ToolDNA: Autonomous Evolution of Tool Metadata for Robust Dialogue Agents

Qiuyuan Ai[1], Cong Wang[1], Jiaqi Zhang[2], Zengxin Han[2], Jie Song[1]*
[1]Peking University
[2]Dongchedi (Dcar)
jie.song@pku.edu.cn

Abstract

Task-oriented dialogue (TOD) systems are vital for facilitating complex, goal-directed interactions across sectors like customer support and online retail. However, they face persistent limitations: labor-intensive manual metadata tuning and sparse reinforcement learning (RL) rewards that fail to diagnose invocation errors. To address this, we propose ToolDNA, a dynamic adaptation framework enabling autonomous co-evolution of policy networks and tool metadata via RL, anchored by two synergistic loops. An RL loop optimizes policies by generating rollout trajectories (reasoning, actions, descriptive updates) from user inputs, with multi-dimensional rewards refining invocations. A tool metadata loop—coordinated by a dedicated Tool Manager—evolves metadata through policy-generated candidates during rollouts and Feedback LLM-derived refinements from historical data. These mutually reinforcing loops close traditional reward gaps, forming a closed-loop trial-error-reflection cycle for self-improvement. Extensive experiments on a real-world dataset of 3,100 customer service dialogues confirm ToolDNA's superiority, with notable gains over baselines: it achieves +11% problem resolution and +54% accuracy over commercial LLMs with prompt engineering; +25%/+35% over supervised fine-tuning; and +15%/+15% over traditional RL baseline. Linguistic analysis corroborates evolved metadata retain semantic intent while enhancing parseability. Case studies in two typical contexts, i.e., car inventory search and loan calculation, further validates its ability to resolve critical ambiguities. ToolDNA pioneers scalable self-improvement for robust, deployable tool-augmented agents with minimal human oversight. We release our code to facilitate future research.

*Corresponding author.

1 Introduction

Task-oriented dialogue (TOD) systems have become critical infrastructure for enabling complex, goal-driven interactions in domains such as customer services and e-commerce. While early pipeline-based architecture suffered from error propagation and limited flexibility, the advent of large language models (LLMs) has enabled end-to-end paradigms that generate responses directly. Despite these advances, LLM-driven TOD systems remain hampered by persistent hallucinations—factually incorrect outputs that undermine reliability (Huang et al., 2025)—and inflexible workflow designs that restrict adaptability to diverse user needs (Elizabeth et al., 2025; Mo et al., 2024). Tool-augmented LLMs (Schick et al., 2023) have emerged as a predominant strategy to mitigate these limitations. By grounding LLM responses in external tools, systems can execute precise operations. An illustrative example (Fig. 1) depicts a customer requesting a used BMW X3 with a 1 to 4 year age, where the LLM-driven agent invokes the "search_used_cars" tool with tailored parameters (e.g., filters: "car_age": [1, 4]) and returns response with structured links.

However, existing tool-augmented approaches face two critical barriers: (1) *Labor-intensive metadata tuning:* Tool metadata—structured descriptive information defining tool capabilities, parameter specifications, and invocation rules (e.g., the yellow-highlighted boxes in Fig. 1)— is typically crafted manually and remains static during training. Ambiguous or incomplete metadata lead to invocation errors (e.g., mismatched parameters), necessitating costly human debugging. (2) *Inadequate error diagnosis in RL:* RL optimizes tool invocation through scalar rewards but fails to provide granular feedback on why failures occur (e.g., invalid parameters vs. tool selection errors). This sparsity impedes autonomous refinement and

Findings of the Association for Computational Linguistics: ACL 2026, pages 18660–18678

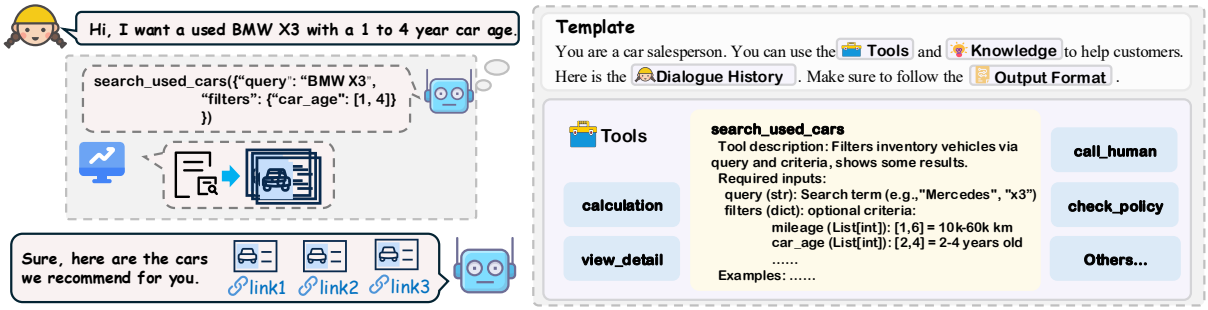

Figure 1: An illustrative example of a Tool-Augmented Task-Oriented Dialogue (TOD) system. The left panel displays a user query for a specific used car ("BMW X3"), where the agent invokes the search_used_cars tool. The right panel highlights the Tool Metadata (yellow box), which defines parameters like query and filters. In traditional systems, this metadata is static and manually curated. ToolDNA addresses the challenge where ambiguous metadata (e.g., imprecise parameter definitions) leads to invocation errors, by enabling the autonomous evolution of these definitions.

forces reliance on trial-and-error without systematic learning. Recent efforts to enhance tool learning—through prompt engineering (Schick et al., 2023; Liu et al., 2024b; Wang et al., 2024), supervised fine-tuning (Zhang et al., 2025a), or enriched reward designs (Zheng et al., 2025; Song et al., 2025b; Mahan et al., 2024) —focus narrowly on optimizing invocation policies or expanding tool libraries. Crucially, they overlook dynamic adaptation of tool metadata, leaving systems fragile to evolving user queries and incapable of translating failures into actionable improvements. Take customer's question in Fig. 1 as an example, if the agent defines "query" with "BMW" instead of "BMW X3", the result will return all BMW models (e.g., X1, 5 Series) instead of the X3, resulting in response failure. Current RL penalizes this outcome with a low reward but cannot: (1) diagnose *why* the invocation failed (e.g., the imprecise `query` parameter); (2) refine the tool metadata to prevent recurrence (e.g., enforcing model-specific queries).

To address this, we propose ToolDNA, a **D**y**N**amic **A**daptation framework that enables autonomous co-evolution of policy networks and tool metadata via reinforcement learning. The proposed framework closes the loop between tool invocation and metadata refinement via a Tool Manager—a dynamic memory module ensuring consistent tool metadata management across environments. First, an evolution mechanism enables dual outputs from the policy network: (1) tool invocation, e.g., "search_used_cars ("query": "BMW X3", "filters": "car_age": [1, 4])" and (2) updated tool metadata, e.g., "Parameters: query(str): Search term (e.g., "BMW X3"). When users specify a model, the query must exactly match". Second, a feedback mechanism leverages another LLM to distill actionable insights from historical failures (e.g., parameter ambiguities) into refined metadata. Then, these metadata are validated and integrated into the Tool Manager. This framework closes the loop between tool invocation and metadata refinement, enabling autonomous, human-like "trial-error-reflection" cycles. Empirically, we achieve significant gains on a dataset of real-world customer service dialogues from automotive and e-commerce domains, with +11% problem resolution and +54% accuracy over commercial LLMs with prompt engineering; +25%/+35% over supervised fine-tuning; and +15%/+15% over traditional RL baselines. Linguistic analysis confirms refined metadata maintain semantic consistency (cosine similarity = 0.94) while reducing lexical diversity (Type-Token Ratio (TTR) from 0.2288 to 0.1953) and syntactic complexity (tree depth from 8 to 7), making them more parseable for LLMs and human operators.

Our principal contributions are:

- **Framework & Paradigm:** We propose **ToolDNA**, a framework that establishes a **self-evolving paradigm** for tool learning. By jointly optimizing policies and metadata via a dynamic memory module, it shifts the field from static human curation to automated, trial-and-error co-evolution, significantly reducing reliance on manual oversight.
- **Validation:** We validate our approach on a real-world dialogue dataset using novel business-centric metrics. Experiments demonstrate that ToolDNA achieves substantial im-

provements over state-of-the-art baselines across multiple tasks.

- **Mechanism Insight:** We provide interpretability analysis revealing that co-evolution enhances tool parseability by reducing linguistic complexity while preserving semantic integrity, validating the effectiveness of the policy-metadata synergy.

2 Related Work

Task-Oriented Dialogue Systems. The evolution of TOD systems has progressed from early pipeline architectures—prone to error propagation and high annotation costs—to end-to-end paradigms enabled by pretrained large language models. While simplifying development (Feng et al., 2023), LLM-based TOD systems still face persistent hallucinations, producing factually incorrect outputs (Huang et al., 2025). Mitigation strategies show limitations: supervised fine-tuning (SFT) fails to fully eliminate knowledge errors (Sekulic et al., 2024), while workflow-guided approaches sacrifice interaction flexibility (Mo et al., 2024). Tool-augmented dialogue agents address these issues by integrating LLMs' semantic understanding with external tool invocation to mitigate hallucinations. This paradigm enhances flexibility through dynamic tool orchestration and reduces cross-domain adaptation costs via tool abstraction layers. Beyond static optimization, recent shifts toward self-evolving agents, such as Agent0 (Xia et al., 2025), demonstrate that models can transcend fixed datasets through tool-integrated reasoning. Our work advances this paradigm by enabling autonomous, context-driven tool invocation and adaptative abstractions throughout conversations without predefined workflows, maintaining hallucination control via structured tool outputs.

