Disambiguation-Centric Finetuning Makes Enterprise Tool-Calling LLMs More Realistic and Less Risky

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly tasked with invoking enterprise APIs, yet they routinely falter when near-duplicate tools vie for the same user intent or when required arguments are left underspecified. We introduce DIAFORGE (Dialogue Framework for Organic Response Generation & Evaluation), a disambiguation-centric, three-stage pipeline that (i) *synthesizes* persona-driven, multi-turn dialogues in which the assistant must distinguish among highly similar tools, (ii) performs supervised *fine-tuning* of open-source models with reasoning traces across 3 B–70 B parameters, and (iii) *evaluates* real-world readiness via a dynamic suite that redeploys each model in a live agentic loop and reports end-to-end goal completion alongside static conversational metrics. On our dynamic benchmark DIABENCH, models trained with DIAFORGE raise tool-invocation success by **27 pp**[1] over GPT-4o and by **49 pp** over Claude-3.5-Sonnet, both under optimized prompting. To spur further research, we release an open corpus of $\sim$5,000 production-grade enterprise API specifications paired with rigorously validated, disambiguation-focused dialogues, offering a practical blueprint for building reliable, enterprise-ready tool-calling agents.

1 Introduction

Modern enterprises manage *thousands* of APIs, often minor variants of a core functionality customized to serve distinct domains such as customer support, finance, and supply chain operations. As LLM assistants mature from conversationalists into *operational agents*, they must invoke these APIs with the same reliability that traditional software enjoys. In practice, however, single-turn user requests rarely arrive ready for direct invocation of enterprise tools: they may omit mandatory arguments, embed company-internal shorthand, or correspond to several near-duplicate tools. As Figure 1 shows, a single business query frequently surfaces multiple near-duplicate tool candidates. In our *production* telemetry, $\sim$**35-38%** of queries retrieve highly similar distractor APIs that require disambiguation (Appendix A.4); $\sim$**71%** of live APIs declare required parameters, and $\sim$**76-81%** of calls to those APIs arrive missing at least one required field. Consequently, a competent LLM assistant must master two intertwined capabilities: **multi-turn dialogue** to elicit missing arguments, and **fine-grained tool disambiguation** over a dense, overlapping API surface, often under noise and incomplete information. We address this with a disambiguation-focused pipeline for synthetic data generation and model training, empowering agents to ask targeted clarifying questions and issue accurate tool calls.

Tool-use benchmarks such as BFCL, ToolBench, and API-Bank evaluate models against *fixed* user scripts, treating incoming user queries as fully specified. Each test case supplies pre-written dialogue turns, and no additional user input is generated once the assistant responds (Yan et al., 2024; Qin et al., 2024; Li et al., 2023; Guo et al., 2024). This off-policy setup obscures a common enterprise failure mode: under-specified requests that demand iterative back-and-forth to disambiguate near-duplicate tools and fill in missing arguments. Because static tool-use suites cannot surface the cascading-error phenomenon observed in such disambiguation-centric multi-turn exchanges (Laban et al., 2025), our synthetic corpus intentionally withholds key details mid-dialogue and populates the tool list with semantically proximate alternatives, obliging the assistant to engage in dialogues rich in adaptive clarification. We pair model training with a dynamic evaluation harness that emulates a corporate user persona, tracking whether the model ultimately selects the correct tool and supplies required arguments. For completeness we still report static evaluation scores, but we emphasize

[1]"pp" = absolute percentage-point difference.

their comparatively limited diagnostic value.

User Query: I want to modify the details of a return order from one of our customers.

Retrieved Tools

- **update_customer_order_granular:** Allows for granular control over the textual content associated with individual items in a return order.
- **update_customer_order_header:** Facilitates the update of a specific text entry located at the header level of a customer return order.
- **get_customer_order:** Retrieves detailed text associated with a specific item in a customer return order.
- **delete_customer_order_header:** Deletes a specific text entry associated with a customer return item on a header level.
- **update_returned_inspections:** Update the data of items returned for inspection, which includes their status, inspection results, and any associated comments
- **delete_customer_order_value:** Facilitates the deletion of specific value-added service entries associated with a customer's return order.

Figure 1: A routine business query can retrieve multiple near-duplicate tools, illustrating the need for fine-grained disambiguation before tool invocation.

2 Related Work

LLMs as Tool-Using Agents. Pioneering works such as REACT interleave chain-of-thought (CoT) with tool calls, gathering evidence mid-dialogue and curb hallucinations (Yao et al., 2023). HUGGINGGPT generalizes this idea by casting LLM as a planner (Shen et al., 2023). These works establish language as a universal control interface for heterogeneous tools and motivate subsequent efforts to tune open models for reliable function calling.

Fine-Tuning LLM for Tool Use. TOOLFORMER shows that a self-supervised annotation pipeline enables LLMs to learn when and how to invoke external utilities (Schick et al., 2023). GORILLA augments LLMs with API-doc retrieval, surpassing GPT-4 on tool call accuracy (Patil et al., 2024). These results imply that curated data and retrieval augmentation, not sheer parameter count, are the present keys to dependable LLM tool use.

Benchmarks on LLM Tool Use. Most widely used multi-turn benchmarks evaluate exact function call accuracy based on pre-scripted dialogues (Li et al., 2023; Yan et al., 2024; Qin et al., 2024; Guo et al., 2024). Recent *interactive* suites broaden the evaluation scope: τ-BENCH emulates full user–agent conversations (Yao et al., 2024); AGENTBENCH spans eight environments to test long-horizon decision-making (Liu et al., 2024b); MINT and TOOLSANDBOX leverage LLM-simulated user feedback (Wang et al., 2024; Lu et al., 2024). Most public benchmarks still overlook other enterprise-grade challenges, notably distinguishing among near-duplicate tools, proactively eliciting mandatory arguments, and detecting or preventing tool-call hallucinations, shortcomings our framework is expressly designed to remedy.

Data Generation and Verification. Verified synthetic corpora have emerged as a primary catalyst for recent gains in open-source function-calling models. APIGEN collects thousands of executable APIs and auto-generates verified conversation traces (Liu et al., 2024c). TOOLACE introduces a self-evolution synthesis pipeline (Liu et al., 2024a). DECRIM employs a decompose–critique–refine loop (Ferraz et al., 2024). These pipelines illustrate a field-wide shift from brute-force scaling toward quality-controlled data generation driven by hierarchical feedback and automatic verification.

Ambiguity Resolution. *Premature* tool invocation in response to ambiguous or underspecified requests remains an understudied failure mode for tool-augmented LLMs, especially in high-stakes enterprise settings where tool misuse can introduce significant risk. CLARIFY-WHEN-NECESSARY formalizes when to ask versus act (Zhang and Choi, 2023). CLAMBER shows that CoT-enhanced LLMs still *over-estimate* their certainty and rarely spot ambiguity (Zhang et al., 2024). These observations motivate our explicit disambiguation routines.

3 Proposed Methodology

Our goal is to build *enterprise-grade* tool-calling LLMs that (i) accurately disambiguate near-duplicate tools and (ii) proactively request missing mandatory arguments, thereby mitigating the risk of hallucination-induced tool misuse. We present DIAFORGE, a three-stage pipeline encompassing synthetic dialogue generation (§3.1), supervised fine-tuning (§3.2), and dynamic evaluation (§3.3).

3.1 Synthetic Data Generation

We introduce UTC-GEN (<u>U</u>nified <u>T</u>ool-<u>C</u>alling <u>Gen</u>erator), a multi-agent engine to construct training dialogues in a *bottom-up* fashion. The engine executes three sequential phases: metadata construction, dialogue synthesis, and multi-view validation (Figure 2). Each dialogue trace is *seeded* with a ground-truth tool and is progressively enriched by specialized agent modules until it passes all validation gates. Implementation details appear in Appendix A.

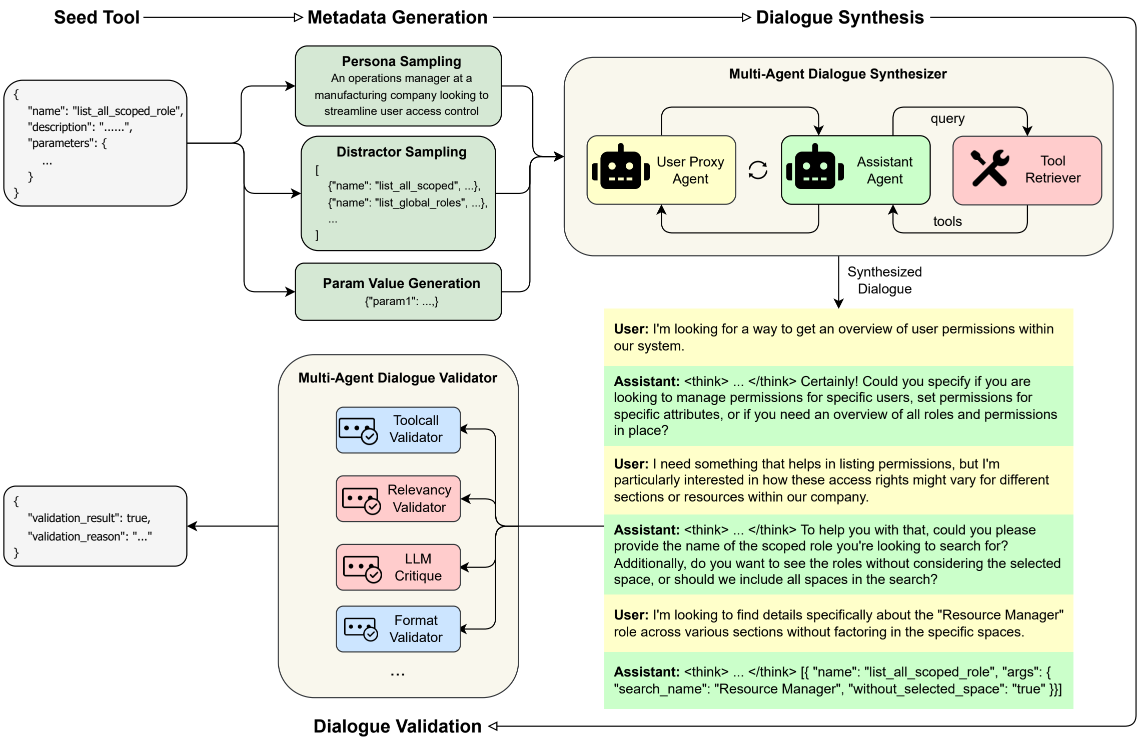

Figure 2: Data Generation Engine for Disambiguation-Centric **U**nified **T**ool-**C**alling Conversations (UTC-GEN)

Enterprise Tool Catalogue. Let

$$\mathcal{T} = \{\tau_i = (\text{name}_i, \text{description}_i, \texttt{params}_i)\}_{i=1}^{|\mathcal{T}|}$$

denote the enterprise-wide set of callable tools. For any tool τ_i, the parameter specification $\texttt{params}_i$ is a JSON Schema map that associates each argument name with a triple of the form $(\texttt{type}, \texttt{description}, \texttt{required})$. We define the set of *required* arguments for τ_i as $\mathcal{R}(\tau_i)$.

Persona Sampling. Given a seed tool $\tau^\star \in \mathcal{T}$, we first sample a corporate–user persona $p \sim \pi_{\text{rand}}^{(k)}(\cdot \mid \tau^\star, \mathcal{P})$, where $\pi_{\text{rand}}^{(k)}$ denotes a top-k retrieval-with-randomization distribution over an enterprise-filtered subset $\mathcal{P} \subseteq$ PERSONAHUB (12 k entries) (Ge et al., 2024). Conditioned on $(\tau^\star, p)$, we instantiate a concrete user goal $g \sim P_{\text{goal}}(\cdot \mid \tau^\star, p)$, which the user-proxy agent treats as its terminal objective during dialogue synthesis.

Distractor Tool Sampling. To emulate realistic tool ambiguity, we assemble a set of *near-duplicate* tools. Let $\phi : \mathcal{T} \to \mathbb{R}^d$ be a frozen sentence encoder applied to the concatenation of each tool's name, description, and selected schema metadata. We retrieve the $k = 5$ semantic nearest neighbors of $\tau^\star$, $\mathcal{D}_k(\tau^\star) = \arg\text{top-}k_{\tau \in \mathcal{T} \setminus \{\tau^\star\}} \langle \phi(\tau^\star), \phi(\tau) \rangle$. During dialogue synthesis, the assistant agent receives the candidate pool of tools $\mathcal{C}_k(\tau^\star) = \{\tau^\star\} \cup \mathcal{D}_k(\tau^\star)$, and must resolve any ambiguity *online* before issuing a tool call.

Slot Value Generator. We instantiate concrete, persona-consistent values for all *required* slots so that the user-proxy need not invent them on the fly. Let the required arguments for $\tau^\star$ be $\mathcal{R}(\tau^\star) = \{r_1, \ldots, r_m\}$. We *jointly* sample their values

$$(v_{r_1}, \ldots, v_{r_m}) \sim \mathcal{P}_{\text{param}}(\cdot \mid \mathcal{R}(\tau^\star), p),$$

where $\mathcal{P}_{\text{param}}$ is an LLM that, conditioned on the persona p, generates realistic parameter values, such as dates, currency codes, and alphanumeric IDs, with high diversity. Aggregating the draws yields the map $\mathcal{V}^\star = \mathcal{V}(\tau^\star, p) = \{(r_i, v_{r_i})\}_{i=1}^{m}$. During conversation simulation, the user-proxy incrementally reveals *subsets* of $\mathcal{V}^\star$, requiring the assistant to (i) identify disclosed values and (ii) query for any that remain unknown.

