, et al. 2025b. [Berkeley function calling leaderboard](#). GitHub repository.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. [Gorilla: Large language model connected with massive APIs](#). *Preprint*, arXiv:2305.15334.
- Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. [APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay](#). In *Advances in Neural Information Processing Systems*, volume 38. Datasets and Benchmarks Track.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. [ToolLLM: Facilitating large language models to master 16000+ real-world APIs](#). In *International Conference on Learning Representations*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551.
- Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. 2024. [Direct multi-turn preference optimization for language agents](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2312–2324. Association for Computational Linguistics.
- Xinyuan Song, Keyu Chen, Ziqian Bi, Qian Niu, Junhao Song, Junyu Liu, Benji Peng, Sen Zhang, Ming Liu, Ming Li, et al. 2024. Mastering reinforcement learning: Foundations, algorithms, and real-world applications.
- Jinwei Su, Qizhen Lan, Yinghui Xia, Lifan Sun, Weiyou Tian, Tianyu Shi, Xinyuan Song, Lewei He, and Yang Jingsong. 2025. Difficulty-aware agentic orchestration for query-specific multi-agent workflows. *arXiv preprint arXiv:2509.11079*.
- Lifan Sun, Yichao Zhang, Xinyuan Song, Ziqian Bi, Keyu Chen, Ming Li, Qian Niu, Junyu Liu, Benji Peng, Sen Zhang, et al. 2025. Exploring multimodal

embeddings for text and impact on language processing.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. [ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases](#). *Preprint*, arXiv:2306.05301.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. [Self-instruct: Aligning language models with self-generated instructions](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508. Association for Computational Linguistics.

watt-ai. 2024. [watt-tool-8B model card](#). Hugging Face model card.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. [WizardLM: Empowering large pre-trained language models to follow complex instructions](#). In *International Conference on Learning Representations*.

Zengzhuang Xu, Bingguang Hao, Zechuan Wang, Yuntao Wen, Yinyi Xu, Yang Liu, Long Chen, Dong Wang, Maolin Wang, Tong Zhao, Yicheng Chen, Cunyin Peng, Jinjie Gu, Leilei Gan, Xiangyu Zhao, Chenyi Zhuang, and Shi Gu. 2025. [FunReason-MT technical report: Advanced data synthesis solution for real-world multi-turn tool use](#). *Preprint*, arXiv:2510.24645.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.

Zichen Yuan, Lifan Sun, Yucen Zhuang, Yue Wang, Xinyuan Song, Tianqi Xu, Siyuan Li, Junchen Fu, Youhua Li, Sirui Hong, et al. 2025. [Solar: Serendipity optimized language model aligned for recommendation](#).

Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2026. [ToolACE-MT: Non-autoregressive generation for agentic multi-turn interaction](#). In *International Conference on Learning Representations*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. [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. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Xibin Zhou, Chenchen Han, Yingqi Zhang, Huan Du, Jiayuan Tian, Jin Su, Renju Liu, Kai Zhuang, Shiyu Jiang, Anthony Gitter, et al. 2025. [Decoding the molecular language of proteins with evolla](#). *bioRxiv*, pages 2025–01.

A Implementation Details

This appendix consolidates the implementation details. The fixed diagnostic partition is used for teacher-forced per-turn evaluation and failure logging, while the seed partition supplies trajectories both for prototype-conditioned synthesis and for the original supervised examples in refinement. BFCLv3 and BFCLv4 examples are used for evaluation only and do not enter synthesis or training. Table 7 lists the training, generation, and data-splitting settings used in the reported runs.

Table 7: Training and generation settings used by FUSE.

Item	Setting
Base model	Qwen3-4B-Instruct-2507
Seed trajectory pool	APIGen-MT-5k
Diagnostic / seed split	20% / 80%, fixed before refinement
Random seed	42
Evaluation data	BFCLv3 and BFCLv4; not used for synthesis or training
Refinement cycles	Three diagnostic-synthesis-SFT cycles
Training framework	Llama-Factory (Zheng et al., 2024)
Optimizer	AdamW (Loshchilov and Hutter, 2019)
Learning rate	2×10^{-5}
Epochs	3
Per-device batch size	2
Gradient accumulation	8
Scheduler / warmup	Cosine schedule, 3% warmup
Weight decay / max grad norm	0.0 / 1.0
Precision	bf16
Sequence cutoff	8192 tokens
Prototype classifier	gpt-5-mini, closed-label classification
Scenario generator	gpt-5-mini, default API decoding
Weighting	$\lambda = 1.0$, mean-normalized weighted resampling

B Prompt Examples

We use compact, structured prompts for prototype assignment and hard-scenario generation. Tables 8 and 9 show complete, lightly compacted examples.

Table 8: Prompt example for prototype classification.

You are classifying an already-detected function-calling failure. Choose exactly one label: P1, P2, P3, P4, P5, or P6. P1: missing-tool fallback failure. P2: underspecified required parameter. P3: argument grounding error. P4: wrong function selection. P5: long-context constraint loss. P6: other. Dialogue prefix: {dialogue_prefix} Visible function schemas: {function_schemas} Model output: {model_output} Reference action: {reference_action} Rule-check error: {low_level_error} Return only JSON: {"prototype": "...", "reason": "..."}.
--

Table 9: Prompt example for prototype-conditioned hard-scenario generation.

Create one multi-turn function-calling scenario for the target failure prototype. Target prototype: {prototype_id} Prototype-specific transformation: {prototype_transformation} Seed trajectory: {seed_trajectory} Visible function schemas: {function_schemas} Requirements: Keep the domain and interaction style close to the seed. Add a stressor that matches the target prototype. Do not use functions or arguments outside the visible schemas. Provide the correct reference assistant action for each assistant turn. If the correct behavior is fallback or clarification, do not invent a function call. Return only JSON with fields: tools, dialogue, reference_actions, target_prototype, rationale.