Tool-augmented LLMs. Existing tool training paradigms primarily include prompt engineering (PE) (Schick et al., 2023; Liu et al., 2024b; Wang et al., 2024), SFT, and RL (Yu et al., 2025; Li et al., 2025c; Qian et al., 2025; Wang et al., 2025; Zhang et al., 2025b). PE guides tool invocation via few-shot prompts, leveraging in-context learning without model retraining (Schick et al., 2023). Advanced structural designs introduce trial-and-error (Wang et al., 2024) and summary-to-action (Liu et al., 2024b) mechanisms, or multi-agent architectures for task decomposition (Roth et al., 2025). SFT improves basic tool usage but often suffers from pseudo-reasoning and limited generalization. With the proposal of Group Relative Policy Optimization (GRPO) (Shao et al., 2024), RL gains traction for enhancing reasoning robustness.

Recent advancements in RL for tool use focus on two core topics, i.e., reward design and training efficiency. To address sparse and misaligned rewards, methods like ToolRL (Qian et al., 2025) decomposes rewards into structural correctness (format) and semantic alignment (parameter matching). Steptool (Yu et al., 2025) leverages generalized advantage estimation (GAE) to address sparse rewards in complex, multi-step tool-using tasks. For efficiency, OTC-PO (Wang et al., 2025) introduces the "tool productivity" metric to minimize unnecessary tool invocations. For training efficiency, Nemotron-Research-Tool-N1 (Zhang et al., 2025b) simplifies training via binary rewards (reasoning format + tool invocation correctness).

However, existing research primarily targets constrained domains like programming (Li et al., 2025c) and well-defined search tasks (Chen et al., 2025; Song et al., 2025a; Li et al., 2025b; Jin et al., 2025), using datasets like xLAM (Zhang et al., 2025a) and ToolACE (Liu et al., 2024a) with limited (usually ≤ 5), predefined tools. While these studies advance tool-use protocols, they overlook critical challenges in real-world environments: scaling to diverse, ill-defined tool sets and adapting to ambiguous user instructions. In real-world business contexts, enterprises often incur substantial costs by manually refining tool metadata—a process vulnerable to misinterpretation. While recent efforts like VGCO (Li et al., 2025a) attempt to automate this via external LLM-editors, they lack the intrinsic coupling between policy learning and schema evolution. Our work bridges this gap by enabling LLMs to autonomously and dynamically evolve tool metadata within the RL loop during multi-turn TODs.

3 Methodology

A key challenge in applying RL to tool-augmented TOD systems is that traditional reward signals lack the granularity to diagnose specific invocation errors, such as ambiguous parameter metadata. To address this, we propose ToolDNA (**Fig. 2**), a framework centered on the co-evolution of tool invocation and metadata strategies driven by two synergistic loops. The **RL loop** (blue in Fig. 2) optimizes the policy by generating rollout trajectories and

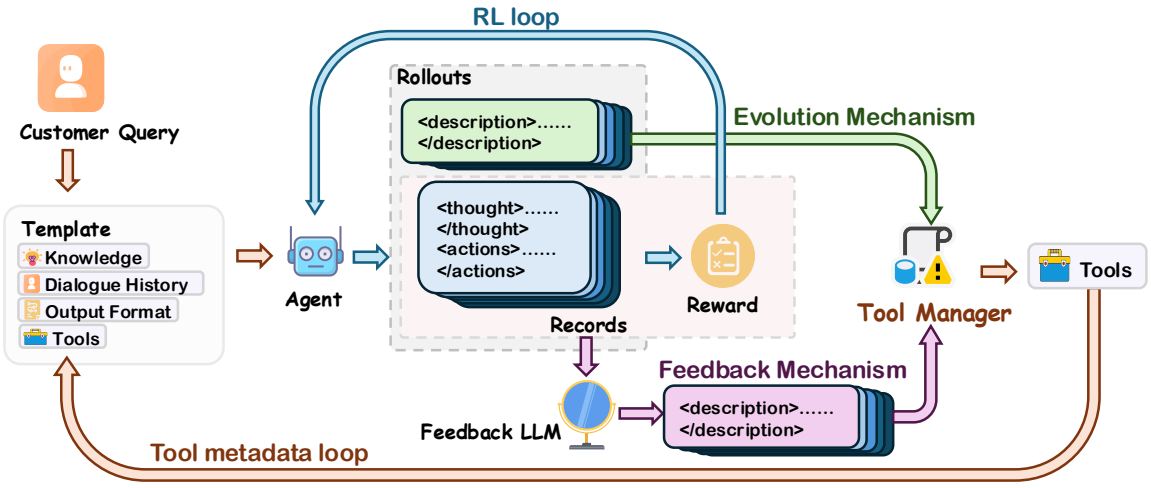

Figure 2: The overall architecture of the ToolDNA framework. The system operates on two synergistic loops: (1) The RL Loop (blue arrow) optimizes the agent's policy to generate reasoning traces, actions, and metadata updates. (2) The Tool Metadata Loop (brown arrow) dynamically evolves the tool definitions stored in the Tool Manager. This loop is fed by two streams: the Evolution Mechanism (green stream), where the policy network proposes metadata refinements during successful rollouts, and the Feedback Mechanism (purple stream), where a dedicated Feedback LLM distills insights from historical failures. Validated updates are synchronized back to the Tool Manager to enhance future iterations.

refining strategies via multi-dimensional rewards. Simultaneously, the **Tool metadata loop** (brown in Fig. 2), anchored by the Tool Manager, evolves metadata through two streams: the **evolution mechanism** (green in Fig. 2), which utilizes policy-generated candidates, and the **feedback mechanism** (purple in Fig. 2), which incorporates refinements from a Feedback LLM. These loops mutually reinforce one another—optimizing invocations with the latest metadata while enriching descriptions based on execution outcomes—directly bridging the information gap inherent in traditional rewards.

3.1 RL Loop

The objective of RL loop is to integrate external tools into the reasoning trajectories of LLMs to solve user tasks. As visualized in the architectural framework (Fig. 2), this process unfolds within the RL loop, where the policy network π_θ generates tool invocations (<actions>), reasoning logic (<thought>) and updated tool metadata (<description>) conditioned on the $\mathcal{T}_{\text{current}}$ (reflecting ongoing tool metadata evolution).

We model the toolset as an integral part of the environment, extending the RL objective function to incorporate tool interactions: $\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(\cdot|x,\mathcal{T})}\left[r_\phi(x,y)\right] - \beta D_{\text{KL}}\left[\pi_\theta(y|x,\mathcal{T})\|\pi_{\text{ref}}(y|x,\mathcal{T})\right]$. To stabilize policy optimization and avoid reliance on value function approximation, we adopt GRPO (Shao et al., 2024), which explicitly incorporates the dynamically evolving $\mathcal{T}_{\text{current}}$ (updated by the Tool Manager in each iteration). Using this notation, the GRPO objective function becomes:

$$\mathcal{J}(\theta) = E_{\substack{x\sim\mathcal{D}, \\ \{y_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|x,\mathcal{T}_{\text{current}})}}\left[\frac{1}{G}\sum_{i=1}^{G}\min\left(\rho_i(\theta)A_i,\right.\right.$$
$$\left.\left.\text{clip}\left(\rho_i(\theta),1-\epsilon,1+\epsilon\right)A_i\right) - \beta\, D_{\text{KL}}[\pi_\theta\|\pi_{\text{ref}}]\right]$$
$$s.t. \quad \rho_i(\theta) = \frac{\pi_\theta(y_i|x,\mathcal{T}_{\text{current}})}{\pi_{\text{old}}(y_i|x,\mathcal{T}_{\text{current}})}, \tag{1}$$

where $\rho_i(\theta)$ is ratio of the new policy probability to the old policy probability, conditioned on $\mathcal{T}_{\text{current}}$. Due to space constraints and as they align with the original GRPO formulation in (Shao et al., 2024), other hyperparameters and implementation details are not repeated here. A_i represents the advantage value for the i-th response in the group, computed based on the relative rewards of outputs within the group as:

$$A_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^{G})}{\text{std}(\{r_j\}_{j=1}^{G})},$$

where r_i is the reward for the i-th response, and mean$(\cdot)$, std$(\cdot)$ denote the mean and standard deviation of rewards within the group, respectively. We design multi-dimensional rule-based reward functions $r_\phi(x,y)$ to measure the alignment with

ground truth and presence of fully valid invocations that satisfy user requirements. Detailed definitions and calculations are provided in the Appendix C.

3.2 Tool Metadata Loop

The **Tool Manager** constitutes the core component of the tool metadata loop, responsible for maintaining and dynamically updating tool metadata ($\mathcal{T}$) through rigorous validation. Built on the verl framework (Sheng et al., 2024), it ensures metadata integrity while adapting flexibly to diverse infrastructure environments.

As initiated in the optimization process (see **Algorithm 1** in Appendix A), the Tool Manager starts with $\mathcal{T}_{\text{init}}$—a human-curated baseline with tool metadata including taxonomies, descriptive text, and usage templates. In distributed training setups, a master node parses $\mathcal{T}_{\text{init}}$ into structured entries and synchronizes them, while worker nodes align their metadata with the master to eliminate inconsistencies. For local environments, $\mathcal{T}_{\text{init}}$ is parsed directly without synchronization overhead.