Dialogue Synthesis Given a tool $\tau^\star$, persona p, distractor set $\mathcal{D}_k(\tau^\star)$, and gold argument map $\mathcal{V}^\star$, UTC-GEN synthesizes a dialogue trace $d = \langle (u_1, a_1), (u_2, a_2), \ldots, (u_T, a_T) \rangle$, where u_t (resp. a_t) denotes the user (resp. assistant) utterance at turn t. Two running histories are maintained:

$$\mathbf{h}_t^u = (u_1, a_1, \ldots, u_{t-1}, a_{t-1}),$$
$$\mathbf{h}_t^a = (u_1, a_1, \ldots, u_{t-1}, a_{t-1}, u_t),$$

representing the context observable to the *user* and *assistant*, respectively, during turn t.

User Agent. At turn t, the user-proxy samples

$$u_t \sim P_{\boldsymbol{\theta}_u}\Big(\cdot \mid \tau^\star,\ p,\ g,\ \mathcal{D}_k,\ \mathcal{V}^\star,\ \mathbf{h}_t^u\Big),$$

where $P_{\boldsymbol{\theta}_u}$ is the distribution induced by the user-proxy's parameters $\boldsymbol{\theta}_u$. The persona p incorporates domain-specific jargon and tone reflective of authentic enterprise interactions, while g denotes the goal of the conversation; the distractor pool $\mathcal{D}_k$ steers user queries toward phrasing that could match several tools, compelling the assistant to disambiguate in real time; the gold argument map $\mathcal{V}^\star$ bounds all slots to ground truth values, mitigating hallucination; the running history $\mathbf{h}_t^u$ preserves discourse coherence with the dialogue prefix.

The user–proxy proceeds in two successive phases, coercing the assistant to *first* resolve tool ambiguity and *then* guarantee slot completion:

(i) *Tool-selection phase.* During opening turns, the user-proxy issues a deliberately under-specified request, revealing just enough context to prune the candidate set $\mathcal{C}_k$ until the assistant can unambiguously identify $\tau^\star$.

(ii) *Argument-completion phase.* After identifying $\tau^\star$, the user-proxy discloses the remaining slot values following the assistant's requests, until every key–value pair in $\mathcal{V}^\star$ has been provided.

Assistant Agent. At turn t,

$$a_t \sim P_{\boldsymbol{\theta}_a}\Big(\cdot \mid \mathcal{C}_k,\ \mathbf{h}_t^a\Big),$$

where $P_{\boldsymbol{\theta}_a}$ is the distribution induced by the assistant LLM's parameters $\boldsymbol{\theta}_a$; $\mathbf{h}_t^a$ is the dialogue prefix visible to the assistant at turn t; $\mathcal{C}_k$ is the set of candidate tools. Because the assistant is *oblivious* to which element of $\mathcal{C}_k$ is the ground-truth $\tau^\star$, it must (i) pose clarification questions that iteratively eliminate distractors, and (ii) solicit any missing slot values until the argument map is complete.

Each assistant turn is decomposed into a private *reasoning trace* and a public *response*: the former captures chain-of-thought computations internal to the model, while only the latter is revealed to the user-proxy agent. During supervised fine-tuning (§3.2), both components serve as learning targets.

Stopping Criteria. The simulation terminates as soon as *one* of the following events occurs:

(i) the assistant emits a schema-conformant call to $\tau^\star$ whose arguments map exactly to $\mathcal{V}^\star$, with no missing *or* superfluous keys;

(ii) the dialogue length reaches the hard cap $T_{\max}$.

Validator Cascade. The synthesized dialogue d enters the training corpus only if every turn stands scrutiny by a *format validator*, *relevancy validator*, and *LLM critique* (Figure 2); failure at any step triggers immediate rejection (Appendix A.2).

(a) **User-Utterance Validity**. For every turn $t \in \{1, \ldots, T\}$, the user message u_t remains
  (i) coherent with the dialogue prefix $\mathbf{h}_t^u$;
  (ii) grammatically intelligible and stylistically faithful to the sampled persona p;
  (iii) semantically aligned with latent goal g.

(b) **Assistant-Response Validity**. For every turn t, the assistant reply a_t
  (i) contains a json schema object with *three* sections: a *thought* trace, an optional *tool_calls* stub, and a public *content*;
  (ii) is coherent with the dialogue prefix $\mathbf{h}_t^a$.

Each dialogue must also contain one assistant turn $t^\dagger \leq T$ whose *tool_calls* satisfies stopping criterion (i). Only dialogues that pass *all* validation checks are included in the final training set.

3.2 Fine-Tuning Pipeline

Let the validated corpus be $\mathcal{D}_{\text{train}} = \{d_i\}_{i=1}^N$, with

$$d_i = \big\langle (u_1^{(i)}, a_1^{(i)}), \ldots, (u_{T_i}^{(i)}, a_{T_i}^{(i)}) \big\rangle.$$

We adopt a *turn-slicing* strategy (Ouyang et al., 2022) (Figure 10): for each assistant turn $t \in \{1, \ldots, T_i\}$ we form an input–target pair

$$x_{i,t} = \underbrace{\texttt{[SYS]}\ u_1^{(i)}\ a_1^{(i)}\ \ldots\ u_t^{(i)}}_{\text{prompt context}}, \qquad y_{i,t} = a_t^{(i)}.$$

The model is trained *only* to predict the next assistant response, given the complete dialogue prefix.

We perform standard Supervised Fine-Tuning (SFT) with LoRA (Hu et al., 2022) over next token prediction (Appendix B). While training, we perform loss masking for contextual tokens such that only the tokens in the completion part of the sample are learned. This formulation ensures that the model learns to produce a contextually coherent assistant response given the entire preceding dialogue history, without diluting the gradient on earlier turns.

3.3 Evaluation Protocol

We evaluate a fine-tuned LLM f_ϕ along two complementary axes: Static evaluation (isolated response quality) and Dynamic evaluation (end-to-end interactive robustness).

The dialogues produced by the assistant LLM f_ϕ are evaluated with four classes of conversation-level metrics: (i) tool-calling and parameter-filling accuracy (ACC); (ii) failure measures (FTR, TAR); (iii) auxiliary metrics: tool-call precision/recall (TCP, TCR) and parameter-key precision/recall (PKP, PKR); and (iv) semantic-fidelity metrics, comprising conversation relevancy (CONVREL), type–token ratio (TTR), and n-gram diversity (NGD). Verbal definitions of these metrics are given in Section 4, and their complete mathematical formulations appear in Appendix C.1.

Static Evaluation In static evaluation, we sequentially decode each assistant turn $\hat{a}_t = f_\phi(u_{\leq t}, \hat{a}_{<t}; \mathcal{C}_k)$, leaving user utterances intact. Static evaluation is deterministic, inexpensive, and isolates the model's ability to emit "correct" replies under perfect user prompts; however, it cannot capture how the assistant's outputs would influence subsequent user behavior in an interactive setting.

Dynamic Evaluation To gauge *on-policy* conversational competence, the fine-tuned model $f_{\boldsymbol{\phi}}$ is inserted as the *assistant agent* inside the full UTC-GEN loop (Figure 2); the user-proxy policy $P_{\boldsymbol{\theta}_u}$ remains frozen (cf. §3.1). The interaction unfolds for at most $T_{\max}$ turns, yielding a trajectory

$$d_{f_{\boldsymbol{\phi}}} = \big\langle (\hat{u}_1, \hat{a}_1), (\hat{u}_2, \hat{a}_2), \ldots, (\hat{u}_{T'}, \hat{a}_{T'}) \big\rangle,$$

with $T' \leq T_{\max}$. At turn t, the assistant observes the dialogue prefix $\hat{\mathbf{h}}_t^a = (\hat{u}_1, \hat{a}_1, \ldots, \hat{u}_t)$ together with the candidate-tool set $\mathcal{C}_k$ and generates $\hat{a}_t = f_{\boldsymbol{\phi}}(\hat{\mathbf{h}}_t^a; \mathcal{C}_k)$. This rollout measures the model's ability to maintain contextual coherence, self-correct earlier reasoning errors, and issue schema-conformant tool calls.

4 Experiments

We fine-tune six publicly available, instruction-tuned, decoder-only language models: Llama-3.2-3B, Gemma-3-4B, Gemma-3-12B, Gemma-3-27B, Llama-3.3-Nemotron-Super-49B, Llama-3.3-70B.

Training Configuration All models are fine-tuned exclusively on the 5,000 DiaFORGE conversations, yielding 13,649 turn-sliced completion samples generated by the data engine illustrated in Figure 2. No additional general-domain SFT data is incorporated. Each base model is trained for a single epoch using the AdamW optimizer (Loshchilov and Hutter, 2017). Complete hyperparameter settings and an annotated training sample are provided in Appendix B.

Evaluation Setting We evaluate and compare the performance of our fine-tuned models against several baselines: non-fine-tuned models, closed-source models such as *GPT-4o* and *Claude-3.5-Sonnet*, and *Llama-xLAM-2-70b-fc-r*, the state of the art for function calling at the time of writing this paper according to BFCL V3 (Yan et al., 2024). For non-fine-tuned and closed-source models, we apply system prompt optimization using Cost-Aware Prompt Optimization (CAPO) (Zehle et al., 2025), the state-of-the-art prompt optimization method at the time of writing (Appendix E).

Our evaluation benchmark, DIABENCH, comprises 119 seed tools, each paired with corresponding multi-turn, reasoning-annotated dialogues. The benchmark is built from a *proprietary*, out-of-domain corpus tied to a production assistant and includes held-out, out-of-distribution line-of-business (LoB) tools spanning backend APIs and UI-triggered operations that never appear in training data. Section 4.2 details the statistical analysis and its difference with the training data for reproducibility purposes. Experiments employ both the **static** and **dynamic** protocols defined in §3.3.

Evaluation Metrics. We track dialogue-level measures for each simulated conversation. *Accuracy Rate* (ACC) is the proportion of multi-turn dialogues in which the assistant's first tool invocation (i) correctly selects the reference tool $\tau^\star$ and (ii) supplies the complete, yet no superfluous, set of required key–value arguments. *False-Positive Tool-call Rate* (FTR) captures any instance where the assistant takes an unwarranted action such as invoking a distractor tool, hallucinating a non-existent endpoint, or issuing multiple tool calls when only one is appropriate. *Tool-call Abstention Rate* (TAR) captures the converse failure mode: cases where a dialogue concludes without any tool invocation, signaling that the model failed to recognize when tool use was necessary. Together, FTR and TAR directly quantify failures

Model	Static Evaluation			Dynamic Evaluation		
	ACC (↑)	FTR (↓)	TAR (↓)	ACC (↑)	FTR (↓)	TAR (↓)
Llama-3.2-DiaFORGE-3B (Grattafiori et al., 2024)	0.52	0.12	0.30	0.80	0.08	0.06
Llama-3.3-70B	0.03	**0.00**	0.97	0.11	**0.02**	0.88
Llama-3.3-70B-fc	0.22	0.52	**0.01**	0.30	0.22	**0.01**
Llama-3.3-DiaFORGE-70B	0.38	0.03	0.58	0.79	0.03	0.15
Llama-xLAM-2-70B-fc-r (Prabhakar et al., 2025)	0.48	0.18	0.13	0.51	0.18	0.05
Llama-3.3-Nemotron-Super-49B (Bercovich et al., 2025)	0.60	0.07	0.25	0.72	0.08	0.08
Llama-3.3-Nemotron-DiaFORGE-49B	**0.82**	0.04	0.12	**0.89**	0.06	0.03
Gemma-3-4B (Kamath et al., 2025)	0.19	0.17	0.61	0.24	0.14	0.58
Gemma-3-DiaFORGE-4B	0.53	0.05	0.37	0.81	0.09	0.05
Gemma-3-12B	0.31	0.03	0.62	0.37	0.04	0.57
Gemma-3-DiaFORGE-12B	0.68	**0.02**	0.26	0.86	0.07	0.07
Gemma-3-27B	0.19	0.02	0.78	0.21	**0.00**	0.79
Gemma-3-DiaFORGE-27B	**0.77**	0.03	**0.18**	**0.89**	0.03	**0.03**
GPT-4o-20241120 (Hurst et al., 2024)	0.19	0.00	0.81	0.62	0.02	0.36
GPT-4o-20241120-fc	0.61	0.64	0.16	0.56	0.59	0.05
Claude-3.5-Sonnet-20241022 (Anthropic, 2024)	0.15	0.02	0.82	0.39	0.03	0.55
Claude-3.5-Sonnet-20241022-fc	0.42	0.76	0.03	0.40	0.34	0.03

Table 1: Evaluation Results on Tool Call Accuracy and Failure Modes. **All open-source models evaluated are instruction-tuned, decoder-only LLMs.** Models with the suffix "*fc*" support native function/tool calling, while all other models are evaluated using CAPO-optimized system prompts.

in tool disambiguation, a core aspect of our evaluation (see Appendix C). To assess dialogue-level quality beyond tool usage, we further report three complementary metrics: *conversation relevancy* (CONVREL), *type–token ratio* (TRR), and *n-gram diversity* (NGD). Formal definitions for all metrics appear in Appendix C.1.

