PrefTool: Lightweight Preference-Aware Retrieval for Personalized Tool Selection

Anonymous ACL submission

Abstract

Personalized tool selection for LLM-based agents aims to identify tools that align with both user intent and individual preferences, particularly in redundant API ecosystems where multiple tools offer similar functionality but differ in non-functional attributes. Existing generative approaches typically depend on fine-tuning large language models to infer user preferences, leading to substantial computational overhead and deployment costs. Despite these expensive training requirements, they still struggle to reliably distinguish functionally equivalent tools and often fail to capture users' implicit non-functional preferences. To this end, we propose PrefTool, a lightweight yet effective preference-aware retrieval framework for personalized tool selection. PrefTool avoids costly generative preference modeling by employing a dual-encoder retrieval architecture that maps user context and tools into a shared representation space. Through structured tool representations, type-aware history modeling, and hard negative sampling, it efficiently captures users' non-functional preferences while maintaining strong selection accuracy. Experiments on a personalized tool selection benchmark demonstrate the effectiveness of PrefTool. Code is available at `https://anonymous.4open.science/r/PrefTool-01CE`.

1 Introduction

In recent years, the rapid advancement of large language models (LLMs) has been driving the evolution of artificial intelligence from passive text generators toward proactive tool-using agents (Schick et al., 2023; Patil et al., 2023). Equipped with external tool invocation capabilities, these agents are required to select and invoke appropriate tools (i.e., APIs) to fulfill user requests (Qu et al., 2024; Li, 2024). However, the space of available tools has grown rapidly with the continuous expansion of public API ecosystems, leading to significant redundancy in real-world tool interfaces (Liu et al., 2025; Blankenstein et al., 2025).

Specifically, for the same class of tasks (e.g., weather queries as shown in Figure 1), multiple APIs provided by different service providers often share highly similar functionalities. In such scenarios, tool selection transcends a purely functional decision problem. As shown in the "User Historical Interactions" (Figure 1), a user's preference may shift or consolidate over time toward a specific provider (e.g., WeatherAPI). While multiple tools may be functionally accurate, a preference-agnostic selection mechanism can still yield "non-preferred but correct results", failing to align with the user's implicit habits. Therefore, beyond functional correctness, tool selection should also consider users' preferences over non-functional attributes (NFAs), such as preferred service providers, interface styles, or habitual usage patterns.

To address this issue, personalized tool selection has gradually emerged as a research focus. Representative works such as PEToolBench (Xu et al., 2025) typically adopt an end-to-end generative paradigm, where the user query, interaction history, and candidate tools are jointly fed into an LLM, which directly generates the target tool. However, these approaches suffer from several limitations. First, they often require additional training of LLMs to perform tool selection, making the overall pipeline more complex and resource-intensive. Besides, they rely on autoregressive decoding to produce tool names token by token, without explicitly modeling the global candidate tool set. As a result, when confronted with a large number of functionally similar tools, they are prone to tool hallucination, i.e., generating invalid or non-existent tool identifiers (Schick et al., 2023; Patil et al., 2023; Qin et al., 2023). Finally, these methods struggle to effectively capture users' implicit non-functional preferences. User interaction histo-

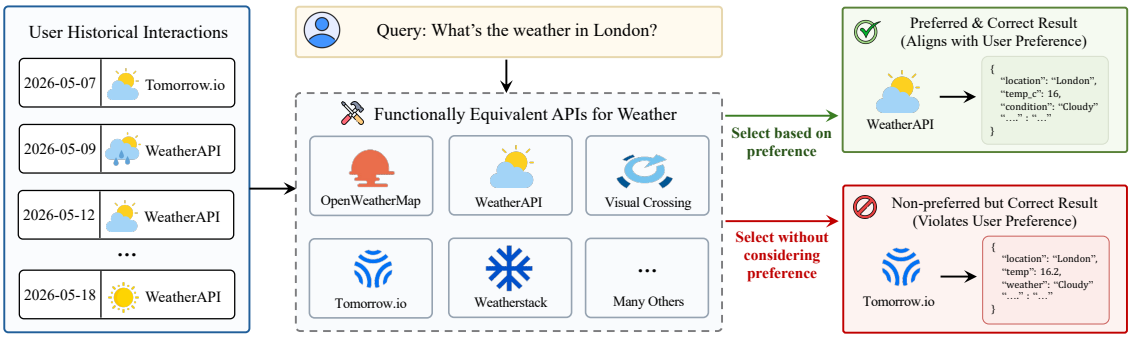

Figure 1: An illustration of preference-aware tool selection. Given a set of functionally equivalent APIs (e.g., weather services), historical interactions reveal the user's evolving preference for *WeatherAPI* over *Tomorrow.io*. Although both tools can return correct results, preference-aware selection better aligns with the user's habitual choices, whereas preference-agnostic selection may produce non-preferred outcomes.