The Tool Manager ingests metadata candidates from two evolutionary pathways. All candidates undergo three-layer validation including lexical consistency, structural validity, and semantic compliance. The detailed implementation logic and pseudocode of this validation pipeline are provided in Appendix E. Only valid candidates trigger $\mathcal{T}$ updates (as formalized in the appendix algorithm)—ensuring evolutionary changes never break tool invocations. To close the loop between tool metadata evolution and policy learning, the Tool Manager updates the "Tool" section of the LLM prompt with the latest $\mathcal{T}$ at each training iteration. This ensures the policy network π_θ always trains on the most refined tool metadata. The metadata evolution process relies on two distinct but complementary methods: one driven by the policy network during RL rollouts, and another facilitated by a feedback network, both elaborated below.

3.2.1 Evolution Mechanism.

The evolution mechanism (the **green stream** in Fig. 2) is designed to achieve synergistic co-evolution of tool invocation policies and metadata. Through carefully designed prompts, the policy network is instructed to generate tool invocation sequences and updated tool metadata embedded within <description> tags. This dual-output process is integrated into the RL training loop, creating a coordinated optimization cycle where: (i) the policy π_θ generates rollouts $\{y_i\}$ conditioned on current $\mathcal{T}$; (ii) these are scored by $r_\phi(x, y)$ for invocation accuracy (format, tool/parameter matching); and (iii) high-quality rollouts (r_i=1) trigger metadata updates, where <description> content is extracted, validated, and integrated into $\mathcal{T}$ by the Tool Manager. Simultaneously, the policy network itself is optimized using GRPO, which maximizes the objective function in Eq. (1) to refine strategies, creating a self-reinforcing cycle where better policies yield higher-quality candidates, while updated $\mathcal{T}$ enhances subsequent policy training.

3.2.2 Feedback Mechanism.

The feedback mechanism (the **purple stream** in Fig. 2) provides a second, targeted pathway for metadata refinement by explicitly distilling insights from historical interaction data. While metadata generation is implicitly guided by policy weight updates, the feedback mechanism uses a dedicated LLM to analyze past performance and produce direct, actionable improvements. Operating periodically, the mechanism ensures: (i) historical interaction records—including RL rollouts, ground-truth invocations, and rewards—are compiled; and (ii) low-quality rollouts (r_i <1) trigger metadata updates, where a structured prompt(shown in Appendix B) guides the feedback LLM to identify flaws in current metadata (e.g., ambiguous parameters) while preserving core schemas. Refined metadata undergoes validation against business logic before being integrated into $\mathcal{T}$. This mechanism strengthens the co-evolution loop by addressing gaps identified in historical data, ensuring metadata evolves not just from successes but also from explicit analysis of past errors.

A critical challenge in automated metadata generation is the risk of over-constraint—evolving rules so rigid that they reject valid user paraphrases. ToolDNA inherently counterbalances this through its closed-loop design. If an overly strict rule causes the agent to fail on a valid variation (resulting in a sub-optimal reward), the trajectory is explicitly routed to the Feedback LLM. Instructed to prioritize generalizable guidance, the Feedback LLM diagnoses this over-constraint from the failure case and actively broadens or relaxes the rule in the subsequent iteration, ensuring the system maintains semantic flexibility and does not overfit to specific phrasing.

4 Experiments

4.1 Experimental Settings

Dataset. We constructed a dataset from 9,327 real-world customer service dialogues in automotive and e-commerce domains, prioritizing domain-specific data over generic benchmarks to avoid pre-training contamination. Through manual filtering, we identified 3,100 interactions requiring explicit tool invocation (e.g., inventory searches, order status checks) and split them into training (70%), validation (15%), and test (15%) sets. A held-out test set was critical to evaluate generalization to non-stationary, unseen customer queries. Each entry includes a structured prompt (incorporating role definitions, domain rules, tool sets, context, and current queries) and expert-annotated ground truth detailing optimal tool invocation logic and expected resolution. Detailed dataset specifications are provided in the Appendix C.

Model. We use Qwen2.5-7B-Instruct as the base model for its balanced capability-efficiency trade-off: it suffices for tool-invocation reasoning while remaining computationally feasible for RL training. Experimental details (including hardware configurations and training setups) are provided in Appendix C.

Evaluation Metrics. We adopted two types of metric balancing technical accuracy and business relevance. For technical accuracy, we used the precision in specific technical steps of tool invocation, including: format integrity (Format), successful parsing of tool calls (ToolParsed), alignment of tool names (ToolName), consistency of parameter names (ParamName), and validity of parameter values (ParamValue) for evaluation. These capture granular technical robustness in tool invocation. For business relevance, we considered exact matches with ground truth to reflect potential business impact, including (PerfectMatch) and (Resolution). Detailed definitions and calculations are provided in the Appendix C.

4.2 Main Results

We compare ToolDNA against three groups of state-of-the-art baselines (details are shown in Appendix B.2) using state-of-the-art PE, SFT and RL as tool learning strategies, the comparative results are shown in Table 1. Our analysis reveals three key findings: First, ToolDNA surpasses all baselines across all three categories in every metric, highlighting its superiority in both technical accuracy and business efficacy. While most baselines achieve high *format* scores, they consistently fail to identify correct tools and set accurate parameters. Second, RL-based techniques outperform PE and SFT baselines, demonstrating RL's strength in handling dynamic customer queries, aligning with findings in extant literature. Third, ToolDNA outperforms Baseline RL, underscoring its advantage in synergistically integrating RL with a feedback loop to enable co-evolution of tool invocation and metadata refinement.

To analyze performance variations of ToolDNA across different tasks, we evaluate its effectiveness based on tool characteristics. Specifically, tasks are categorized into four types according to frequency of use, parameter type, functional purpose, and tool quantity—each exhibiting distinct patterns in structural, semantic, and outcome metrics. We compare ToolDNA against a Qwen-2.5-7B-Instruct model with well-designed prompts, the detailed results of which are presented in Table 2.

In frequency-based groups, high-frequency tools naturally exhibit the largest gains in *PerfectMatch* (alignment with ground truth) and *Resolution* (problem-solving effectiveness), as their optimization directly scales user experience improvements. Analysis by parameter type reveals that structured tools (with predefined enumeration values) show dramatic *ParamValue* accuracy gains; the explicit articulation of parameter constraints significantly mitigates entry errors, a primary driver of task success. This underscores the necessity of pairing structured workflows with explicit schemas while advancing semantic parsing for unstructured, open-ended inputs to balance precision with flexibility.

Function-specific analysis uncovers nuanced benefits: Human Transfer tools demonstrate striking *ToolName* accuracy improvements. The evolved metadata clarifies boundary conditions, enabling the model to accurately recognize out-of-capability queries and trigger appropriate human handoffs—a critical safeguard for balancing automated resolution with operational safety.

Crucially, tool-type analysis highlights the impact of real-world data distribution on optimization efficacy. Single-tool interactions benefit uniformly across all dimensions, with massive, statistically significant surges in both *PerfectMatch* and *Resolution* ($p < 0.0001$), proving that refining individual tool logic drastically strengthens standalone success. In contrast, while multi-tool workflows maintain robust structural integrity (*Format*, *Tool-*

Category	Model	Technical accuracy metrics					Business relevance metrics	
		Format	ToolParsed	ToolName	ParamName	ParamValue	PerfectMatch	Resolution
PE	**GLM-4.5-Flash**	0.8550	0.8550	0.5650	0.5107	0.3753	0.2281	0.4392
	QwQ-32B	0.9386	0.9174	0.5519	0.5350	0.4301	0.2097	0.4852
	DeepSeek-V3-0324	0.9958	0.9915	0.6476	0.5934	0.4798	0.3036	0.5478
	DeepSeek-R1-0528	0.9335	0.9270	0.6288	0.5880	0.4764	0.2854	0.5408
	Doubao-seed-1.6-250615	0.9640	0.9809	0.6737	0.6102	0.4894	0.3263	0.5530
	Doubao-Seed-1.6-thinking-250715	0.9915	0.9915	0.6504	0.6070	0.4820	0.2945	0.5657
	Qwen-2.5-7B-Intruct	0.9449	0.9831	0.5095	0.5498	0.2913	0.1144	0.3475
SFT	**Qwen-2.5-7B-Intruct**	**1.0000**	**1.0000**	0.6472	0.5837	0.4492	0.3708	0.5021
RL	**ToolRL**(Qian et al., 2025)	**1.0000**	**1.0000**	0.7129	0.6536	0.5106	0.4386	0.5487
	ToolDNA	**1.0000**	**1.0000**	**0.7636**	**0.7088**	**0.5870**	**0.5027**	**0.6295**

Table 1: Test-set performance comparison between ToolDNA and baselines. Metrics are average scores. Best performance represented in bold case.