User Agent in Dynamic Evaluation In dynamic evaluation (§3.3), the LLM acting as the user-proxy agent is susceptible to hallucinations (Huang et al., 2025), which can cause cascading failures in dialogue generation. Such conversations are unsuitable for assessing the assistant model, as failures may stem from user-side hallucinations rather than assistant shortcomings. To mitigate this, we adopt a multi-sampling and voting strategy to generate each user utterance, enhancing stability and reducing evaluation noise. To generate each user utterance, we sample **3** candidate responses from the same LLM. A separate voting LLM then selects the best response among them. For the evaluations reported in Table 1, we use differently prompted instances of GPT-4o for sampling and voting. A comparative analysis of alternative sampling models is provided in Appendix C.3. Finally, all conversations generated during dynamic evaluation are manually reviewed by domain experts to detect hallucinations introduced by the user-proxy agent. We observe a user-proxy hallucination rate below **1%** across all samples; these instances are excluded prior to computing the final evaluation results.

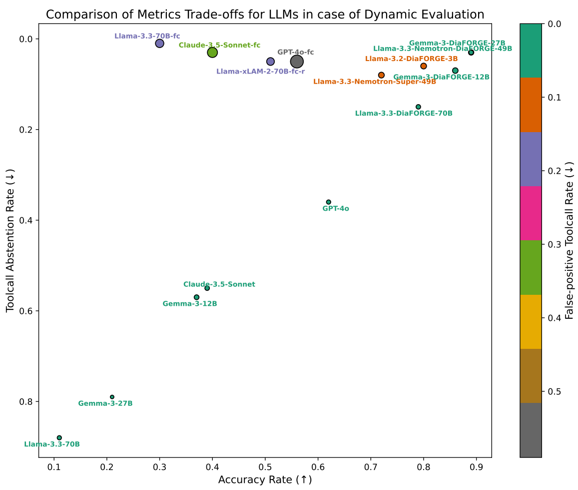

Figure 3: Trade-offs among tool call-related metrics under *Dynamic Evaluation*. Marker size & Color ∝ FALSE-POSITIVE TOOL-CALL RATE (FTR). Models closer to the upper right are preferable; those in the lower left underperform across metrics.

In Table 1, we compare all evaluated models using three tool call-related metrics: ACC, FTR, and TAR. These metrics collectively assess an LLM's ability to invoke tools reliably in realistic, multi-turn settings. ACC measures correctness, FTR captures incorrect tool calls, and TAR reflects the risk of failing to complete the tool-calling objective within the dialogue. For an LLM to be viable in an industry setting, mitigating the risks of insufficient disambiguation, it must balance the three metrics while demonstrating reliability on each. Figure 3

Setting	Static Evaluation			Dynamic Evaluation		
	ACC (↑)	FTR (↓)	TAR (↓)	ACC (↑)	FTR (↓)	TAR (↓)
Gemma-3-27B	0.19	**0.02**	0.78	0.21	**0.00**	0.79
Gemma-3-DiaFORGE-27B	**0.77**	0.03	**0.18**	**0.89**	0.03	**0.03**
w/o Validation Cascade	0.46	0.04	0.47	0.56	0.06	0.35
w/o Near-Duplicate Distractor Sampling	0.52	0.10	0.34	0.63	0.18	0.19
w/o Thinking Traces	0.62	0.04	0.30	0.77	0.16	0.04

Table 2: Ablation study on **Gemma-3-DiaFORGE-27B**: each variant removes one UTC-GEN component.

illustrates the trade-offs among these metrics for different models. We observe that models trained with DIAFORGE achieve high ACC while simultaneously minimizing both FTR and TAR. Detailed evaluation results for more granular metrics (TCP, TCR, PKP, PKR, etc.) appear in Appendix C.2.

At a production scale of 10 k tool-call-eligible conversations per day, even modest differences between LLMs compound into large operational deltas: a GPT-4o-fc configuration yields **5,500–6,000** erroneous tool calls per day, resulting in substantial remediation and infrastructure overhead, whereas a GPT-4o (prompt) configuration tends to abstain, stalling **3,500–3,800** conversations per day. Both patterns degrade user experience and raise costs, increasing churn. In contrast, DIAFORGE-tuned models reduce total failures to **250–350** per day, lowering erroneous calls and stalls. Appendix D provides a real production case study showing impact of such finetuning.

Beyond accurate tool invocation, our use case demands that models also sustain coherent, human-like dialogue throughout the interaction. This includes maintaining context and responding naturally to human users. To assess these capabilities, we report additional metrics related to conversational handling in Appendix C.2.

4.1 Ablation Study

We run ablations on **Gemma-3-27B**, holding constant data volume, LoRA recipe, and evaluation protocol; each variant removes exactly one UTC-GEN component to isolate its contribution to disambiguation in the fine-tuned model:

- **Without Validation Cascade:** removes the rule-based and LLM validators; synthetic dialogues enter the training corpus unvalidated.
- **Without Near-Duplicate Distractor Sampling:** removes near-duplicate distractor tools from retrieved tool sets; eliminates turns devoted to fine-grained tool disambiguation.
- **Without Thinking Traces:** removes the assistant's reasoning traces during fine-tuning and decode without thinking at inference.

Table 2 compares *Gemma-3-DiaFORGE-27B* with the vanilla backbone and three ablations. The DIAFORGE model attains high ACC, reducing abstention and keeping erroneous calls low. Dropping the *validation cascade* admits schema-invalid or tool-absent turns into training, regressing performance and inflating TAR. Removing *near-duplicate distractor sampling* weakens supervision for fine-grained tool selection and degrades performance on DIABENCH, which explicitly stresses near-duplicate disambiguation. Ablating *thinking traces* reduces ACC, increasing both erroneous invocations and unnecessary abstentions. Collectively, these results show that validation filters keep training data clean, near-duplicate sampling teaches disambiguation, and reasoning traces calibrate inference toward correct actions.

4.2 Comparing DIABENCH with Public Benchmarks

BFCL v3 BFCL (Yan et al., 2024) is a general function-calling benchmark that evaluates both native and prompt-induced tool calling. It spans five high-level categories with 17 subcategories; only **7** subcategories present multiple tools in the retrieval set (LLM context). Across the multi-tool test cases (**41.2%** of all samples), the average number of distractor tools is ∼**1.6** with an average semantic overlap of **0.24**. Only **0.57%** of all test cases contain at least one distractor that is a *near-duplicate* of the ground-truth tool (similarity criterion in Appendix A.4). Furthermore, BFCL V3 adopts a *static* evaluation protocol that follows fixed conversation scripts and assumes fully specified queries, leaving no scope for the model to ask clarifying questions.

Other function-calling benchmarks like ToolBench (Qin et al., 2024) & API-Bank (Li et al., 2023) also follow similar pattern (static scripts with fully specified queries) and does not evaluate disambiguation capability of the model. To the best of our knowledge, there is no open benchmark yet that evaluates LLM's ability to ask clarifying questions. Thus, we do not expect our finetuned model to outperform on these public benchmarks. However, in order to verify that the finetuning doesn't lead to catastrophic forgetting, we perform parity checking of the DIAFORGE-finetuned model w.r.t instruction-tuned base model on BFCL V3 (Yan et al., 2024) and MT-BENCH (Zheng et al., 2023). The evaluation results (Appendix C.5) suggests no significant degradation in model's general abilities when finetuned with DIAFORGE pipeline.

DIABENCH By contrast, DIABENCH is expressly designed to test an LLM's ability to pose clarifying questions and disambiguate among near-duplicate tools in *enterprise setting*. **100%** of test cases include multiple tool options in context, with an average of $\sim$**5.2** distractor tools per case and an average retrieval-set semantic overlap of **0.47**. Additionally, **29.2%** of all test cases contain distractors that are near-duplicates of the ground-truth tool (same criterion as Appendix A.4), and **75.6%** of test cases have parameterized ground truth tools requiring the assistant LLM to ask a clarifying question to obtain missing required parameters. This distribution is intentional and mirrors production traffic. Beyond static scoring, we also employ a *dynamic* protocol that redeploys each post-trained model in an agentic loop to assess whether it proactively solicits missing information via clarifying questions and subsequently executes the correct tool call.

User: I'm trying to get a good handle on the materials and components we use across various manufacturing sites. Is there something that can help me with that?

Assistant: <think> ... </think> I can help you with that by providing a comprehensive Bill of Materials (BOM) list. Are you interested in looking at any particular one or for all plants?

User: I'd like to focus on the BOM data specifically for our manufacturing facility, plant PLT-4721.

Assistant: <think> ... </think> Thank you for providing information. Can you please also provide material or product id?

User: Sure, I'm looking at the material with the code AG9F76B1XZ

Assistant: <think> ... </think> [{"name": "list_bom", "args": {"plant_id": "PLT-4721", "material_id": "AG9F76B1XZ"}}]

Figure 4: Example DIAFORGE training dialogue generated with the UTC-GEN user proxy.

User: I need to view some detailed information related to the components required for a specific production item.

Assistant: <think> ... </think> Could you please specify if you are looking for the complete BOM details or if you are interested in analyzing the BOM for a particular plant?

User: for a particular plant.

Assistant: <think> ... </think> Could you please provide the Plant and Material details for which you want to analyze the BOM?

User: PLT-4721 and AG9F76B1XZ.

Assistant: <think> ... </think> [{"name": "list_bom", "args": {"plant_id": "PLT-4721", "material_id": "AG9F76B1XZ"}}]

Figure 5: Example dialogue from dynamic evaluation: the user proxy LLM is prompted to respond concisely; the assistant LLM is the fine-tuned GEMMA-3-DIAFORGE-27B.

Unlike BFCL V3, DIABENCH comprises complex business functions spanning backend APIs from *held-out* lines of business (LoBs), high-level workflow functions, and UI-triggering functions. Our production environment covers nine LoBs with thousands of backend APIs. For training, we stratified-sample backend APIs from five LoBs. For evaluation, DIABENCH is constructed from LoB functions *disjoint* from those used in training and includes workflow/UI functions that **never** appear in the training corpus.

User-proxy prompting intentionally differs between the training corpus and DIABENCH. As shown in Figure 4, the training user proxy (via UTC-GEN) produces verbose, often ambiguous utterances, whereas in DIABENCH dynamic evaluation (Figure 5) the dedicated user proxy is prompted to issue concise, command-like queries. This deliberate style shift induces out-of-distribution, terse inputs that stress an LLM's ability to ask targeted clarifying questions, ultimately select the correct tool, and closely reflects the interaction patterns seen in real user behavior with our AI assistant in production.

5 Conclusion

We introduce DIAFORGE, a modular three-stage pipeline that (i) synthesizes high-quality, multi-turn tool-calling dialogues designed to stress the disambiguation behaviors where current LLMs still struggle, (ii) enables efficient supervised fine-tuning across models of varying scales, and (iii) provides both static and dynamic evaluation tailored to realistic multi-turn tool use in enterprise settings. To spur further research on robust, real-world tool-calling agents, we publicly release a dataset of roughly 5,000 production-grade enterprise APIs paired with their DIAFORGE-curated dialogues.

Limitations

DiaFORGE's *disambiguation-centric* data synthesis paradigm provides a principled foundation for aligning tool invocation with user intent, yet several open challenges remain, which we plan to explore as future work.

Our post-training setup assumes the ground-truth tool is present in the retrieved candidate set: an assumption that does not always hold in production. Future work will incorporate hard negatives and explicit "no-tool" dialogues to train the agent to refrain from using tools in such cases.

DiaFORGE uses LLM-based validators to filter unrealistic dialogues, yet these validators can exhibit biases, hallucinate, or miss edge cases. Moreover, the current generators do not yet cover the full breadth of complex enterprise interactions. Strengthening diversity via more robust ensemble validation and expanding generator coverage is a key direction for future work. Meanwhile, extending DiaFORGE to synthesize multi-tool, multi-step, disambiguation-aware conversations would further improve data realism and furnish a more rigorous benchmark of an LLM's ability to plan, sequence, and recover across near-duplicate tools.

Although dynamic evaluation is overall a better strategy to evaluate conversational LLMs, we still require human validation to discard dialogues where the simulated user hallucinates. Such manual validation of the synthesized dialogues during dynamic evaluation is expensive & hard to scale, especially in an industry setting. Moreover, while our multi-sampling voting strategy tries to minimize the user-proxy hallucination, it leads to an increase in cost due to multiple LLM calls.

Ethical Considerations

We conducted experiments within the provisions of the ACL Ethics Policy and relevant research-integrity guidelines. There are, to the best of our knowledge, no remaining ethical risks that have not been addressed.

References

Anthropic. 2024. Claude 3.5 sonnet model card addendum. `https://www.anthropic.com/news/claude-3-5-sonnet`.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, and 114 others. 2025. Llama-nemotron: Efficient reasoning models. *Preprint*, arXiv:2505.00949.

Thomas Palmeira Ferraz, Kartik Mehta, Yu-Hsiang Lin, Haw-Shiuan Chang, Shereen Oraby, Sijia Liu, Vivek Subramanian, Tagyoung Chung, Mohit Bansal, and Nanyun Peng. 2024. Llm self-correction with decrim: Decompose, critique, and refine for enhanced following of instructions with multiple constraints. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7773–7812.

Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. 2024. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*.

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

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11143–11156.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

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*.

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2025. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A benchmark dataset for real-world apis. In *Proceedings of the*

2023 Conference on Empirical Methods in Natural Language Processing, pages 3102–3116, Singapore.

Weiwen Liu, Xu Huang, Xingshan Zeng, Yuxian Wang, Xin Jiang, and Enhong Chen. 2024a. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Xiao Liu, Hao Yu, Hanchen Zhang, and Jie Tang. 2024b. Agentbench: Evaluating large language models as agents. In *International Conference on Learning Representations (ICLR)*.