ries are inherently heterogeneous, often containing a mixture of explicit positive feedback, negative feedback signals, and long-term evolving usage behaviors. Directly encoding such unstructured history into a single context can lead to information dilution, making it difficult for the model to reliably capture latent non-functional preferences, such as service provider bias and long-term usage habits, from complex and heterogeneous contexts (Yoon et al., 2026; Weng et al., 2024).

To this end, in this paper, we propose **PrefTool**, a *lightweight* ***Pref*** erence-aware retrieval framework for personalized ***Tool*** selection. Unlike end-to-end generative approaches, PrefTool reformulates tool selection as a retrieval problem. Given the current query and the user's historical preferences, it leverages a lightweight retrieval module to select the most appropriate tool from a candidate set that best matches the user's preferences. PrefTool consists of two stages: *representation learning* for constructing expressive tool and user representations, and *preference modeling* for aligning user preferences with candidate tools in a shared embedding space. *In the representation learning stage*, we design a schema-based fingerprint representation for tools, which encodes each tool into a compact form consisting of its name, provider metadata, and parameter structure. This design removes redundant natural language descriptions and reduces input complexity. Meanwhile, to handle the heterogeneity of user interaction histories, we propose a type-aware aggregation mechanism that models different behavioral signals separately and constructs a unified user preference representation. *In the preference modeling stage*, we learn to match user representations with candidate tools in a shared embedding space. We adopt a *lightweight* dual-encoder architecture to encode user context and tools independently, and train the model using a contrastive learning objective. To better capture non-functional attributes, we construct hard negatives that are functionally similar but originate from different providers. This encourages the model to move beyond functional matching and focus on distinguishing non-functional attributes such as provider identity, thereby enabling more accurate personalization. Our main contributions are summarized as follows:

- We propose PrefTool, a lightweight preference-aware retrieval framework for personalized tool selection. By reformulating tool selection as a retrieval problem, PrefTool enables more stable and efficient tool selection by incorporating user historical preferences.
- To enhance non-functional attribute modeling, we first introduce a schema-based tool fingerprint and a type-aware history aggregation method. Then, we further employ a contrastive learning strategy with provider-level hard negatives to focus on non-functional differences. This two-stage framework enables effective tool selection.
- On the personalized tool selection benchmark, PrefTool achieves an average tool selection accuracy of 91.23%, representing a 13.4% improvement over the best-performing generative baseline, which relies on resource-intensive LLM training. Notably, PrefTool is trained entirely on a single NVIDIA RTX 4090 (24GB) GPU, further demonstrating the efficiency and lightweight nature of the proposed retrieval-based framework.

2 Related Work

Tool Learning and Personalized Tool Selection. Tool learning enables large language models to invoke external tools effectively, extending their capabilities. Existing research can be broadly categorized into two technical paradigms. The first is an end-to-end generative tool calling, which treats tool selection as a sequence generation task, where the LLM directly outputs tool names and parameters. Representative works include Toolformer, Gorilla, ToolBench, GenTool, and ToolACE (Schick et al., 2023; Patil et al., 2023; Qin et al., 2023; He et al., 2025; Liu et al., 2024). While generative methods provide a unified modeling paradigm, they rely on autoregressive decoding over large candidate spaces, which is computationally expensive and requires repeated LLM inference during both training and deployment. Moreover, the lack of explicit global modeling over the tool set makes them prone to tool hallucination, especially when facing a large number of functionally similar tools. The second is retrieval-augmented tool selection, which decouples tool selection into retrieval and invocation stages, first filtering a small set of relevant tools before passing them to the LLM for invocation, thereby reducing context length pressure. Representative works include ToolRetriever, Tool-Dreamer, Tool-Embed and Tool-Rank, and Easy-Tool (Chen et al., 2026a; Sengupta et al., 2025; Lu et al., 2025b; Yuan et al., 2024). However, these methods primarily focus on functional semantic matching, overlooking users' personalized preferences over non-functional attributes such as providers and interface styles.