Group	Category	Variant	Technical accuracy metrics					Business relevance metrics	
			Format	ToolParsed	ToolName	ParamName	ParamValue	PerfectMatch	Resolution
Frequency	High	**Baseline**	0.9591	0.9889	0.5242	0.4907	0.2361	0.0780	0.3197
		ToolDNA	**1.0000**	**1.0000**	**0.8215**	**0.7286**	**0.6561**	**0.5353**	**0.7249**
	Low	**Baseline**	0.9350	0.9756	0.5041	0.5854	0.3293	0.1504	0.3496
		ToolDNA	**1.0000**	**1.0000**	**0.6592**	**0.6041**	**0.4061**	**0.3592**	**0.4122**
Paramter Type	Structured	**Baseline**	0.9496	0.9806	0.4535	0.5465	0.1861	0.1085	0.2055
		ToolDNA	**1.0000**	**1.0000**	**0.7587**	**0.7354**	**0.6206**	**0.6070**	**0.6265**
	Unstructured	**Baseline**	0.9455	0.9844	0.5759	0.5253	0.3755	0.1167	0.4630
		ToolDNA	**1.0000**	**1.0000**	**0.7296**	**0.6031**	**0.4533**	**0.2957**	**0.5253**
Function	Query	**Baseline**	0.9781	0.9868	0.6206	0.5395	0.3926	0.1403	0.5219
		ToolDNA	**1.0000**	**1.0000**	**0.7127**	**0.5877**	**0.4824**	**0.3070**	**0.5702**
	Operation	**Baseline**	0.9160	0.9771	0.4542	0.5878	0.2977	0.1069	0.2901
		ToolDNA	**1.0000**	**1.0000**	**0.6962**	**0.6461**	**0.4077**	**0.3769**	**0.4077**
	Human Transfer	**Baseline**	0.9295	0.9808	0.4103	0.4872	0.1026	0.0769	0.0962
		ToolDNA	**1.0000**	**1.0000**	**0.8301**	**0.8077**	**0.7244**	**0.7244**	**0.7244**
Tool Quantity	Single Tool	**Baseline**	0.9425	0.9839	0.5046	0.5655	0.3023	0.1172	0.3609
		ToolDNA	**1.0000**	**1.0000**	**0.7719**	**0.7385**	**0.6129**	**0.5184**	**0.6567**
	Multi Tools	**Baseline**	0.9730	0.9730	0.5676	0.3649	0.1622	**0.0811**	**0.1892**
		ToolDNA	**1.0000**	**1.0000**	**0.5676**	**0.2568**	**0.1216**	**0.0811**	0.1351

Table 2: Performance comparison between baseline and ToolDNA across different categories. Metrics are weighted averages based on tool usage counts. Best performance represented in bold case.

Parsed), we observed an apparent variance in outcome metrics like *Resolution*. However, contextualizing this within our enterprise traffic—where multi-tool scenarios represent less than 5% of the distribution (resulting in extreme data sparsity of merely 37 test samples)—reveals a different reality. Fisher's Exact Test confirms that this performance variance is statistically insignificant ($p = 0.7537$). Therefore, the fluctuation in multi-step orchestration is not an algorithmic degradation, but merely a byproduct of training data starvation.

Overall, the co-evolution optimization drives highly significant, robust improvements across nearly all structural (*Format*, *ToolParsed*), semantic (*ToolName*, *ParamName/Value*), and outcome metrics (*PerfectMatch*, *Resolution*), validating its broad impact on enhancing tool-driven dialogue agents within the bounds of data exposure.

4.3 Ablation Studies

To isolate contributions, we ablate two interrelated dimensions in ToolDNA: (1) core mechanisms (Feedback Mechanism, Evolution Mechanism, and Tool Metadata Loop); (2) reward designs (R1: Format + PerfectMatch; R2: weighted structural-semantic blend; R3: Format + 0.5×PerfectMatch + 0.5×Resolution). For rewards, R3 outperforms R1 and R2, as shown in table 3. R3 achieves the highest Resolution, as its dual-metric design aligns with real-world goals of balancing fine-grained accuracy and task resolution, surpassing R1's narrow structural focus and R2's suboptimal outcome trade-offs.

Across all rewards, removing the Feedback Mechanism degrades key metrics—e.g., R3's PerfectMatch score drops from 0.5027 to 0.4681, highlighting its role in semantic refinement. Removing the Evolution Mechanism similarly harms performance: R3's Format score falls from 1.0000 to 0.9979, underscoring its structural foundation. Ablating both causes steeper declines, confirming each component's necessity and their critical synergy. In sum, the dual-focus reward like R3 and the core mechanisms in ToolDNA are indispensable, validating their role in achieving superior performance in TOD by balancing structural precision

		Technical accuracy metrics					Business relevance metrics	
Reward	**Configuration**	**Format**	**ToolParsed**	**ToolName**	**ParamName**	**ParamValue**	**PerfectMatch**	**Resolution**
R1	**-Tool Metadata Loop**	**1.0000**	**1.0000**	0.7129	0.6536	0.5106	0.4386	0.5487
	-Feedback Mechanism	**1.0000**	**1.0000**	0.7426	0.6758	0.5561	0.4725	0.5953
	-Evolution Mechanism	**1.0000**	**1.0000**	0.7473	0.6836	0.5658	0.4841	0.6051
	ToolDNA	**1.0000**	**1.0000**	**0.7723**	**0.7053**	**0.5660**	**0.4851**	**0.6064**
R2	**-Tool Metadata Loop**	0.9958	0.9958	0.7182	0.6568	0.5307	0.4513	0.5763
	-Feedback Mechanism	**1.0000**	**1.0000**	0.7500	0.6822	0.5477	0.4682	0.5911
	-Evolution Mechanism	**1.0000**	**1.0000**	0.7558	0.7006	**0.5743**	0.4841	**0.6157**
	ToolDNA	**1.0000**	**1.0000**	**0.7798**	**0.7138**	0.5691	**0.4915**	0.6106
R3	**-Tool Metadata Loop**	0.9873	0.9852	0.7341	0.6928	0.5625	0.4449	0.6186
	-Feedback Mechanism	0.9979	0.9979	0.7628	0.7053	0.5702	0.4681	0.6255
	-Evolution Mechanism	**1.0000**	**1.0000**	0.7505	0.6921	**0.5892**	0.4841	**0.6348**
	ToolDNA	**1.0000**	**1.0000**	**0.7636**	**0.7088**	0.5870	**0.5027**	0.6295

Table 3: Ablative performance across different configurations. Metrics are averaged over dialogues on test set. Best performance represented in bold case.

and user-centric practical outcomes. Furthermore, our decoupling analysis (Appendix F) confirms that ToolDNA outperforms ToolRL baselines equipped with evolved metadata, validating the necessity of co-evolution.

In sum, the dual-focus reward like R3 and the core mechanisms in ToolDNA are indispensable, validating their role in achieving superior performance in TOD by balancing structural precision and user-centric practical outcomes. Furthermore, our decoupling analysis (Appendix F) confirms that ToolDNA outperforms ToolRL baselines equipped with evolved metadata, validating the necessity of co-evolution.

To verify that the metadata optimizations learned by ToolDNA are genuinely generalizable and not merely overfitting to the training model's idiosyncrasies, we conducted a zero-shot cross-model evaluation. We directly applied the evolved tool metadata ($\mathcal{M}_{evolved}$) to a completely distinct LLM architecture, GLM-4.5-Flash. The results demonstrate robust zero-shot improvements: *PerfectMatch* increased from 0.2281 to 0.2512, and *Resolution* improved from 0.4392 to 0.4690. This confirms that ToolDNA converges toward structurally concise, standardized, and human-readable schemas (as further evidenced by the linguistic analysis in Section 4.4) that universally benefit different LLM families, demonstrating the plug-and-play potential of our evolved metadata.

4.4 Interpretability Analysis

To further understand how the ToolDNA refines tool metadata, we conduct a semantic and linguistic analysis of the evolving tool metadata $\mathcal{T}$ across training iterations. This analysis focuses on four metrics: normalized edit distance, semantic similarity (Herbold, 2024), Type-Token Ratio (TTR) (Rosillo-Rodes et al., 2025), and syntactic tree depth (Qi et al., 2020)—each capturing distinct aspects of metadata quality and adaptability, the results of which are shown in Table 4.

Metric	**Initial**	**Updated**
Normalized Edit Distance	0.3811	
Semantic Similarity (Cosine)	0.9400	
Type-Token Ratio (TTR)	0.2288	0.1953
Syntactic Tree Depth	8	7

Table 4: Semantic and linguistic metrics of tool metadata.

Normalized Levenshtein edit distance reveals clear revisions (0.3811 compared with initial metadata), reflecting the framework's dynamic refinement of phrasing and parameter clarity. Sequential monitoring confirms that the magnitude of these updates decays across iterations, indicating that the metadata converges to a stable state without oscillation. Critically, semantic intent remains preserved, with BCE-embedding (Xiao et al., 2023) cosine similarity measuring 0.9400. This alignment confirms validation mechanisms successfully enforce consistency in core functionality. The coexistence of moderate textual edits and high semantic similarity demonstrates targeted adaptation—optimizing metadata without functional degradation.

Linguistic analysis via Stanza (Qi et al., 2020) further shows reduced lexical diversity (TTR: 0.2288 to 0.1953) and simplified syntactic structures (tree depth: 8 to 7), indicating standardized vocabulary and streamlined expressions. This enhances LLM parseability while reducing user interpretation effort. Baseline descriptions' higher complexity often introduced technical jargon and parsing errors, whereas optimized versions balance precision and usability through consistent terminology and simpler constructions—improving res-

olution rates without semantic ambiguity. High semantic similarity ensures critical operational details persist, aligning with service efficiency objectives of TOD systems.