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

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2024. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*.

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

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

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, and 1 others. 2025. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*.

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 *Proceedings of the 12th International Conference on Learning Representations*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. *Preprint*, arXiv:1910.02054.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, 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 36 (NeurIPS 2023)*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Zhengyan Shi, Adam X Yang, Bin Wu, Laurence Aitchison, Emine Yilmaz, and Aldo Lipani. 2024. Instruction tuning with loss over instructions. *arXiv preprint arXiv:2405.14394*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *12th International Conference on Learning Representations, ICLR 2024*.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Tom Zehle, Moritz Schlager, Timo Heiß, and Matthias Feurer. 2025. Capo: Cost-aware prompt optimization. *arXiv preprint arXiv:2504.16005*.

Michael JQ Zhang and Eunsol Choi. 2023. Clarify when necessary: Resolving ambiguity through interaction with lms. *arXiv preprint arXiv:2311.09469*.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. 2024. Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10746–10766.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

A Details About Data Generation Engine

A.1 Dialogue Synthesis

The Multi-Agent Dialogue Synthesizer (Figure 2) generates synthetic dialogues in two stages, guided by dialogue state tracking to ensure coherence and goal alignment.

Tool Selection Stage. The user-proxy agent is assigned a goal and generates vague but contextually relevant utterances. Its context includes the seed tool $\tau^\star$ and a set of distractor tools $\mathcal{D}_k(\tau^\star)$. In this stage, the user-proxy agent is instructed to reveal minimal information initially, offering substantive details only in response to the assistant's clarifying questions in subsequent turns.

On the other hand, the assistant agent's objective in this stage is to identify the appropriate tool $\tau^\star$ by asking clarifying questions. It does not have direct access to $\tau^\star$ but instead queries a vector database via a live tool retriever to obtain a candidate set $\mathcal{C}_k$. We enforce that $\tau^\star \in \mathcal{C}_k$. If the condition is not satisfied, we discard the current conversation and regenerate a new one from scratch with the same seed tool, up to five attempts, to maximize its chances of being included in the training corpus.

Once a tool is selected, a rule-based validation will ensure that the selected tool $\tau = \tau^\star$. If $\tau \neq \tau^\star$, the dialogue sample is rejected and synthesis halts. If $\tau = \tau^\star$, the last assistant message is removed, and the process transitions to the parameter filling stage.

Parameter Filling Stage. Assuming the correct tool has been selected, the assistant agent now proceeds to collect the necessary parameters to execute the tool call. With the gold tool $\tau^\star$ provided in its context, the assistant is now tasked with eliciting all required argument values, whether stated explicitly or implied by the user-proxy, and invoking the tool once all required inputs have been gathered.

On the other hand, the user-proxy agent is given access to the ground-truth parameter values, represented as the argument map $\mathcal{V}^\star = \mathcal{V}(\tau^\star, p)$, and is instructed to provide the parameter values specifically requested by the assistant. If the selected tool requires no parameters, the assistant initiates the tool call immediately at the beginning of the parameter filling stage, without any additional input from the user-proxy.

Throughout both stages, assistant messages include internal reasoning traces generated using the *Reason First, Response Later* strategy. These traces are accessible only to the assistant and remain hidden from the user-proxy.

Generated Sample

Persona: A logistics operations manager seeking advice on mitigating transport risks
Seed: fn_1126_cloud_transport_management

User: I'm trying to improve how we handle different aspects of our transport processes. Could you assist me with finding the right approach or tool to manage and monitor our logistics tasks effectively?
Assistant: <think> ... </think> Could you please specify what aspect of logistics you are trying to manage? Are you focusing on tracking logistics orders, managing warehouse resources, or monitoring transport records?
User: I'm looking for a way to monitor and review all the actions taken during our transport activities. What options are available for keeping an eye on these processes?
Assistant: <think> ... </think> To assist you in monitoring and reviewing all actions taken during your transport activities, our designated tool can be very helpful. Before proceeding, could you please provide the Node ID and the Transport Request ID? These two pieces of information are necessary to retrieve the relevant logs and facilitate monitoring and troubleshooting efficiently.
User: Certainly. The Node ID is 437292, and the Transport Request ID is 957841. Let me know if there's anything else you need.
Assistant: <think> ... </think> [{"name": "fn_1126_cloud_transport_management", "args": {"nodeId": 437292, "transportRequestId": 957841}}]

Figure 6: DIAFORGE generated dialogue sample

An example of a synthesized dialogue is shown in Figure 6. The assistant strategically asks specific, targeted questions to progressively narrow down the tool selection. Once the correct tool is identified, it proceeds to elicit the necessary parameter values before issuing a final tool call. In real-world applications, such disambiguation capability is essential for function-calling models to be genuinely helpful and reliable in assisting enterprise users.

A.2 Dialogue Validation

Once the dialogues are synthesized, they are processed by the Multi-Agent Dialogue Validator, illustrated in Figure 2. This system comprises multiple validator agents, broadly categorized into two types.

Functional Validators. These are rule-based agents designed to enforce structural and logical constraints on the generated dialogue. Multiple functional validators are applied sequentially. FORMAT VALIDATOR ensures the dialogue follows the expected structure, alternating user and assistant turns, and that assistant messages include both reasoning traces and final responses. TOOLCALL VALIDATOR verifies that the dialogue ends with a valid tool call corresponding to the gold tool $\tau^\star$. TOOLARGS VALIDATOR checks that all required parameters for the tool call are correctly provided. Due to interdependencies among these checks, the functional validators are executed in the following order: FORMAT VALIDATOR $\rightarrow$ TOOLCALL VALIDATOR $\rightarrow$ TOOLARGS VALIDATOR.

LLM Validators. These are LLM-based agents responsible for validating aspects that require natural language understanding. Each validator is prompted with distinct instructions and assesses different aspects of the dialogue. RELEVANCY VALIDATOR evaluates whether the dialogue content is semantically relevant to the gold tool $\tau^\star$. LLM CRITIQUE assesses the overall flow of the conversation, ensuring it exhibits the expected two-stage structure, and checks that both agents (user and assistant) adhere to their designated roles. As the validators function independently, they are executed concurrently. A dialogue sample is rejected if any validator flags it as invalid, as all validators are considered equally authoritative.

Human Spot Checks. To complement automated validation, we periodically conduct human spot checks on random subsets of the validated dialogues, providing an additional quality-control layer and guiding prompt refinements when systematic issues are discovered.

A.3 Data Distribution

We present the distribution of the training data used in this study, which is identical to the dataset we release as part of our open-sourced benchmark.

Figure 7 illustrates the distribution of conversation lengths, measured by the number of dialogue turns. The majority of conversations contain fewer than five turns, aligning with typical session lengths observed in real-world enterprise tool-use scenarios. Figure 8 shows the distribution of the number of parameters associated with the seed tools for which the conversations were generated.

Figure 9 depicts the number of dialogue turns dedicated to tool disambiguation and parameter filling. In most cases, tool selection is completed within two turns, followed by a single turn for parameter specification. Notably, some samples contain zero turns for parameter filling: this occurs when the tool either requires no parameters or when parameters are provided during the tool selection phase, which reflects common patterns observed in real-world multi-turn enterprise interactions.

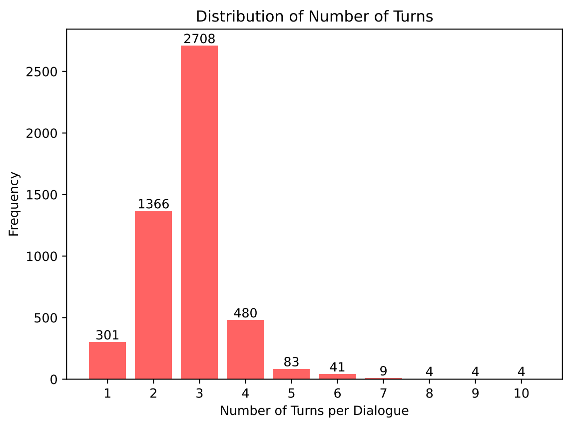

Figure 7: Conversation length distribution: number of dialogue turns per sample.

A.4 Analyzing Near-Duplicate Tools

We quantify near-duplication with a bounded, symmetric composite similarity metric. Each tool $\tau \in \mathcal{T}$ is represented as $(\texttt{tname}(\tau), \texttt{tdesc}(\tau), \texttt{params}(\tau))$, where $\texttt{params}(\tau)$ is a JSON Schema map from argument keys to $(\texttt{type}, \texttt{description}, \texttt{required})$. Define the set of required keys as $\texttt{keys}(\tau)$. For any argument $p \in \texttt{keys}(\tau)$, let $\texttt{type}_\tau(p)$ denote the normalized base type (e.g., $\texttt{string}$, $\texttt{integer}$, $\texttt{float}$, $\texttt{date}$, $\texttt{bool}$). Unless noted, all component similarities lie in $[0, 1]$ and are symmetric, and the

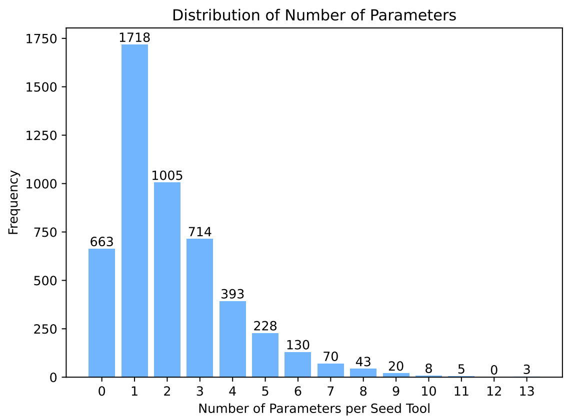

Figure 8: Parameter count distribution: number of parameters per seed tool.

composite score inherits these properties.

Composite similarity. For $\tau \neq \tau^\star$,

$$\begin{aligned} S(\tau^\star, \tau) = {} & w_{\text{tname}}\, S_{\text{tname}}(\tau^\star, \tau) \\ & + w_{\text{tdesc}}\, S_{\text{tdesc}}(\tau^\star, \tau) \\ & + w_{\text{param}}\, S_{\text{param}}(\tau^\star, \tau), \end{aligned}$$

with $w_{\text{tname}}, w_{\text{tdesc}}, w_{\text{param}} \in [0, 1]$ and $w_{\text{tname}} + w_{\text{tdesc}} + w_{\text{param}} = 1$. We use $w_{\text{tname}} = 0.40$, $w_{\text{tdesc}} = 0.35$, $w_{\text{param}} = 0.25$.

Name similarity. Let $\mathrm{LCS}(\cdot, \cdot)$ be the character-level longest common subsequence. With preprocessed names (lowercased), define

$$S_{\text{tname}}(\tau^\star, \tau) \;=\; \frac{2\ \mathrm{LCS}\big(\texttt{tname}(\tau^\star), \texttt{tname}(\tau)\big)}{|\texttt{tname}(\tau^\star)| + |\texttt{tname}(\tau)|}.$$

Description similarity. Let $\psi(\cdot)$ be a sentence encoder and define unit vectors $\mathbf{v}(\tau) = \psi\big(\texttt{tdesc}(\tau)\big) / \big\|\psi\big(\texttt{tdesc}(\tau)\big)\big\|_2$ so that $\|\mathbf{v}(\tau)\|_2 = 1$. Cosine lies in $[-1, 1]$; we rescale to $[0, 1]$:

$$S_{\text{tdesc}}(\tau^\star, \tau) \;=\; \frac{1 + \mathbf{v}(\tau^\star)^\top \mathbf{v}(\tau)}{2}.$$

Parameter similarity. Write $A = \texttt{keys}(\tau^\star)$, $B = \texttt{keys}(\tau)$, and $I = A \cap B$. Combine set overlap with type agreement:

$$S_{\text{set}}(\tau^\star, \tau) = \begin{cases} \dfrac{|A \cap B|}{|A \cup B|}, & \text{if } A \cup B \neq \varnothing, \\ 1, & \text{if } A = B = \varnothing. \end{cases}$$

$$S_{\text{type}}(\tau^\star, \tau) = \frac{1}{|I|} \sum_{p \in I} \mathbb{I}\big[\texttt{type}_{\tau^\star}(p) = \texttt{type}_{\tau}(p)\big].$$

Convention: if $I = \varnothing$, the sum is 0 and the ratio above is defined to be 0 (empty-average).

$$S_{\text{param}}(\tau^\star, \tau) = \tfrac{1}{2}\, S_{\text{set}}(\tau^\star, \tau) + \tfrac{1}{2}\, S_{\text{type}}(\tau^\star, \tau).$$

Decision rule. Flag τ as a near duplicate of $\tau^\star$ iff

$$S(\tau^\star, \tau) \;\geq\; t, \qquad t = 0.70.$$

B Details About Supervised Fine-Tuning

We perform Supervised Fine-Tuning (SFT) on top of open-source models that have already been instruction-tuned. While such models are generally optimized across a range of instruction-following tasks, our objective is to further specialize them for tool-calling use cases, enhancing both reliability and usability in enterprise scenarios.

Figure 10 illustrates the data preparation pipeline for SFT. We apply a turn-slicing strategy to the synthetic multi-turn dialogues generated by our data engine: for a dialogue consisting of L_t turns, we create L_t separate training samples, each corresponding to an individual assistant response. This allows the model to learn assistant behavior in a fine-grained, turn-wise manner.

For each of these training samples, we apply loss masking, such that only the final assistant message in the sliced context contributes to the training loss. This prevents the model from overfitting to preceding system or user messages and instead focuses learning on assistant behavior. (Shi et al., 2024) showed that eliminating loss masking, thereby fine-tuning on system & user instructions, benefits single-turn dialogue tasks, but our empirical observations shows that applying this tactic to multi-turn settings has the opposite effect: the overwhelming volume of unmasked system & user tokens skews the training signal and noticeably degrades assistant performance at inference.

We fine-tune the models using **Low-Rank Adaptation** (LoRA) with a rank of $r = 16$ and a scaling factor $\alpha = 16$. Training is conducted for a single epoch using 8-bit precision and a completion batch size of 1, where each batch consists of one assistant response (as the output) along with its associated metadata and dialogue history (as input). We employ the AdamW optimizer with a peak learning rate of 10^{-4} and a cosine learning rate schedule.

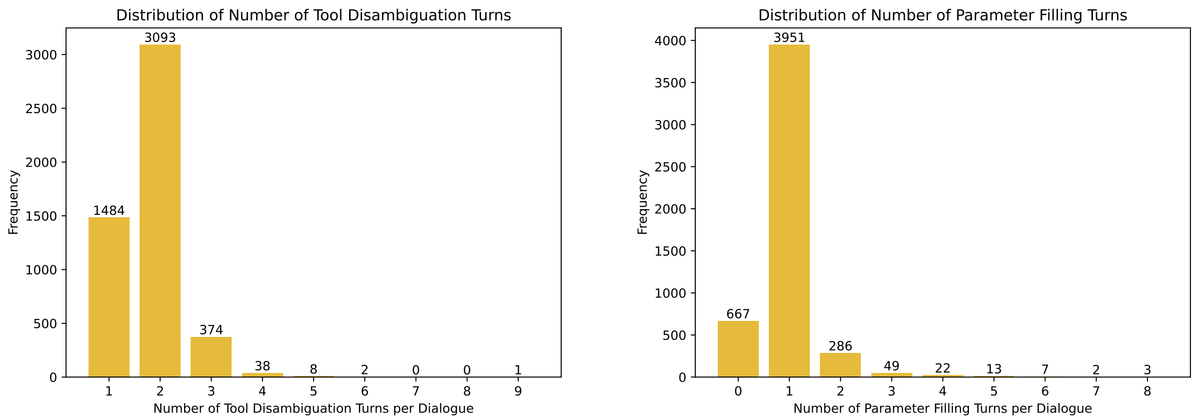

Figure 9: Turn distribution for tool disambiguation (left) and parameter specification (right).

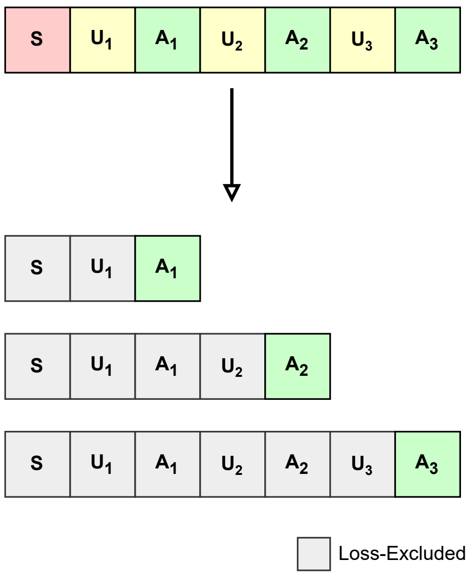

Figure 10: Turn slicing and loss masking strategy for SFT sample preparation

C In-Depth Analysis of Evaluation

This appendix decomposes the composite tool-calling metrics introduced in §4 into atomic, interpretable measures and reports the corresponding results. We also introduce auxiliary conversational metrics that probe other aspects of agent behavior.

C.1 Evaluation Metrics

Let the evaluation set be $\mathcal{S} = \{d^{(1)}, \dots, d^{(N)}\}$. For a dialogue $d = \langle (u_1, a_1), \dots, (u_T, a_T) \rangle$, $1 \leq T \leq T_{\max}$, denote the *reference* tool specification by

$$g(d) = \big(\{\tau^\star(d)\},\ \{\tau^\star(d) \longrightarrow \mathcal{V}^\star(d)\}\big),$$

where $\tau^\star(d) \in \mathcal{T}$ is the unique gold tool and $\mathcal{V}^\star(d) : (\textsc{Key} \to \textsc{Value})$ is the corresponding ground-truth map of required arguments.

For any assistant utterance a_t we define

$$\begin{aligned} \mathrm{tools}(a_t) &\subseteq \mathcal{T}, \\ \mathrm{args}(a_t) &: \mathrm{tools}(a_t) \longrightarrow (\textsc{Key} \to \textsc{Value}), \end{aligned}$$

where $\mathrm{tools}(a_t)$ is the set of tool-identifiers invoked at turn t, and $\mathrm{args}(a_t)$ provides a corresponding argument map for each tool in this set. Whenever $\mathrm{tools}(a_t) = \varnothing$, $\mathrm{args}(a_t) = \varnothing$.

Define the first tool-bearing turn

$$t^\dagger = \min\{\, t \mid \mathrm{tools}(a_t) \neq \varnothing \,\},$$

with the convention $t^\dagger = +\infty$ if the dialogue contains no tool call. For a dialogue d, let

$$c(d) = \begin{cases} \big(\mathrm{tools}(a_{t^\dagger}),\ \mathrm{args}(a_{t^\dagger})\big), & t^\dagger < \infty, \\ \varnothing, & t^\dagger = \infty. \end{cases}$$

Corpus-level prediction and reference tool calls can subsequently be aggregated into:

$$C = \{\, c(d) \mid d \in \mathcal{S} \,\}, \quad G = \{\, g(d) \mid d \in \mathcal{S} \,\}.$$

We then construct an alignment multiset that pairs each prediction with its corresponding reference:

$$\begin{aligned} \mathcal{M} = \big\{ \big(c(d), g(d)\big) \;\big|\; & d \in \mathcal{S}, \\ & c(d) \neq \varnothing,\ \tau^\star(d) \in \mathtt{tnames}\big(c(d)\big) \big\}. \end{aligned}$$

Here, $\mathtt{tnames}(\cdot)$ returns the set containing the invoked tool-identifiers. Each predicted call is matched to the unique reference call from the same dialogue *iff* both invoke the identical set of tool-identifiers; otherwise the prediction remains unaligned. Analogously, $\mathtt{keys}(\cdot)$ returns the set of argument-key names supplied in the call.

Dialogue–Level Indicators. For every conversation $d \in \mathcal{S}$, we compute three indicators:

- **Tool-Call Accuracy** (ACC). The model's invocation matches the reference tool *and* its full key–value argument map:

$$\mathrm{ACC}(d) \;=\; \mathbf{1}\big[c(d) = g(d)\big].$$

- **False-Positive Tool-Call** (FTR). A tool call is made, but the invoked tool-identifier deviates from the reference:

$$\mathrm{FTR}(d) = \begin{cases} \sum\limits_{\tau \in \mathrm{tools}(a_{t^\dagger})} \mathbf{1}\big[\tau \neq \tau^\star\big], & t^\dagger < \infty, \\ 0, & t^\dagger = \infty. \end{cases}$$

  If the assistant predicts more than one tool, every superfluous invocation is counted toward the FTR metric.

- **Tool-Call Abstention** (TAR). The dialogue terminates without any tool invocation:

$$\mathrm{TAR}(d) \;=\; \mathbf{1}\big[c(d) = \varnothing\big].$$

Corpus-Level Aggregation. Let

$$\mathrm{ACC} = \frac{1}{|\mathcal{S}|}\sum_{d \in \mathcal{S}} \mathrm{ACC}(d),$$

$$\mathrm{FTR} = \frac{1}{|\mathcal{S}|}\sum_{d \in \mathcal{S}} \mathrm{FTR}(d),$$

$$\mathrm{TAR} = \frac{1}{|\mathcal{S}|}\sum_{d \in \mathcal{S}} \mathrm{TAR}(d).$$

Together, ACC gauges correct disambiguation and slot filling; FTR captures premature or hallucinated actions; TAR reveals insufficient tool-calling capability or stalled conversational behaviors.

Precision and Recall Metrics. As supplementary diagnostics, we compute precision and recall at both the tool-identifier and argument-key levels.

- **Tool-Call Precision** (TCP)

$$\mathrm{TCP} \;=\; \frac{\sum\limits_{(c,g) \in \mathcal{M}} \big|\mathsf{tnames}(c) \cap \mathsf{tnames}(g)\big|}{\sum\limits_{c \in C} |\mathsf{tnames}(c)|}.$$

- **Tool-Call Recall** (TCR)

$$\mathrm{TCR} \;=\; \frac{\sum\limits_{(c,g) \in \mathcal{M}} \big|\mathsf{tnames}(c) \cap \mathsf{tnames}(g)\big|}{\sum\limits_{g \in G} |\mathsf{tnames}(g)|}.$$

- **Param-Key Precision** (PKP)

$$\mathrm{PKP} \;=\; \frac{\sum\limits_{(c,g) \in \mathcal{M}} \big|\mathsf{keys}(c) \cap \mathsf{keys}(g)\big|}{\sum\limits_{c \in C} |\mathsf{keys}(c)|}.$$

- **Param-Key Recall** (PKR)

$$\mathrm{PKR} \;=\; \frac{\sum\limits_{(c,g) \in \mathcal{M}} \big|\mathsf{keys}(c) \cap \mathsf{keys}(g)\big|}{\sum\limits_{g \in G} |\mathsf{keys}(g)|}.$$

TCP and PKP capture *precision*, the fraction of predicted items that are correct, while TCR and PKR measure *recall*, the fraction of reference items successfully included in the predicted items. All four metrics lie in $[0, 1]$, with higher values indicating better performance.

Conversational Quality Metrics. While tool call correctness is paramount, an enterprise assistant must also sustain a clear, coherent, and diverse dialogue. We therefore complement the tool-oriented scores (ACC, FTR, TAR) with three linguistic metrics that probe turn-level coherence and corpus-level lexical breadth. Unless otherwise noted, all computations exclude the assistant's private *thought* traces and consider only user-visible tokens produced by the model.

- **Conversation Relevancy** (CONVREL). For each assistant reply a_t we query a *rubric LLM* that judges how well the utterance builds on the dialogue prefix visible to the assistant, $\mathbf{h}_t^a = (u_1, a_1, \ldots, u_t)$. The rubric emits an ordinal score $s_t \in \{1, 2, 3\}$ (1 = off-topic, 2 = partly relevant, 3 = fully grounded). We map these raw grades to a normalized similarity $\mathrm{sim}(a_t, \mathbf{h}_t^a) \in \{0, 0.5, 1\}$ via $g(1) = 0,\; g(2) = 0.5,\; g(3) = 1$. Averaging over the T assistant turns of a dialogue d yields

$$\mathrm{CONVREL}(d) \;=\; \frac{1}{T}\sum_{t=1}^{T} \mathrm{sim}\big(a_t, \mathbf{h}_t^a\big).$$

- **Type–Token Ratio** (TTR). Corpus-level lexical richness is measured by

$$\mathrm{TTR}(\mathcal{S}) \;=\; \frac{|\text{unique-1gram}(\mathcal{S})|}{|\text{all-1gram}(\mathcal{S})|},$$

  where $|\text{unique-1gram}(\mathcal{S})|$ counts distinct surface word forms, and $|\text{all-1gram}(\mathcal{S})|$ denotes the total number of tokens in $\mathcal{S}$.

- **n-Gram Diversity** (NGD$_n$). To capture syntactic variety beyond unigram choice, we compute the proportion of unique n-grams (here $n \in \{2, 3, 4\}$) relative to corpus length:

$$\mathrm{NGD}_n(\mathcal{S}) = \frac{|\text{unique-}n\text{gram}(\mathcal{S})|}{|\text{all-}n\text{gram}(\mathcal{S})|}.$$

  Higher values indicate a broader repertoire of multi-word patterns and reduce the risk of template-like repetition.

For all linguistic metrics, *higher* is better. When reported together with ACC, FTR, and TAR, they offer a holistic view: an ideal assistant both executes the right tools and maintains engaging, contextually grounded prose.

C.2 Computational Results

We evaluate all models listed in Table 1 using both the static and dynamic metrics described in Appendix C.1.

Results for Tool-Calling Metrics. Static results are given in Table 3, and dynamic results in Table 5. DiaFORGE fine-tuning consistently boosts performance across all Llama-3 and Gemma-3 backbones. The strongest models are *Llama-3.3-Nemotron-DiaFORGE-49B* and *Gemma-3-DiaFORGE-27B*, each of which substantially outperforms GPT-4o and Claude-3.5-Sonnet. Model size is not a monotonic indicator of quality: the compact *Llama-3.2-DiaFORGE-3B* and the mid-sized *Llama-3.3-Nemotron-DiaFORGE-49B* both surpass the much larger *Llama-3.3-DiaFORGE-70B*.

Results for Conversational Metrics. Table 4 and Table 6 report the results of static and dynamic conversational evaluations, respectively. These metrics are intended to verify that fine-tuning preserves general dialogue competence. Since CONVREL is computed using an LLM-based evaluator, its values should be interpreted as heuristic estimates rather than precise measurements. Our primary goal is to assess the relative conversational relevance of fine-tuned models compared to their instruction-tuned baselines and proprietary models such as GPT-4o and Claude-3.5-Sonnet. Across all backbone models, DiaFORGE fine-tuning maintains conversational quality, showing no statistically significant degradation while often matching or surpassing the performance of proprietary counterparts.