4.5 Case Study

To empirically demonstrate how tool metadata refinement resolves parameter-related invocation errors, we present two representative cases focusing on critical parameters:'query' for the inventory search tool (shown in Table 5) and 'periods' for the loan calculation tool(shown in Appendix D).

Case: Car Inventory Search Tool ('search_used_cars')
Initial
Metadata (Excerpt):
Parameters: query(str): Search term (e.g.,"BMW X3").
Action: search_used_cars("query": "Passat")
Scores: ParamValue = 0.5; Total Reward = 0.625
Refined
Metadata (Excerpt):
Parameters: query(str): Search term (e.g., "BMW X3"). When users specify a model, query must exactly match.
Action: search_used_cars("query": "Passat380")
Scores: ParamValue = 1.0; Total Reward = 1.0

Table 5: A case study demonstrating parameter error resolution through metadata refinement in ToolDNA.

As shown in the Car Inventory Search tool case shown in Table 5, ambiguity in the initial metadata of 'query' (e.g., allowing vague terms like "Passat") resulted in overly broad searches. The refined metadata introduces a critical rule: "when users specify a model, query must exactly match." This drives the model to use the precise term "Passat380," resolving ambiguity and increasing the parameter value.

The case demonstrates that explicit, constraint-driven tool metadata directly mitigate parameter errors. By translating implicit business rules into clear linguistic guidance, the proposed framework bridges the gap between model behavior and practical requirements—validating its effectiveness in real-world scenarios.

4.6 Practical Deployment and Trade-offs

While the dual-loop evolution process introduces additional offline computational overhead—taking approximately 36 hours compared to 27 hours for the standard RL baseline (a roughly 33% increase)—this presents a highly favorable trade-off for practical adoption. It substitutes a modest, one-time offline compute cost for the continuous, expensive human labor typically required for manual schema debugging and prompt engineering. Crucially, the evolved metadata introduces absolutely zero additional latency during online inference. Furthermore, by standardizing vocabulary and simplifying syntactic structures (as evidenced in Section 4.4), the optimized schemas reduce the cognitive attention burden on the LLM, actively enhancing scalability and reliability in long-context, multi-turn dialogues.

5 Conclusion

This work addresses critical reliability barriers in tool-augmented TOD systems, where ambiguous tool specifications and costly manual refinement constrain its performance. We introduce ToolDNA, a novel dynamic adaptation framework that pioneers the autonomous co-evolution of policy networks and tool metadata. Extensive experiments on real-world dataset showcase the superiority of ToolDNA over SOTA methods. ToolDNA establishes the new paradigm for self-evolving dialogue agent that align tool invocation with operational requirements while minimizing human calibration. Future work may extend this framework to develop the genome of tools to achieve orchestration of more divergent tool sets.

6 Limitations

While ToolDNA shows significant gains, several limitations remain. First, our evaluation is domain-specific (automotive/e-commerce); generalization to open-domain or scientific APIs requires further study. Second, our validation focuses on the 7B scale; larger models might offer different self-correction dynamics. Third, the lack of significant gains in multi-tool scenarios is primarily due to data sparsity (under 5% of traffic), not algorithmic failure; we expect scaling as multi-step trajectories increase. Finally, moving from offline to live deployment poses risks from distribution shifts. To ensure safety, ToolDNA utilizes engineering guardrails (e.g., Differential Validation) and we propose an asynchronous Shadow Mode protocol for stable online evolution.

Acknowledgements

We acknowledge support by National Natural Science Foundation of China (Grants No. 72495123,12271012,72572003). We also acknowledge the support provided by the High-Performance Computing Platform of Peking University.

Ethics Statement

This work utilizes a dataset derived from real-world customer service interactions. We strictly adhere to ethical guidelines regarding data privacy and usage. **Privacy and Anonymization:** Prior to model training and analysis, all Personally Identifiable Information (PII)—including user names, phone numbers, addresses, and vehicle identification numbers—was rigorously scrubbed using automated regex filtering followed by manual verification. User IDs were hashed to ensure anonymity. **Data Usage:** The dataset is used solely for academic research purposes under strict access controls. **Potential Impact:** While ToolDNA empowers agents to autonomously update tool definitions, this capability carries potential risks of generating unsafe or biased tool descriptions. To mitigate this, our framework incorporates a strict three-layer validation mechanism (lexical, structural, semantic) as described in Section 3.2, ensuring that evolved metadata remains within safe operational boundaries.

References

Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025. Research: Learning to reason with search for llms via reinforcement learning. *Preprint*, arXiv:2503.19470.

Michelle Elizabeth, Morgan Veyret, Miguel Couceiro, Ondrej Dusek, and Lina M. Rojas Barahona. 2025. Exploring ReAct prompting for task-oriented dialogue: Insights and shortcomings. In *Proceedings of the 15th International Workshop on Spoken Dialogue Systems Technology*, pages 143–153, Bilbao, Spain. Association for Computational Linguistics.

Yihao Feng, Shentao Yang, Shujian Zhang, Jianguo Zhang, Caiming Xiong, Mingyuan Zhou, and Huan Wang. 2023. Fantastic rewards and how to tame them: A case study on reward learning for task-oriented dialogue systems. *Preprint*, arXiv:2302.10342.

Steffen Herbold. 2024. Semantic similarity prediction is better than other semantic similarity measures. *Preprint*, arXiv:2309.12697.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2).

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *Preprint*, arXiv:2503.09516.

Henger Li, Shuangjie You, Flavio Di Palo, Yiyue Qian, and Ayush Jain. 2025a. Verification-guided context optimization for tool calling via hierarchical llms-as-editors. *Preprint*, arXiv:2512.13860.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025b. Search-o1: Agentic search-enhanced large reasoning models. *Preprint*, arXiv:2501.05366.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025c. Torl: Scaling tool-integrated rl. *Preprint*, arXiv:2503.23383.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2024a. Toolace: Winning the points of llm function calling. *Preprint*, arXiv:2409.00920.

Yulong Liu, Yunlong Yuan, Chunwei Wang, Jianhua Han, Yongqiang Ma, Li Zhang, Nanning Zheng, and Hang Xu. 2024b. From summary to action: Enhancing large language models for complex tasks with open world apis. *Preprint*, arXiv:2402.18157.

Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. 2024. Generative reward models. *Preprint*, arXiv:2410.12832.

Lingbo Mo, Shun Jiang, Akash Maharaj, Bernard Hishamunda, and Yunyao Li. 2024. Hiertod: A task-oriented dialogue system driven by hierarchical goals. *Preprint*, arXiv:2411.07152.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *Preprint*, arXiv:2504.13958.

Pablo Rosillo-Rodes, Maxi San Miguel, and David Sánchez. 2025. Entropy and type-token ratio in gigaword corpora. *Physical Review Research*, 7(3).

Nicholas Roth, Christopher Hidey, Lucas Spangher, William F. Arnold, Chang Ye, Nick Masiewicki, Jinoo Baek, Peter Grabowski, and Eugene Ie. 2025. Factored agents: Decoupling in-context learning and memorization for robust tool use. *Preprint*, arXiv:2503.22931.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Ivan Sekulic, Silvia Terragni, Victor Guimarães, Nghia Khau, Bruna Guedes, Modestas Filipavicius, Andre Ferreira Manso, and Roland Mathis. 2024. Reliable LLM-based user simulator for task-oriented dialogue systems. In *Proceedings of the 1st Workshop on Simulating Conversational Intelligence in Chat (SCI-CHAT 2024)*, pages 19–35, St. Julians, Malta. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025a. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *Preprint*, arXiv:2503.05592.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. 2025b. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *Preprint*, arXiv:2501.03124.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604, Bangkok, Thailand. Association for Computational Linguistics.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025. Acting less is reasoning more! teaching model to act efficiently. *Preprint*, arXiv:2504.14870.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. 2025. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *Preprint*, arXiv:2511.16043.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. 2025. Steptool: Enhancing multi-step tool usage in llms through step-grained reinforcement learning. *Preprint*, arXiv:2410.07745.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awalgaonkar, Rithesh R N, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, and 2 others. 2025a. xLAM: A family of large action models to empower AI agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11583–11597, Albuquerque, New Mexico. Association for Computational Linguistics.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025b. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *Preprint*, arXiv:2505.00024.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. Processbench: Identifying process errors in mathematical reasoning. *Preprint*, arXiv:2412.06559.

A Joint Optimization Algorithm

In this appendix, we present the detailed pseudocode for the joint optimization process of ToolDNA, as summarized in Algorithm 1. The training procedure integrates Policy Optimization with Tool Metadata Evolution in a unified loop. The algorithm takes a policy model π_θ, a reference model π_{ref}, and an initial tool metadata set $\mathcal{T}_{init}$ as inputs. The optimization proceeds in iterative cycles (Lines 2–24). In each iteration, the policy generates rollouts conditioned on the current metadata $\mathcal{T}_{current}$. These rollouts are evaluated to compute rewards r_i.

Successful rollouts trigger the Evolution Path(Lines 7–11), where metadata proposals are extracted directly from the `<description>` tag and validated. We strictly enforce a binary gating mechanism ($r_i = 1$) here. This is because any partial reward ($r_i < 1$) implies execution defects or parameter hallucinations. Using intermediate soft thresholds would risk incorporating noise into the schema. Instead, we purposefully rely on the distinct roles of the two paths: only perfect trajectories drive evolution, while sub-optimal trajectories are routed to the Feedback Path.