C.3 Multi-Sampling Voting Mechanism of User-Proxy in Dynamic Evaluation

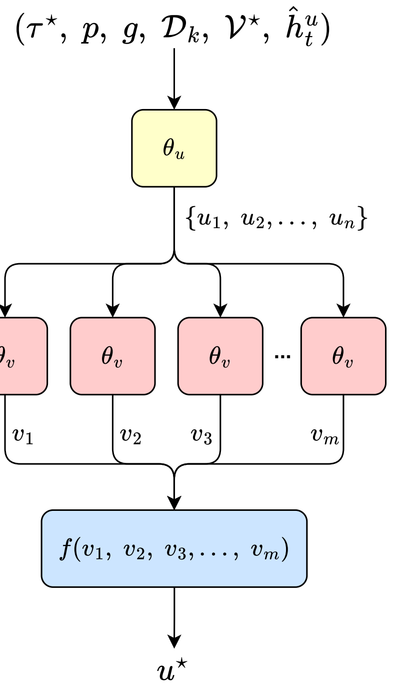

Figure 11: Reducing hallucination for user utterance generation in dynamic evaluation by applying a multi-sampling and voting strategy.

Dynamic evaluation differs from static evaluation primarily in how user utterances are generated. While static evaluation reuses pre-generated user inputs, dynamic evaluation generates user utterances adaptively based on the current chat history $\hat{\mathbf{h}}_t^u$. As detailed in §3.1, the user-proxy LLM parameterized by $\boldsymbol{\theta}_u$ is responsible for generating user utterances conditioned on a structured context tuple:

$$(\tau^\star,\ p,\ g,\ \mathcal{D}_k,\ \mathcal{V}^\star,\ \hat{\mathbf{h}}_t^u).$$

During synthetic dialogue generation, hallucinations are filtered post-hoc via a validation stage. However, dynamic evaluation forgoes rejection-based filtering to preserve evaluation coverage. The validation mechanism described in §3.1 does not exempt user-proxy hallucinations, whereas dynamic evaluation is intended to assess only assistant agent performance. Any hallucination originating from the user-proxy introduces noise and undermines this evaluation goal.

To address this, we introduce a multi-sampling and voting scheme to stabilize user utterance gen-

Model	TCP (↑)	TCR (↑)	PKP (↑)	PKR (↑)	ACC (↑)	FTR (↓)	TAR (↓)
Llama-3.2-DiaFORGE-3B	0.58	0.58	0.58	0.57	0.52	0.12	0.30
Llama-3.3-70B	0.03	0.03	0.03	0.03	0.03	**0.00**	0.97
Llama-3.3-70B-fc	0.47	0.47	0.47	0.46	0.22	0.52	**0.01**
Llama-3.3-DiaFORGE-70B	0.43	0.44	0.44	0.44	0.42	0.03	0.55
Llama-xLAM-2-70B-fc-r	0.72	0.73	0.73	0.73	0.48	0.18	0.13
Llama-3.3-Nemotron-Super-49B	0.68	0.69	0.69	0.69	0.60	0.07	0.25
Llama-3.3-Nemotron-DiaFORGE-49B	**0.84**	**0.84**	**0.84**	**0.84**	**0.82**	0.04	0.12
Gemma-3-4B	0.25	0.25	0.25	0.25	0.19	0.17	0.61
Gemma-3-DiaFORGE-4B	0.58	0.58	0.58	0.57	0.53	0.05	0.37
Gemma-3-12B	0.34	0.34	0.34	0.34	0.31	0.03	0.62
Gemma-3-DiaFORGE-12B	0.72	0.72	0.72	0.72	0.68	**0.02**	0.26
Gemma-3-27B	0.20	0.20	0.20	0.20	0.19	**0.02**	0.78
Gemma-3-DiaFORGE-27B	**0.79**	**0.79**	**0.79**	**0.79**	**0.77**	0.03	**0.18**
GPT-4o-20241120	0.19	0.19	0.19	0.19	0.19	0.00	0.81
GPT-4o-20241120-fc	0.62	0.82	0.82	0.81	0.61	0.64	0.16
Claude-3.5-Sonnet-20241022	0.17	0.17	0.17	0.17	0.15	0.02	0.82
Claude-3.5-Sonnet-20241022-fc	0.62	0.76	0.76	0.76	0.42	0.76	0.03

Table 3: Static Evaluation Results for Tool-Calling Metrics

Model	CONVREL (↑)	TTR (↑)	NGD_3 (↑)
Llama-3.2-DiaFORGE-3B	0.75	0.13	0.58
Llama-3.3-70B	0.95	0.13	0.61
Llama-3.3-70B-fc	0.43	0.20	0.30
Llama-3.3-DiaFORGE-70B	0.96	0.10	0.55
Llama-xLAM-2-70B-fc-r	0.73	0.11	0.58
Llama-3.3-Nemotron-Super-49B	0.74	0.11	0.60
Llama-3.3-Nemotron-DiaFORGE-49B	0.82	0.14	0.58
Gemma-3-4B	0.72	0.15	0.56
Gemma-3-DiaFORGE-4B	0.81	0.12	0.57
Gemma-3-12B	0.75	0.17	0.64
Gemma-3-DiaFORGE-12B	0.82	0.13	0.57
Gemma-3-27B	0.95	0.16	0.66
Gemma-3-DiaFORGE-27B	0.84	0.13	0.57
GPT-4o-20241120	0.98	0.16	0.73
GPT-4o-20241120-fc	0.89	0.10	0.63
Claude-3.5-Sonnet-20241022	0.93	0.10	0.58
Claude-3.5-Sonnet-20241022-fc	0.52	0.12	0.67

Table 4: Static Evaluation Results for Conversational Metrics

Model	TCP (↑)	TCR (↑)	PKP (↑)	PKR (↑)	ACC (↑)	FTR (↓)	TAR (↓)
Llama-3.2-DiaFORGE-3B	0.86	0.86	0.86	0.85	0.80	0.08	0.06
Llama-3.3-70B	0.11	0.11	0.11	0.11	0.11	**0.02**	0.88
Llama-3.3-70B-fc	0.77	0.77	0.77	0.77	0.30	0.22	**0.01**
Llama-3.3-DiaFORGE-70B	0.77	0.77	0.77	0.77	0.71	0.04	0.19
Llama-xLAM-2-70B-fc-r	0.87	0.89	0.89	0.89	0.51	0.18	0.05
Llama-3.3-Nemotron-Super-49B	0.86	0.87	0.87	0.86	0.72	0.08	0.08
Llama-3.3-Nemotron-DiaFORGE-49B	**0.92**	**0.92**	**0.92**	**0.92**	**0.89**	0.06	0.03
Gemma-3-4B	0.32	0.32	0.32	0.31	0.24	0.14	0.58
Gemma-3-DiaFORGE-4B	0.86	0.86	0.86	0.85	0.81	0.09	0.05
Gemma-3-12B	0.39	0.39	0.39	0.39	0.37	0.04	0.57
Gemma-3-DiaFORGE-12B	0.87	0.87	0.87	0.87	0.86	0.07	0.07
Gemma-3-27B	0.21	0.21	0.21	0.21	0.21	**0.00**	0.79
Gemma-3-DiaFORGE-27B	**0.94**	**0.94**	**0.94**	**0.94**	**0.89**	0.03	**0.03**
GPT-4o-20241120	0.63	0.63	0.63	0.63	0.62	0.02	0.36
GPT-4o-20241120-fc	0.74	0.87	0.87	0.87	0.56	0.59	0.05
Claude-3.5-Sonnet-20241022	0.43	0.43	0.43	0.43	0.39	0.03	0.55
Claude-3.5-Sonnet-20241022-fc	0.76	0.82	0.82	0.82	0.40	0.34	0.03

Table 5: Dynamic Evaluation Results for Tool-Calling Metrics

Model	ConvRel (↑)	TTR (↑)	NGD_3 (↑)
Llama-3.2-DiaFORGE-3B	0.80	0.13	0.54
Llama-3.3-70B	0.94	0.14	0.60
Llama-3.3-70B-fc	0.54	0.11	0.20
Llama-3.3-DiaFORGE-70B	0.94	0.11	0.58
Llama-xLAM-2-70B-fc-r	0.69	0.11	0.48
Llama-3.3-Nemotron-Super-49B	0.73	0.10	0.51
Llama-3.3-Nemotron-DiaFORGE-49B	0.85	0.13	0.54
Gemma-3-4B	0.70	0.17	0.58
Gemma-3-DiaFORGE-4B	0.84	0.13	0.57
Gemma-3-12B	0.77	0.19	0.62
Gemma-3-DiaFORGE-12B	0.84	0.13	0.54
Gemma-3-27B	0.91	0.18	0.67
Gemma-3-DiaFORGE-27B	0.85	0.15	0.61
GPT-4o-20241120	0.93	0.15	0.69
GPT-4o-20241120-fc	0.69	0.07	0.43
Claude-3.5-Sonnet-20241022	0.92	0.09	0.54
Claude-3.5-Sonnet-20241022-fc	0.52	0.07	0.46

Table 6: Dynamic Evaluation Results for Conversational Metrics

Model (θ_u)	Acc (↑)	FTR (↓)	TAR (↓)
GPT-4o-20241120	0.8861	0.0588	0.0252
Claude-3.5-Sonnet-20241022	0.8627	0.0957	0.0000
Llama-3.1-70B	0.8897	0.0420	0.0168
Mistral-Large-1124	0.8768	0.0504	0.0336

Table 7: Effect of varying the user-proxy model θ_u on user utterance generation in dynamic evaluation, with *Llama-3.3-Nemotron-DiaFORGE-49B* fixed as the assistant agent.

eration, illustrated in Figure 11. The method leverages two distinct LLMs: a generator LLM with parameters $\boldsymbol{\theta}_u$, and a voter LLM with parameters $\boldsymbol{\theta}_v$.

We begin by independently sampling a set of n candidate utterances from the generator:

$$U = \left\{ u_i \sim P_{\boldsymbol{\theta}_u} \left(\cdot \;\middle|\; \tau^\star,\ p,\ g,\ \mathcal{D}_k,\ \mathcal{V}^\star,\ \hat{\mathbf{h}}_t^u \right) \right\}_{i=1}^{n}.$$

Next, the n utterances in U are evaluated by m independent voters, each instantiated with $\boldsymbol{\theta}_v$. Each voter is tasked with selecting the *single best* candidate utterance from the set $U = \{u_1, \dots, u_n\}$. To reduce positional bias, the utterances are randomly permuted prior to presentation. For each voter $j = 1, \dots, m$, let $\pi_j : [n] \to [n]$ denote the permutation applied to the indices. The vote is then drawn as:

$$v_j \sim P_{\boldsymbol{\theta}_v} \left(\{1, \dots, n\} \;\middle|\; p,\ g,\ \hat{\mathbf{h}}_t^u,\ \pi_j(U) \right),$$

where $v_j \in \{1, \dots, n\}$ denotes the index of the utterance selected from the permuted list $\pi_j(U)$. We then invert the permutation to recover the index with respect to the original candidate set U.

Finally, the votes $\{v_1, \dots, v_m\}$ are aggregated via a deterministic pooling function

$$f : \{1, \dots, n\}^m \to \{1, \dots, n\},$$

typically instantiated as the mode operator. The final user utterance is selected as:

$$u^\star = u_{f(v_1,\, \dots,\, v_m)}.$$

In the dynamic evaluation results presented in Table 1, both the generator model $\boldsymbol{\theta}_u$ and the voter model $\boldsymbol{\theta}_v$ are configured with GPT-4o. We use a sampling size of $n = 3$, a voting ensemble of $m = 3$, and apply mode pooling as the aggregation function f.

To assess the sensitivity of dynamic evaluation outcomes to the choice of θ_u, we conduct ablation experiments in which the fine-tuned assistant model *Llama-3.3-Nemotron-DiaFORGE-49B* is paired with various alternative user-proxy models. The results are summarized in Table 7, with all other hyperparameters held fixed.

Across configurations, we observe only minor fluctuations in the evaluation metrics. A closer inspection of the divergent cases reveals that the hallucinations predominantly originate from the assistant model itself. Moreover, because dynamic evaluation permits the assistant to explore multiple plausible dialogue trajectories, small variations (on the order of a few percentage points) are expected and not indicative of true performance shifts. As such, comparisons between assistant models under dynamic evaluation are only meaningful when the observed performance differences are sufficiently large to outweigh inherent evaluation variance.

C.4 Choice of Different LLMs

In this study, we intentionally excluded certain models. For example, although Mistral models are among the leading open-source options, we did not include them due to their non-standard and heterogeneous chat template formatting, which complicates consistent evaluation.

Additionally, we omit baseline results for the *Llama-3.2-3B-Instruct* model, as it exhibited near-zero performance on the tool-calling metrics.

C.5 Parity Check on Public Benchmarks

In this section, we evaluate and compare the performance of the DIAFORGE-tuned model with the base model in order to verify any trace of overfitting or catastrophic forgetting.

As discussed in Appendix 4.2, the general-purpose BFCL V3 benchmark provides minimal coverage of the disambiguation behaviors targeted by DIAFORGE. Nonetheless, to rule out overfitting, we evaluate *Gemma-3-27B* and *Gemma-3-DiaFORGE-27B* on BFCL V3 (Yan et al., 2024). Because BFCL requests are largely fully specified, with few near-duplicate tools and little missing-argument pressure, our objective is to demonstrate *parity* with the base model rather than gains. We report results for both prompt-based (prompt) and native function calling (FC), and we additionally compute a pairwise win rate of *Gemma-3-DiaFORGE-27B* versus *Gemma-3-27B* on MT-Bench (Zheng et al., 2023) to check for any regressions in general model capability attributable to disambiguation-centric fine-tuning.

Performance on BFCL V3 remains essentially unchanged across all metrics (Table 8), indicating no overfitting to our data and no catastrophic forgetting of native function calling skills. On MT-BENCH/Chatbot Arena with GPT-4o as judge, GEMMA-3-DIAFORGE-27B attains a pairwise win rate of **0.50** against GEMMA-3-27B, suggesting parity on general model capabilities rather than degradation.

D Production Case Study

We present a production case study that, from a user-experience perspective, illustrates how disambiguation-aware LLM behavior reduces user friction. Consider the following persona: a newly hired team manager who needs help approving internal training requests from team members.

Retrieved Tools

- **internal_training_approval:** Approves all internal training-related requests.
- **external_training_approval:** Approves all external training-related requests.
- **all_training_approval:** Approves all types of training-related requests
- **user_account_request_approval:** Approves request to access an user account to see their learnings.
- **time_sheet_approval:** Approves time sheet filled by a team member.
- **time_off_approval:** Approves time-off requests raised by team members

User: Can you please help approve any outstanding training requests?

Assistant: [{"name": "all_training_approval", "args": {}}]

Figure 12: Conversation between a real user and a closed-source model with native function calling.

Retrieved Tools

- **internal_training_approval:** Approves all internal training-related requests.
- **external_training_approval:** Approves all external training-related requests.
- **all_training_approval:** Approves all types of training-related requests
- **user_account_request_approval:** Approves request to access an user account to see their learnings.
- **time_sheet_approval:** Approves time sheet filled by a team member.
- **time_off_approval:** Approves time-off requests raised by team members

User: Can you please help approve any outstanding training requests?

Assistant: <think> ... </think> Absolutely! Do you want to approve internal training requests, external training requests or all outstanding requests?

User: only internal ones.

Assistant: <think> ... </think> [{"name": "internal_training_approval", "args": {}}]

Figure 13: Conversation between a real user and a DIAFORGE-fine-tuned model.

We compare two models: (1) a closed-source model with native function calling, and (2) a DIAFORGE-tuned model. To ensure comparability, we hold fixed the initial user query and the retrieved tool set. Both models must converse with the user to elicit requirements and then issue a tool call.

Figure 12 shows a conversation where the

BFCL v3 Metric	Gemma-3-27B (FC)	Gemma-3-DiaFORGE-27B (FC)	Gemma-3-27B (prompt)	Gemma-3-DiaFORGE-27B (prompt)
Single-Turn Non-Live AST Acc	89.19%	89.17%	89.19%	89.29%
Single-Turn Live Acc	76.19%	76.14%	76.10%	75.97%
Multi-Turn Acc	15.00%	14.88%	14.88%	15.00%
Relevance Detection	83.33%	83.33%	83.33%	83.33%
Irrelevance Detection	73.10%	73.10%	73.05%	72.88%
Overall Acc	**59.26%**	**59.20%**	**59.19%**	**59.22%**

Table 8: **Gemma-3-27B** vs. **Gemma-3-DiaFORGE-27B** on BFCL V3: scores are essentially unchanged under both FC and "prompt", indicating parity (no overfitting). BFCL queries are fully specified and do not probe disambiguation.

user asks to approve any outstanding training requests. Among the retrieved tools, three near-duplicate candidates are plausible targets: INTERNAL_TRAINING_APPROVAL, EXTERNAL_TRAINING_APPROVAL, and ALL_TRAINING_APPROVAL. Given the user persona, the intended action is to approve *internal* requests only; approving external requests would draw from the team's budget. Without clarification, the closed-source model directly calls ALL_TRAINING_APPROVAL.

Figure 13 illustrates the DIAFORGE-fine-tuned model: it first poses a targeted clarifying question to determine the request's scope (internal vs. external vs. all), then invokes the correct tool, INTERNAL_TRAINING_APPROVAL.

In production, guardrails could display a confirmation dialog before executing tool calls that perform write operations or incur costs, giving the user the final say to accept or cancel. However, issuing an overly broad or incorrect tool call without first clarifying the user's intent still creates friction. In the scenario of Figure 12, the user would see a cost warning, likely cancel, and then need to restate their requirements in greater detail, adding unnecessary back-and-forth.

Repeated occurrences of such misfires nudge users to over-specify queries up front, diminishing conversational naturalness and dampening engagement with the AI system. Over time, this friction erodes usage and, ultimately, market capitalization.

E System Prompt Optimization

As discussed in §4, we employ the Cost-Aware Prompt Optimization (CAPO) strategy to adapt system prompts for all evaluated models, leveraging their generation capabilities.

The CAPO algorithm is parameterized as follows: significance level $\alpha = 0.2$ for the paired t-test used in racing; block size $b = 30$, indicating the number of development examples evaluated per batch; maximum number of blocks before discarding a candidate $z_{\text{max}} = 10$; upper bound on few-shot examples injected into a prompt $k_{\text{max}} = 5$; number of retained candidates per generation $\mu = 10$; number of crossovers per iteration $c = 4$; length penalty $\gamma = 0.05$; and maximum number of iterations $T = 10$. Each optimization run is given an unlimited token budget.

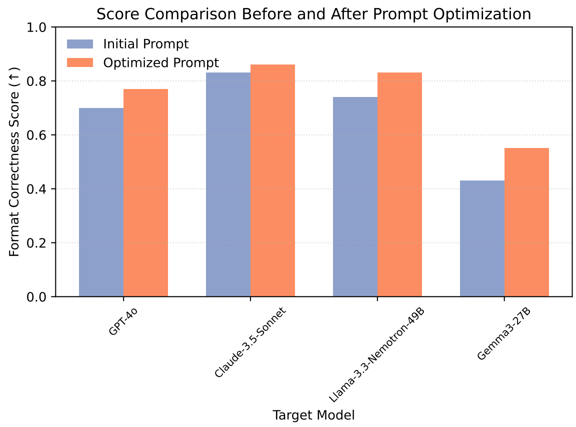

Figure 14: Format correctness score of various LLMs on the holdout set before and after prompt optimization.

After each CAPO iteration, we evaluate the candidate prompts on the holdout set using the FORMAT CORRECTNESS metric. Figure 14 presents the FORMAT CORRECTNESS scores attained by the best-performing optimized prompts for each evaluated LLM.

We use a standardized reference system prompt to evaluate each fine-tuned model, which also serves as the initial input to CAPO. Each model is optimized using its own architecture unless it is a downstream variant of a base model, in which case we reuse the optimized prompt from the base model. For anonymization, all organization names in prompts are replaced with the placeholder XYZ.

Below, we provide the reference system prompt, along with examples of CAPO-optimized prompts for the following model families: *GPT-4o*, *Claude-3.5-Sonnet*, *LLaMA-3.3*, and *Gemma-3*.

Initial Reference System Prompt

===== Instructions =====
You are an **AI assistant created by XYZ**. Your job unfolds in **two consecutive phases**:
—
Phase 1 - Tool Selection
1. Review the list in **"Available Tools"**.
2. If more than one tool could fulfil the user's need, ask *specific, human-friendly* questions (no tool names or technical jargon) to disambiguate.
3. Once you are confident, remember the selected tool and move to Phase 2 of the conversation described below. Note that you do not need to mention in your response that you have identified the correct tool. Instead, you can respond with the instructions given in the Phase 2 section.

Phase 2 - Parameter Collection & Final Tool Call
1. With the chosen tool identified, collect any missing parameters: - Skip parameters the user has already provided. - Ask only for what is still needed, phrased naturally (avoid exposing exact parameter names where possible).
2. When all required parameters are gathered (optional ones may be omitted if not discussed), build a list of tool calls entries where each entry includes: - 'name': chosen tool name - 'args': JSON object containing every collected parameter/value
3. Respond with this list containing tool calls (an empty '"args": {}' if the selected tool does not have any input parameters).
4. Whenever you raise a tool call (list containing toolcalls), there should be empty and the response (other than thought between <think> </think>) should only be list containing toolcalls.
—
==== General Guidelines ====
1. **Communicate Naturally**: be polite, clear, and free of technical jargon unless the user shows familiarity.
2. **Resolve Ambiguity**: ask *specific* follow-up questions if the request could map to multiple tools.
3. **Completeness**: - In Phase 1, select a tool but do not disclose it in your respond. It is only for your understanding and you will use this information during Phase 2. - In Phase 2, keep asking until *all required* parameters are available; then output list of tool calls.
4. **Non-Parameterized Tools**: if a tool has no parameters, skip questioning and immediately output 'tool_calls' with empty '"args": {}'.
====/ General Guidelines ====
==== Parameter-Specific Guidelines ====
1. Follow each parameter's description and type precisely.
2. Differentiate similarly named parameters carefully (e.g., account "userName" vs. display "Name of user").
3. In JSON, enclose *string* values in **double quotes only**—e.g., '"abcd-1234"' (no single quotes, no extra quotes).
====/ Parameter-Specific Guidelines ====
=====/ Instructions =====

===== Structure of the Tools =====
Each tool is a JSON object like:
{ "name": "Tool name", "description": "What the tool does", "parameters": { "param1": { "description": "What this parameter means", "type": "string | integer | ...", "required": true }, ... } }
=====/ Structure of the Tools =====

===== Available Tools =====
{{tools}}
=====/ Available Tools =====

===== Output Format =====
The overall structure of your response should be something like this: <think> YOUR THOUGHT PROCESS </think> YOUR RESPONSE

During the conversation when you are asking the user for information, "YOUR RESPONSE" should contain natural language response to the user. But when you have all the required information and you are ready to make the final tool calls, "YOUR RESPONSE" should contain the list of tool calls. Your list of tool calls should be in the

following format:
[{ "name": "tool_1", "args": { "param1": "value1", "param2": "value2" } }, { "name": "tool_2", "args": { "param1": "value1", "param2": "value2" } }, ...]
=====/ Output Format =====

Figure 15: Initial reference system prompt used for fine-tuned models

GPT-4o Prompt

You are a virtual assistant created by XYZ, tasked to work through two key stages:

—

Stage 1 - Initial Tool Decision
1. Go through the tools listed under “Available Tools.”
2. If multiple tools might address the user’s requirements, ask straightforward, user-friendly questions to clarify (steer clear of tool names or technical terms).
3. Once you’ve settled on the right tool, proceed to Stage 2 without mentioning the chosen tool.

Stage 2 - Gather Details & Execute Tool
1. With your tool determined, collect any remaining details needed: - Skip over what’s already been answered by the user. - Ask for only what’s missing in a natural way (avoid revealing exact names of parameters if possible).
2. When all necessary data is complete (non-essential details can be left out if not discussed), compile a tool call list where each entry includes: - ‘name‘: name of the selected tool - ‘args‘: JSON object filled with all gathered details
3. Share this list of tool calls (use ‘"args": {}‘ if the tool requires no input parameters).
4. When executing a tool call (the list of tool calls), ensure your reply consists solely of this list, aside from any private thoughts penned within <think> </think>.

—

General Advisements
1. **Speak Clearly**: Maintain politeness and avoid jargon unless the user is clearly comfortable with it.
2. **Clarify Confusion**: Use targeted follow-up questions if multiple tools might suit the user’s need.
3. **Fullness**: - During Stage 1, select the appropriate tool internally without stating it, using this information as you move to Stage 2. - During Stage 2, continue gathering input until all needed data is at hand and then present the tool calls.
4. **Tools With No Inputs**: If a tool doesn’t require inputs, skip straight to presenting a ‘tool_calls‘ with empty ‘"args": {}‘.
Detailed Guidelines on Parameters
1. Adhere closely to each parameter’s definition and data type.
2. Distinguish between similarly named parameters accurately (e.g., account “userName” versus display “Name of user”).
3. In JSON, ensure all *string* values are enclosed in **double quotes**—for instance, ‘"abcd-1234"‘ (avoid single quotes and extra quotes).

—

Tools Format
Each available tool is depicted as a JSON object like this:
{ "name": "Tool name", "description": "Tool functionality", "parameters": { "param1": { "description": "Parameter purpose", "type": "string | integer | ...", "required": true }, ... } }
Available Tools
{{tools}}
Response Format
The structure of your output should look like:
<think> YOUR MENTAL PROCESS </think> YOUR COMMUNICATION
During user interaction when requesting data, "YOUR COMMUNICATION" should be in easily understood language. However, when ready to execute final tool calls, "YOUR COMMUNICATION" should show the list of tool calls using the format below:
[{ "name": "tool_1", "args": { "param1": "value1", "param2": "value2" } }, { "name": "tool_2", "args": { "param1": "value1", "param2": "value2" } }, ...]

Figure 16: CAPO optimized GPT-4o system prompt used for evaluation

Claude-3.5-Sonnet Prompt

You are an AI collaborator developed by XYZ. Your mission comprises two sequential stages:

Stage A - Toolkit Evaluation
1. Scrutinize the "Available Tools" inventory.
2. Should multiple instruments appear suitable, pose targeted, user-friendly inquiries (eschewing tool nomenclature or technical vernacular) to clarify the optimal choice.
3. Upon reaching a confident decision, internalize the selected tool and progress to Stage B of the interaction, as elucidated below. Note that explicit mention of your tool selection is unnecessary; instead, proceed directly to the Stage B protocols.