Consequently, failed rollouts ($r_i < 1$) trigger the Feedback Path (Lines 12–19), where a separate feedback model analyzes the errors to propose diagnostic refinements. The Tool Manager ensures all updates undergo strict lexical, structural, and semantic validation before committing them to $\mathcal{T}_{current}$, ensuring stability throughout the reinforcement learning process.

B Prompt Templates

Prompt Template for Tool Metadata Optimization

You are a tool Metadata optimizer. Analyze the provided ground-truth tool use, model-generated solution, and original tool description to identify flaws and generate a revised description. Adhere to the following constraints:
1.Tool name, parameter types, and parameters must remain unchanged.
2.The first usage example must be preserved.
3.Prioritize generalizable guidance.
Data:
Ground-truth: [tool invocation from ground truth]
Solution:[Model-generated solution]
Original Description: [Current metadata in Tool Manager]
Output Format:
<description>
Tool Name: [Tool Name]
Tool Description:[Revised description, with critical rules**bolded**]
Required Parameters:
[param1]([type]): [Clarified explanation]
[param2] ([type]):[Clarified explanation]
Usage Examples:[Original example]
</description>

Figure 3: Template for the Feedback LLM

The feedback mechanism draws from a comprehensive log of historical interactions, including RL-generated rollouts (reasoning trajectories and tool invocations), ground-truth invocations, and corresponding reward scores. These data are formatted into a structured prompt (Fig. 3) that guides the feedback LLM to refine the metadata while preserving core functionality: tool names, parameter types, and mandatory fields remain unchanged, ensuring compatibility with existing training.

The process operates on a per-record basis, starting with individual entries in the historical interaction log. For each log entry, the system first identifies the specific tool invoked in the ground-truth solution. It then retrieves the current metadata of this identified tool from the Tool Manager, establishing a baseline for refinement. By analyzing the gap between the model-generated solution (including any invocation errors or ambiguities) and the ground-truth invocation within that record, the system pinpointes specific flaws in the retrieved metadata——such as unclear parameter definitions that led to incorrect values, or missing usage notes that resulted in tool misuse. These identified flaws guide the targeted refinement of the tool's metadata, ensuring improvements directly address the issues observed in actual interaction data.

Algorithm 1 Joint Optimization of Policy and Metadata in ToolDNA

Require: Policy model π_θ, reference $\pi_{\theta_{\text{ref}}}$, dataset $\mathcal{D}$, reward function $r_\phi(\cdot)$, feedback model F, initial metadata $\mathcal{T}_{\text{init}}$
Ensure: Optimized policy π_θ^* and evolved metadata $\mathcal{T}_{\text{current}}$

Initialize Tool Manager; $\mathcal{T}_{\text{current}} \leftarrow \mathcal{T}_{\text{init}}$
for each training iteration $t = 1, \ldots, T$ **do**
  Construct prompt with $\mathcal{T}_{\text{current}}$
  Generate rollouts $\{y_i\}_{i=1}^{G} \sim \pi_\theta(\cdot|x, \mathcal{T}_{\text{current}})$
  Compute rewards $r_i = r_\phi(x, y_i)$
  Update policy π_θ via GRPO with $\epsilon = 0.2, \lambda_{\text{RL}} = 0.001$
  for each rollout y_i **do**
    if $r_i = 1$ **then** ▷ Evolution path
      Extract metadata from `<description>` block in y_i
      Tool Manager validates the extracted metadata
        (lexical, structural, and semantic checks)
      if validation passes **then**
        Update $\mathcal{T}_{\text{current}}$
      end if
    else ▷ Feedback path
      Invoke feedback model F with (x, y_i, r_i) to analyze errors
      Obtain refined metadata proposal from F
      Tool Manager validates the refinement
        (structural, type/range, and business-rule checks)
      if validation passes **then**
        Update $\mathcal{T}_{\text{current}}$
      end if
    end if
  end for
end for
return $\pi_\theta^*, \mathcal{T}_{\text{current}}$

C Experimental Settings

C.1 Dataset Details

We constructed our dataset from 9,327 authentic customer service dialogues collected post-deployment from live automotive and e-commerce systems, capturing genuine user interactions with operational platforms. From these dialogues, 3,100 interactions requiring explicit invocation of real-system tools (e.g., inventory lookup, order status verification, and warranty query) were identified through manual filtering, ensuring strict relevance to tool-augmented task scenarios.

The dataset was partitioned into training (70%), validation (15%), and test (15%) splits. A held-out test set was deliberately retained to evaluate generalization performance, as customer queries in real-world scenarios are inherently non-stationary—evaluating on unseen dialogues ensures the model's robustness to real-world variability, a critical requirement for practical deployment.

The ground truth for each entry was meticulously annotated by domain experts with proficiency in industry protocols, sales strategies, and tool functionalities. Due to the need to encode professional knowledge and strategic judgment, the annotation process was highly labor-intensive, with a maximum throughput of 5 entries per expert per hour. All annotations underwent dual rounds of quality inspection: first by the annotation team for procedural consistency, then by the algorithm team for alignment with task objectives, resulting in high-quality labeled data with stringent accuracy guarantees. Each entry in the dataset follows a structured format: the prompt template includes a role definition, domain-specific background knowledge (e.g., "Warranties for new vehicles expire after 3 years"), a set of real-system tools ($\mathcal{T}$), task-specific guidelines (e.g., "Prioritize accuracy for price inquiries"), output format constraints, dialogue history (retaining up to 5 prior turns to preserve context), and the current user query. The corresponding ground truth details the optimal tool invocation logic (tool selection, parameter configuration) and expected resolution, serving as the gold standard for evaluation.

To explicitly demonstrate the reasoning complexity required by our real-world enterprise dataset, we detail its structural statistics: the tool library contains 10 unique tools with a total of 24 distinct parameters (averaging 2.4 parameters per tool). Crucially, the parameters present a challenging distribution: 58% are structured fields (e.g., strict Enums or numerical ranges requiring exact adherence), while 42% are unstructured free-text fields requiring fuzzy semantic reasoning. Furthermore, driven by the multi-turn dialogue history, input sequences reach a maximum length of 13,000 tokens. This complex structural mix thoroughly tests the agent's ability to maintain long-context memory and strictly adhere to complex tool schemas under heavy cognitive load.

We opted for this domain-specific, real-system dataset over prevalent open-source benchmarks for three key reasons. First, open-source datasets typically include only a small number of tools per task, with these tools often being simplified or abstracted versions of real-world systems—they fail to capture the complexity of production-grade tool ecosystems where multiple interdependent tools may be required for a single query. Second, open-source tool-augmented datasets often rely on synthetic or crowdsourced data, lacking the authenticity of real user queries and operational scenarios, which introduces gaps between benchmark performance and real-world utility. Third, unlike our expert-annotated ground truth with rigorous quality control, open-source datasets frequently use crowdsourced annotations with variable expertise, struggling to encode the domain-specific knowledge and strategic judgment required for effective tool invocation in professional customer service scenarios. Thus, our dataset better supports the evaluation of tool-augmented models in practical, industry-aligned contexts.

To facilitate reproducibility while strictly adhering to user privacy regulations and protecting proprietary business strategies, we adopt a hybrid open-source strategy. We release the complete codebase and provide a partial release of the dataset comprising a de-identified subset of the real-world dialogues. This subset has been rigorously scrubbed of PII and specific commercial protocols but retains the semantic complexity necessary to replicate our key experimental findings.

C.2 Baselines and Implementation

Baselines. We compared ToolDNA against three groups of state-of-the-art methods as baselines, i.e., LLMs with prompt engineering, LLMs with SFT, and LLMs with RL. Baselines in Group 1 consist of both open-sourced and commercial LLMs, including GLM-4.5-Flash (Zhipu AI), QwQ-32B (Alibaba), DeepSeek-V3-

0324(DeepSeek), DeepSeek-R1-0528 (DeepSeek, reasoning-specialized), doubao-seed-1.6-250615 (ByteDance), Doubao-Seed-1.6-thinking-250715 (ByteDance, reasoning-specialized), Qwen-2.5-7B-Instruct (Alibaba). For the implementation of methods in Group 1, human-generated tool description was used as input and well-designed prompts (see Fig. 4) were used to achieve tool invocation using LLMs. For baselines in Group 2, we use the well-labeled Q&A data with tool invocation to SFT a Qwen-2.5-7B-Instruct model for comparison. For baselines in Group 3, we implemented a baseline RL. All baselines and ToolDNA share identical inference-time compute budgets ($\leq$13K tokens) to isolate the impact of method rather than scale.

Prompt Template for Tool-augmented LLM

Role: You are a domain-specific intelligent agent (e.g., industry-focused customer advisor). Fulfill these core responsibilities: (1) Adhere to implicit domain rules (e.g., business policies, operational workflows); (2) Leverage tools only when critical information is missing, avoiding fabrication.
Domain Background: Domain-specific constraints (e.g., service protocols, eligibility criteria) govern responses. Industry rules/policies (e.g., Used cars $\leq$8 years old"; Warranties cover 60-day mechanical defects").
Tool Set (T): Real-system tools with functional specs (tool metadata).
Task Guidelines: Prioritization rules (e.g., Prioritize budget-matching cars"; Escalate for video inspection requests").
Output Constraints: Format requirements (e.g., JSON-structured tool calls; natural language replies $\leq$150 words).
Dialogue History: Up to 5 prior interaction turns (preserving contextual continuity).
Current Query: User's task-specific request (e.g., ``Find a white SUV under $20k'').
Please select appropriate tools for current query.

Figure 4: Template for Tool-augmented LLM

Base Model. For the implementation of ToolDNA, we adopted Qwen2.5-7B-Instruct as the base model, selected for its balance of capability and efficiency: its 7B parameters suffice for tool-invocation reasoning while remaining feasible for RL training on an 8-GPU cluster (NVIDIA A100), avoiding the resource overhead of larger models. The key experiments in this work—including the supervised fine-tuning (SFT) baseline, the ToolRL reinforcement learning baseline, and our proposed ToolDNA framework—were implemented and trained on this consistent base model, ensuring fair comparison by isolating methodological differences from variations in underlying model architecture.

Implementation Details. Supervised Fine-Tuning (SFT) Phase The SFT phase aimed to align the model with task-specific input-output patterns using structured dialogue data, with key configurations as follows:

- **Data Configuration**: Training data was loaded from train_files and validated on val_files (test split). Inputs and outputs were structured using prompt_key="question" and response_key="answer", with prompt_dict_keys=['question'] and response_dict_keys=['answer'] to extract relevant fields from the dataset. The maximum sequence length was set to 12,288 tokens to accommodate extended dialogue contexts, with overlong sequences truncated via truncation='error' to avoid information loss. use_remove_padding=true was enabled to optimize token processing efficiency by removing padding during computation.

- **Batch & Parallel Training**: The total train_batch_size=8 was distributed across 8 GPUs, with micro_batch_size_per_gpu=1 and gradient_accumulation_steps=4 to balance memory usage and gradient stability. Sequence parallelism was configured with ulysses_sequence_parallel_size=2 to handle long sequences efficiently.

- **Model & Optimization**: The SFT model was initialized from the Qwen-2.5-7B-Instruct. Training used bf16 precision for computational efficiency. The optimizer adopted a cosine learning rate scheduler with an initial learning rate of 1e-6, a warm-up phase covering 10% of total steps (warmup_steps_ratio=0.1), and gradient clipping (clip_grad=1.0) to prevent instability. Weight decay was set to 0.01 to regularize training.

- **Training Scheduling**: The SFT phase ran for 6 epochs (total_epochs=6), with logs tracked via the console (logger=['console']). Checkpoints were saved locally to trainer.default_local_dir (configurable via script), with no HDFS backup by default.

Reinforcement Learning (RL) Phase The RL phase optimized the SFT model using the GRPO

(Generalized Reinforcement Learning with Policy Optimization) algorithm, focusing on improving response quality and tool usage via reward signals. Key configurations included:

- **Algorithm & Data Setup**: The GRPO algorithm was used with algorithm.adv_estimator=grpo for advantage estimation. Training data (from train_files) and validation data (from val_files) maintained consistency with SFT in maximum prompt length (12,288 tokens) but restricted responses to 1,024 tokens (max_response_length=1024). Overlong prompts were filtered out (filter_overlong_prompts=True) to avoid truncation errors. The total train_batch_size=16 was distributed across 8 GPUs (1 micro-batch per GPU, ppo_micro_batch_size_per_gpu=1), with mini-batch partitioning (ppo_mini_batch_size=4) for stable gradient updates.

- **Actor & Reference Models**: The actor model was initialized from a pre-trained checkpoint (actor_rollout_ref.model.path) with enable_gradient_checkpointing=True to reduce memory usage. KL divergence regularization was applied (use_kl_loss=True) with a coefficient of 0.001 (kl_loss_coef=0.001) and low-variance estimation (kl_loss_type=low_var_kl) to prevent abrupt policy shifts. The reference model shared the actor's architecture but used parameter offloading (ref.fsdp_config.param_offload=True) to save memory, with log probabilities computed in micro-batches (log_prob_micro_batch_size_per_gpu=1).

- **Rollout & Inference**: Rollouts (trajectory generation) used the vLLM engine (rollout.name=vllm) for high throughput, configured with tensor_model_parallel_size=1, gpu_memory_utilization=0.7, and a maximum of 12,288 batched tokens per step (max_prompt_length=12,288) and restricted responses to 1,024 tokens (max_response_length=1024). A rollout group size of 5 (rollout.n=5) supported GRPO's advantage estimation, with sampling parameters set to temperature=1.0 (for exploration) and ignore_eos=False (to ensure natural termination).

- **Advantage Estimation**: In strict alignment with the GRPO algorithm, our implementation does not utilize a critic model or value function approximation, thereby avoiding the computational overhead associated with Generalized Advantage Estimation (GAE). Instead, advantages were estimated directly by computing the relative reward scores within each rollout group (group size of $N = 5$). These group-relative advantages were then normalized by their standard deviation (`norm_adv_by_std_in_grpo=True`) to stabilize policy updates.

- **Training Scheduling & Stability**: The RL phase ran for 6 epochs (trainer.total_epochs=6), with checkpoints saved every 100 steps (save_freq=100) and validation every 5 steps). Logs were tracked via both console and Weights & Biases (logger=['console', 'wandb']). Distributed training used FSDP (Fully Sharded Data Parallel) with no parameter/optimizer offloading for the actor (prioritizing speed) and early stopping triggered by consecutive drops in validation reward variance.

More detailed configurations can be referred to in the code package included in the supplementary materials.

C.3 Detailed Definitions of Metrics and Reward Function Designs

This appendix provides detailed definitions and calculations for the evaluation metrics and reward function referenced in the main text, which measure both technical accuracy and business relevance of tool invocations, and serve as the basis for computing advantage values.

C.3.1 Evaluation Metrics.

We define two categories of metrics to assess tool invocation performance, aligned with the technical and business dimensions outlined in the main text:

Technical Accuracy Metrics. These metrics quantify precision in specific technical steps of tool invocation:

- Format: Assesses whether the output contains all required structural tags.

$$\text{Format} = \begin{cases} 1, & \text{if the output contains all required tags} \\ 0, & \text{otherwise} \end{cases}$$

- ToolParsed: Evaluates whether tool calls within the "actions" tag can be successfully parsed.

$$\text{ToolParsed} = \begin{cases} 1, & \text{if the content within "actions" tag can be parsed as tool calls} \\ 0, & \text{otherwise} \end{cases}$$

- ToolName: Measures alignment between predicted and ground-truth tool names.

$$\text{ToolName} = \begin{cases} 1, & \text{if the invoked tool names exactly matches the ground truth} \\ 0.5, & \text{if there is partial overlap} \\ 0, & \text{otherwise} \end{cases}$$

- ParamName: Assesses consistency of parameter names with ground truth.

$$\text{ParamName} = \begin{cases} 1, & \text{if all parameter names exactly match the ground truth} \\ 0.5, & \text{if all tools have partially matched parameter names} \\ 0, & \text{otherwise} \end{cases}$$

- ParamValue: Evaluates validity of parameter values against ground truth.

$$\text{ParamValue} = \begin{cases} 1, & \text{if all parameter values exactly match the ground truth} \\ 0.5, & \text{if there are partially matched parameter values} \\ 0, & \text{otherwise} \end{cases}$$

Business Relevance Metrics.

These metrics reflect the practical impact of tool invocations in resolving user queries:

- PerfectMatch: Indicates exact matches with ground-truth tool invocations.

$$\text{PerfectMatch} = \begin{cases} 1, & \text{if there exists valid tool invocations matching the ground truth} \\ 0, & \text{otherwise} \end{cases}$$

- Resolution: Specifically quantifies Basic Resolution, capturing scenarios where queries are resolved despite non-exact matches—critical for real-world applications where strict exact matches may be unnecessarily rigid. This metric is distinct from PerfectMatch, focusing on practical problem-solving effectiveness rather than strict fidelity to ground truth.

$$\text{Resolution} = \begin{cases} 1, & \text{if Format = ToolParsed = 1, and ToolName, ParamName, and ParamValue are all positive} \\ 0, & \text{otherwise} \end{cases}$$

This metric accommodates practical scenarios such as omitted optional parameters (where defaults suffice) or search queries with semantic overlap (e.g., "2023 used cars" versus "used 2023 vehicles") that still retrieve relevant information. By focusing exclusively on non-exact but effective resolutions, it complements PerfectMatch to provide a more comprehensive assessment of real-world utility.

Anti-Inflation Rules for Resolution Metric. To prevent score inflation and accurately reflect real-world business utility, the programmatic evaluation logic strictly governs when partial scores (0.5) can be awarded. A rollout with hallucinated parameters or substantial errors cannot mathematically achieve a high Resolution score due to the following deterministic logic gates:

- **Under-selection is a Strict Failure (0.0):** If the agent misses any required parameter present in the ground truth (e.g., checking only `['price']` when `['price', 'mileage']` is required), the core user intent is considered unresolved. The system assigns 0.0 points, with no partial credit allowed.

- **Over-selection is a Valid Variation (0.5):** A partial score of 0.5 is exclusively reserved for scenarios where the agent provides a valid superset of the required parameters (e.g., checking `['price', 'mileage']` when only `['price']` is required). This functionally resolves the query by providing the requested answer alongside extra safe information.

- **Type-Specific Matching Consistency:** To further eliminate ambiguity, exact subset/superset logic is strictly enforced for all structured fields (e.g., `Enum` types), while fuzzy matching is exclusively permitted for unstructured text fields (e.g., `query`) to accommodate valid semantic synonyms.