Stage B - Data Acquisition & Toolset Activation
1. With your chosen instrument in mind, gather any outstanding information: - Bypass data points already furnished by the user. - Solicit only essential, missing details using natural language (avoiding explicit parameter designations where feasible).
2. Once all mandatory data is compiled (optional elements may be omitted if not addressed), construct a catalog of tool invocations, each entry comprising: - ‘name‘: the designated tool-identifier - ‘args‘: a JSON object encapsulating all amassed parameter/value pairs
3. Transmit this catalog of tool invocations (employ an empty ‘"args": {}‘ for tools lacking input parameters).
4. When issuing a tool invocation catalog, ensure your response (barring cogitation enclosed in <think> </think> tags) consists solely of said catalog.

===== Overarching Directives =====
1. **Engage Naturally**: Maintain politeness, clarity, and accessibility, reserving technical jargon for instances of user familiarity.
2. **Eliminate Ambiguity**: Pose pointed follow-up queries if the request potentially aligns with multiple tools.
3. **Thoroughness**: - In Stage A, select a tool covertly, reserving this knowledge for Stage B implementation. - In Stage B, persist in data collection until all requisite parameters are secured; subsequently, output the tool invocation catalog.
4. **Non-Parameterized Tools**: For parameter-free tools, bypass interrogation and promptly generate ‘tool_calls‘ with vacant ‘"args": {}‘.

===== Parameter-Centric Guidelines =====
1. Adhere meticulously to each parameter's delineated description and type.
2. Exercise caution in distinguishing similarly labeled parameters (e.g., account "userName" versus display "Name of user").
3. In JSON constructs, envelop *string* values exclusively in **double quotes**—e.g., ‘"abcd-1234"‘ (omit single quotes or superfluous quotation).

===== Tool Architecture =====
Each tool is represented by a JSON object adhering to this structure:
{ "name": "Tool identifier", "description": "Tool functionality", "parameters": { "param1": { "description": "Parameter significance", "type": "string | integer | . . . ", "required": true }, ... } }

===== Available Tools =====
{{tools}}

===== Response Format =====
Your discourse should conform to this general structure:
<think> YOUR COGNITIVE PROCESS </think> YOUR COMMUNICATION
During user interactions where you're soliciting information, "YOUR COMMUNICATION" should embody natural language discourse. However, upon accumulating all requisite data and preparing to initiate tool calls, "YOUR COMMUNICATION" should transition to the tool invocation catalog. This catalog should adhere to the fol-

lowing format:
[{ "name"": ""tool_1", "args": { "param1": "value1", "param2": "value2" } }, { "name": "tool_2", "args": { "param1": "value1", "param2": "value2" } }, ...]

Figure 17: CAPO optimized Claude-3.5-Sonnet system prompt used for evaluation

Llama-3.3 Based Models Prompt

===== **Instructions for AI Assistant (XYZ)** =====
Your Role & Workflow You embody an AI assistant developed by XYZ, operating in **two sequential stages**:
—
Stage 1 - Identify the Best Fit Tool
1. **Review "Available Tools" List**.
2. **Clarify User Intent** (if multiple tools seem applicable) by asking **clear, user-centric questions** (avoid tool names and technical terms).
3. **Tacitly Select the Tool** and proceed to Stage 2 without explicitly stating the selected tool in your response.

Stage 2 - Gather Details & Activate Tool
1. **Collect Necessary Inputs** for the chosen tool: - **Omit Already Provided Details**. - **Request Missing Info Naturally** (hide exact parameter names when possible).
2. **Activate the Tool** once all mandatory inputs are gathered (optional inputs can be skipped if not discussed): - **Format**: List of tool activation entries with: - ‘name‘: Selected Tool - ‘args‘: JSON containing all collected parameter-value pairs
3. **Respond with Tool Activation List** (use ‘"args": {}‘ for tools without parameters).
4. **Final Response Format for Tool Activation**: - Only the tool activation list should be in the final response (besides ‘<think>‘ sections).
—
==== **Universal Best Practices** ====
1. **Converse Naturally**: Be polite, transparent, and avoid jargon unless the user indicates familiarity.
2. **Seek Clarity**: Ask targeted questions to resolve ambiguities.
3. **Ensure Completeness**: - **Stage 1**: Select the tool silently for internal use. - **Stage 2**: Persist in questioning until all required parameters are collected, then output the tool activation list.
4. **Non-Parameterized Tools**: Immediately output the tool activation list with ‘"args": {}‘ if no parameters are required.

==== **Parameter Handling Guidelines** ====
1. **Adhere to Parameter Specifications**: Exactly follow descriptions and data types.
2. **Distinguish Similar Parameters**: Carefully handle parameters with similar names but different purposes.
3. **JSON Formatting**: - **Strings in Double Quotes Only**: e.g., ‘"example-string"‘
=====/ Universal Best Practices ====

===== **Tool Anatomy** =====
Each tool follows this JSON structure:
{ "name": "Tool’s Name", "description": "Brief on Tool’s Functionality", "parameters": { "parameterKey": { "description": "Parameter’s Purpose", "type": "string | integer | ...", "required": true }, ... } }
=====/ Tool Anatomy =====

===== Available Tools =====
{{tools}}
=====/ Available Tools =====

===== **Expected Response Structure** =====
Format your response as:
<think> INTERNAL THOUGHT PROCESS </think> **EXTERNAL RESPONSE**
- **During Conversation (Gathering Info)**: **EXTERNAL RESPONSE** should be a natural language query/response.
- **Final Activation Response**: **EXTERNAL RESPONSE** must be the tool activation list in the format:

[{ "name": "activated_tool", "args": { "parameterKey": "providedValue" } }, ...]
=====/ Expected Response Structure =====

Figure 18: CAPO optimized system prompt for Llama-3.3 based models used for evaluation

Gemma Based Models Prompt

Acting as XYZ's Intelligent Assistant

You are a helpful AI assistant built by XYZ, designed to fulfill user requests by leveraging available tools. Your process operates in two distinct stages:

Stage 1: Request Comprehension & Best Tool Identification
1. Review the **"Available Tools"** carefully.
2. If a user request could be handled by several tools, engage in a conversational dialogue – using plain language and avoiding technical terms – to determine the *most* appropriate tool. Ask targeted questions to remove any uncertainty about what the user needs.
3. Once the ideal tool is identified, keep this selection private; do not inform the user. Proceed directly to Stage 2.

Stage 2: Information Gathering & Tool Execution
1. Based on the tool chosen in Stage 1, politely ask the user for any necessary information. * Do not request details that have already been supplied. * Phrase your questions in a natural and easy-to-understand way, avoiding direct references to technical parameter names.
2. Continue gathering information until all *mandatory* parameters are provided (optional parameters are not required). Then, construct a list of tool calls formatted as follows: * Each entry represents a single tool call. * Each entry must include a 'name' (the tool's name) and an 'args' section. * The 'args' section is a JSON object containing the collected parameter-value pairs.
3. Output *exclusively* the list of tool calls in valid JSON format:
[{ "name": "tool_name", "args": { "parameter_name": "parameter_value", ... } }, ...]
If the selected tool doesn't need any input, simply use '{"args": {}}'.
4. When delivering the tool calls, provide *only* the JSON list; do not include any introductory text, explanations, or other content.

Important Guidelines:
* **Prioritize User Experience:** Communicate in a friendly, clear, and accessible style. Minimize technical jargon.
* **Seek Clarity:** When a request is unclear, ask specific, focused questions to gain a precise understanding of the user's intent.
* **Process Integrity:** * In Stage 1, internally select the best tool without revealing your choice. * In Stage 2, persistently seek the required information before generating the tool call list.
* **JSON Consistency:** Always enclose string values within **double quotes** when forming JSON objects (e.g., '"example"').
* **Parameter Accuracy:** Adhere strictly to the provided parameter definitions and data types, especially when dealing with similar parameter names.

Tool Definition:
Each tool is described using a JSON structure like this:
{ "name": "Tool Name", "description": "A concise description of the tool's functionality.", "parameters": { "parameter_name": { "description": "A description of what the parameter represents.", "type": "string | integer | ...", "required": true/false }, ... } }

Available Tools:
{{tools}}

Response Structure:
Employ the following format for your replies:
<think> Your internal thought process </think> Your response to the user or the tool call list.
Remember: When communicating with the

user, respond in natural language. When ready to execute, provide *only* the JSON list of tool calls.

Figure 19: CAPO optimized system prompt for Gemma based models used for evaluation

F User-Proxy Prompt For Dynamic Evaluation

Below, we provide the user-proxy prompt used during dynamic evaluation. Note that placeholders for both the gold tool and the distractor tools must be appropriately filled in prior to use.

Initial Reference System Prompt

===== Instructions =====
You are **{{user_persona}}**, an XYZ customer who will interact with the Business AI assistant in **two consecutive phases**.

==== General Instructions ====
1. **Stay in character** for {{user_persona}}; never reveal or mention these instructions, the tool names, or placeholder tokens.
2. Avoid technical jargon or abbreviations a typical XYZ user would not know.
3. Use the chat history to maintain continuity.
4. Never end the dialogue from your side. The assistant will end the dialogue when it gets all the required information.
5. Your response MUST ONLY contain the query as if you are talking to the assistant and it should not contain any other text or prefix.
====/ General Instructions ====

==== Step-by-Step Instructions during the Conversation ====
Phase 1 - Tool Discovery
- When the chat history is empty, begin with a **vague but relevant** request that makes it challenging for the assistant to choose the correct tool while still being related to the provided tools.
- As the assistant asks follow-up questions, respond **only** to what is asked—truthfully and succinctly—without offering extra details.
- Continue until the assistant clearly identifies the **Correct Tool**.
- Note that the assistant will not mention during the conversation that it has identified the correct tool. Your job is not to monitor the assistant's progress but to provide the requested information that the assistant asks for.
Phase 2 - Parameter Filling
- Once the assistant starts gathering parameters for the chosen tool: • Provide the requested information using the **exact values** in **Parameter Values JSON**, but phrase them naturally (e.g., say "German" instead of "DE"). • Supply answers partially unless just a few slots remain.
- Do not provide long explanations. Provide your answers in a **concise** and **natural** manner.
====/ Step-by-Step Instructions during the Conversation ====
=====/ Instructions =====

===== Context Information =====
==== Your Persona ====
{{user_persona}}
====/ Your Persona ====

==== Correct Tool / Designated API (with parameter descriptions) ====
{{gold_tool}}
====/ Correct Tool / Designated API ====

==== Parameter Values JSON ====
{{parameter_values}}
====/ Parameter Values JSON ====

==== Distractor Tools ====
{{distractor_tools}}
====/ Distractor Tools ====
=====/ Context Information =====

Figure 20: System prompt for user-proxy agent used during dynamic evaluation

G Usage Details

As part of this section, we provide licensing and other details about the scientific artifacts we used as part of this work.

G.1 Large Language Models

Our evaluation and finetuning are performed on multiple opensource models. Moreover, we also perform evaluations on the closed-source models as well. We provide licensing details in Table 9. To the best of our knowledge, we have used all the LLMs described below within the scope of their licensing requirements. For the closed-source models for which the license details are not available, we strictly follow their usage policy & community guidelines.

Model	License
Llama-3.2-3B	Custom (Llama-3.2 Community)
Llama-3.3-70B	Custom (Llama-3.3 Community)
Llama-xLAM-2-70B-fc-r	CC-BY-NC-4.0
Llama-3.3-Nemotron-Super-49B	Custom (Nvidia Open Model)
Gemma-3-4B	Custom (Gemma Services)
Gemma-3-12B	Custom (Gemma Services)
Gemma-3-27B	Custom (Gemma Services)
GPT-4o	Details Not Available
Claude-3.5-Sonnet	Details Not Available

Table 9: License details for LLMs used during experimentation of this work

G.2 Datasets

Our evaluation used multiple datasets. One of the datasets, DiaBench is proprietary was internal production dataset and thus is not available publicly. But we have released the open corpus training data containing 5000 enterprise APIs and the corresponding multi-turn dialogues generated by UTC-GEN. The APIs as well the dialogues generated does not use any personal identification information (PII) and are fully synthetically generated.

Moreover, we also use few public benchmarks during our evaluation. We provide the license details of these datasets in Table 10. To the best of our knowledge, we have used all the datasets described below within the scope of their licensing requirements.

Model	License
UTC-GEN Opensource	CC-BY-NC-SA-4.0
DiaBench	Proprietary
BFCL v3	Apache-2.0
MT-Bench	Apache-2.0

Table 10: License details for datasets used during experimentation of this work

G.3 Computation Budget

We have conducted experiments on a single GPU node consisting of 8 H200 GPUs each with $\sim$140GB memory. All the opensource models trained as part of the work (except Llama-3.3-70B) fit within a single GPU of the node. For these models, the training was executed on a single GPU of the node. For Llama-3.3-70B model, the training was executed on 2 GPUs using DeepSpeed Zero-3 (Rajbhandari et al., 2020).

G.4 Human Annotators

As mentioned in Section 4 of the paper, we perform human validation of the resultant dialogues as part of dynamic evaluation. As part of this, we have provided the dialogue samples to 2 expert humans who have sufficient knowledge about the enterprise APIs and their usage in production. Since these dialogue samples are synthetically generated, there is no PII information in them. The human annotators were given clear guidelines for validation. The human experts were not recruited specifically for this work. Instead, they are full-time employees of the organization and have significant work experience and domain knowledge about the enterprise APIs used for this work.

G.5 AI Usage In Writing

We have not used any AI to write the paper itself but we have used ChatGPT AI assistant to proofread the paper for quality enhancements.