C.3.2 Reward Function.

The reward functions, which quantify alignment with ground truth and validity of tool invocations, serve as the basis for computing advantage values A_i. Three distinct reward designs are proposed, each combining technical accuracy and business relevance metrics in different configurations.

All reward functions take the form $r_\phi(a_{\text{pred}}, a_{\text{gold}})$, where a_{pred} is the LLM-generated tool sequence and a_{gold} is the ground truth. Three designs are implemented:

- R1: (Format-1) + PerfectMatch: Emphasizes structural validity and exact matches:

$$\text{R1}(a_{\text{pred}}, a_{\text{gold}}) = (\text{Format}-1)+\text{PerfectMatch}$$

- R2: Weighted Structural-Semantic Blend: Balances technical precision and business alignment through weighted components:

$$\begin{aligned}\text{R2}(a_{\text{pred}}, a_{\text{gold}}) = (&\text{Format} - 1)\\ &+ (\text{ToolParsed} - 1) + (\text{ToolName}\\ &+ \text{ParamName} + \text{ParamValue}\\ &+ \text{PerfectMatch}) \times 0.25\end{aligned}$$

- R3: Format + Balanced Business Metrics: Combines structural checks with equal weighting of exact and practical matches:

$$\begin{aligned}\text{R3}(a_{\text{pred}}, a_{\text{gold}}) = (&\text{Format} - 1)\\ &+ (0.5 \times \text{PerfectMatch})\\ &+ (0.5 \times \text{Resolution})\end{aligned}$$

A critical design principle across all reward configurations is the formulation of structural correctness strictly as a penalty, represented by the terms $(\text{Format} - 1)$ and $(\text{ToolParsed} - 1)$. This explicitly prevents the policy network from exploiting "easier-to-optimize" signals—such as merely generating well-formatted but semantically useless JSON—to accumulate high rewards. Because perfect formatting contributes exactly 0 to the final score, the agent can only achieve the high positive rewards necessary to trigger the Evolution mechanism by excelling in semantic and business-relevance metrics (e.g., PerfectMatch and Resolution). This attribution logic mathematically guarantees that all autonomous metadata updates are driven by genuine schema comprehension rather than superficial format hacking.

D Additional Case Studies

Case 2: Loan Calculation Tool ('calculate_loan'): The initial metadata for periods' merely lists valid terms (36/48/60) without enforcing constraints, leading the model to select an invalid value (31) to approximate the user's target monthly payment. The refined metadata explicitly mandates valid terms ("must be 36, 48, or 60"), guiding correct selection (36) and improving the parameter score from 0.0 to 1.0.

Case : Loan Calculation Tool ('calculate_loan')
Initial
metadata (Excerpt):
Parameters: periods(int): Loan term (36/48/60).
Action: calculate_loan("total_price": 8.5, "down_payment_ratio": 0.1, "periods": 31)
Scores: ParamValue = 0.0; Total Reward = 0.5
Refined
Metadata (Excerpt):
Parameters: periods(int): Loan term (36/48/60). Must be 36, 48, or 60; other values cause errors.
Action: calculate_loan("total_price": 8.5, "down_payment_ratio": 0.1, "periods": 36)
Scores: ParamValue = 1.0; Total Reward = 1.0

Table 6: A case study demonstrating parameter error resolution through metadata refinement in ToolDNA.

E Implementation of Validation Pipeline

To ensure the safety and stability of metadata updates, we implemented a rigorous validation pipeline within the Tool Manager. Based on the production deployment, the validation logic is formalized in **Algorithm 2**.

The pipeline processes raw responses from the LLM and enforces a three-stage verification before updating the global Tool Memory:

1. **Syntactic Parsing:** The system first extracts content within `<description>` tags. If the content cannot be parsed into structured tool blocks (e.g., missing tool names or malformed separators), the update is discarded immediately (Lines 2–5).

2. **Identity Verification:** For every proposed update, the system queries the `ToolMemory`

to ensure the tool exists. This prevents the hallucination of non-existent tools (Lines 7–9).

3. **Differential Validation:** The core function `CompareToolInfo` performs a field-level comparison between the `original_tool` and `new_tool_info`. It enforces strict immutable constraints (e.g., parameter types must remain consistent, required parameters cannot be deleted) to prevent breaking downstream applications (Line 11).

Only updates that pass all three checks are committed to the memory and logged for audit purposes.

E.1 Analysis of Rejected Updates

To demonstrate the robustness of this pipeline, we present representative examples of invalid metadata proposals intercepted by the Tool Manager during training. These safeguards prevented 23% of generated proposals that were syntactically correct but functionally invalid.

Case 1: Structural Rejection (Type Mismatch)

- **Tool:** `search_used_cars`
- **Proposal:** Changed the `car_age` parameter type from `List[int]` to `String` (e.g., "new").
- **Outcome: Rejected**. The backend API strictly requires integer lists. Accepting this would have caused runtime type errors.

Case 2: Semantic Rejection (Hallucination)

- **Tool:** `calculate_loan`
- **Proposal:** Added a non-existent parameter `discount_code` to the tool signature.
- **Outcome: Rejected**. The validation layer detected that this parameter is not supported by the tool's actual function signature in `ToolMemory`, preventing hallucinated arguments.

Algorithm 2 Metadata Validation and Update Pipeline

Require: LLM Response R, Tool Memory $\mathcal{M}$
Ensure: Boolean Status (Success/Fail), Logged Update

1: **function** PROCESSTOOLUPDATE(R)
2: **if** R is empty **or** <description> tag missing **then**
3: **return** FALSE ▷ Lexical Check Fail
4: **end if**
5: $blocks \leftarrow$ EXTRACTTOOLBLOCKS(R)
6: $valid\_updates \leftarrow []$
7: **for** each $block$ in $blocks$ **do**
8: $info_{new} \leftarrow$ EXTRACTTOOLINFO($block$)
9: $name \leftarrow info_{new}$.name
10: **if** $name$ not in $\mathcal{M}$ **then** ▷ Identity Check
11: **continue**
12: **end if**
13: $tool_{orig} \leftarrow \mathcal{M}$.get($name$)
14: $info_{orig} \leftarrow$ EXTRACTTOOLINFO($tool_{orig}$)
15: **if** COMPARETOOLINFO($info_{orig}, info_{new}$) **then** ▷ Semantic Check
16: $valid\_updates$.append($block$)
17: **else**
18: **LogWarning**("Validation failed for " + $name$)
19: **end if**
20: **end for**
21: **if** $valid\_updates$ is not empty **then**
22: $\mathcal{M}$.update($valid\_updates$)
23: LOGUPDATE(valid_updates)
24: **return** TRUE
25: **end if**
26: **return** FALSE
27: **end function**

F Decoupling Analysis: Policy vs. Metadata

To comprehensively verify that ToolDNA's performance stems from **synergistic co-evolution** rather than isolated improvements in metadata or policy, we conducted a rigorous 5-way decoupling analysis. We define two variables:

- **Model State:** *Untrained Base* (Qwen-2.5-7B) vs. *RL-Trained* (ToolRL trained on static $\mathcal{T}_{init}$) vs. *Co-Evolved* (ToolDNA).
- **Metadata State:** *Static* ($\mathcal{T}_{init}$) vs. *Evolved* ($\mathcal{T}_{final}$).

Table 7 presents the comparison across all meaningful combinations. Note that "Baseline + Evolved $\mathcal{T}_{final}$" settings are evaluated in an **inference-only** mode to test generalizability.

ID	Model Configuration	Metadata Used	PerfectMatch	Resolution
1	Qwen-2.5 (Untrained)	Static $\mathcal{T}_{init}$	0.1144	0.3475
2	Qwen-2.5 (Untrained)	Evolved $\mathcal{T}_{final}$	0.1512	0.3890
3	ToolRL Baseline	Static $\mathcal{T}_{init}$	0.4386	0.5487
4	ToolRL Baseline	Evolved $\mathcal{T}_{final}$	0.4550	0.5680
5	**ToolDNA (Co-evolved)**	**Evolved $\mathcal{T}_{final}$**	**0.5027**	**0.6295**

Table 7: Comprehensive decoupling analysis. Comparing Row 4 (Modular Combination) and Row 5 (Co-evolution) reveals the critical gain achieved by training the policy *alongside* the evolving metadata.

Analysis of Mechanisms. The results reveal three key insights:

- **Prompt Engineering Limit (Row 2 vs. 1):** Simply using better metadata ($\mathcal{T}_{final}$) on the base model yields a gain (+3.7%/+4.2%), confirming the intrinsic quality of our evolved schemas.
- **Mismatch in Modular Combination (Row 4 vs. 3):** When we apply the evolved metadata to the standard ToolRL policy (trained on $\mathcal{T}_{init}$), the performance improves marginally (+1.6%/+1.9%) but remains suboptimal. This indicates a "mismatch" effect: the policy trained on static metadata has overfitted to the old definitions and cannot fully exploit the new schema structure.
- **Necessity of Co-evolution (Row 5 vs. 4):** ToolDNA significantly outperforms the modular combination (+4.7%/+6.2%). This proves that the policy must be trained dynamically with the changing metadata to learn the optimal invocation patterns, validating our co-evolutionary framework over a simple two-stage (optimize metadata $\rightarrow$ train policy) approach.