Recently, personalized tool selection has attracted increasing attention. PEToolLLM formalizes interaction history into a task framework for this area (Xu et al., 2025). ToolSpectrum introduces user profiles and environmental factors to extend personalization dimensions (Cheng et al., 2025). PTBench and PTool define two subtasks of tool preference and profile-dependent queries (Huang et al., 2025). TAPS proposes structured label-enhanced preference modeling (Taktasheva and Dalton, 2025), and ETAPP constructs evaluation benchmarks from the dual perspectives of personalization and proactivity (Hao et al., 2025). However, these methods follow the generative paradigm, with limited efficiency and accuracy in preference recognition. PrefTool models personalized tool selection as a contrastive retrieval problem, avoiding hallucination in the selection while accurately modeling non-functional preferences.

Contrastive Learning in Personalized Learning. Contrastive learning has been widely explored in personalized learning to model user preferences and behavioral patterns. Existing methods mainly focus on three directions: representation alignment, sequential preference modeling, and hard negative construction. The first line globally aligns user and item representations in a shared embedding space. RecDCL removes redundancy in positive representations through feature-level and batch-level contrastive learning (Zhang et al., 2024a), while JCLRec introduces dual contrastive views within a diffusion generation framework to improve sequential guidance signals (Zhu et al., 2025). Similar ideas have also been adopted in tool retrieval. For example, MassTool employs a dual-encoder architecture with contrastive regularization to jointly optimize tool detection and retrieval (Lin et al., 2025). The second line emphasizes fine-grained modeling of historical behavior sequences to capture dynamic user preferences. CLLMRec leverages LLMs to augment sequential views for modeling behavioral differences (Lu et al., 2025a), whereas HFCLRec introduces filtering with long- and short-term preference decoupling to alleviate interaction noise (Yang et al., 2026). Another direction lies in hard negative construction, improving representation discriminability via semantically similar yet behaviorally distinct samples. HNLMRec employs LLM-generated hard negatives to enhance preference distinction (Zhao et al., 2025a), while SCONE generates challenging negative samples through stochastic sampling strategies (Lee et al., 2024).

Inspired by these advances, PrefTool formulates personalized tool selection as a contrastive retrieval problem, combining structured tool representations, type-aware history modeling, and hard negative sampling to capture users' non-functional preferences in redundant tool ecosystems.

3 Framework of PrefTool

As illustrated in Figure 2, PrefTool consists of two stages: (1) a representation learning stage and (2) a preference modeling stage. In the first stage, we construct schema-based fingerprints for tools and transform heterogeneous interaction histories into unified structured user anchors, enabling both tools and user preferences to be represented within a shared schema-level feature space. In the sec-

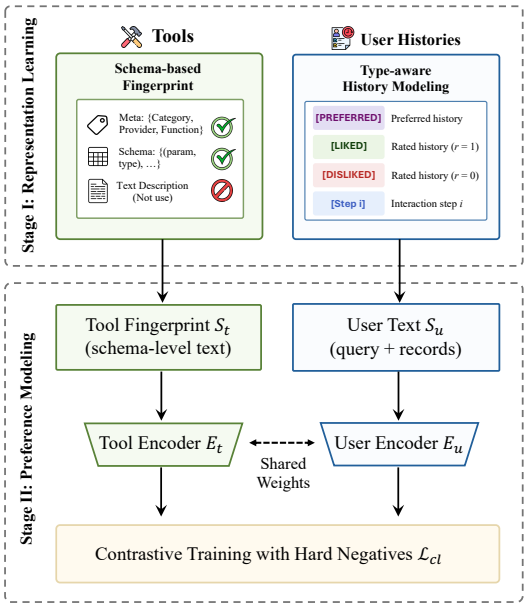

Figure 2: Overview of the proposed PrefTool framework. Stage I performs representation learning by constructing schema-based fingerprints for tools and type-aware user anchors from heterogeneous interaction histories, enabling both tools and user preferences to be represented within a shared schema-level textual space. Stage II employs a shared-parameter dual-encoder with contrastive learning to align user preferences and candidate tools in a dense semantic space, where provider-level hard negatives encourage sensitivity to non-functional attributes.

ond stage, we employ a dual-encoder architecture with contrastive learning to align user preferences and candidate tools in the dense semantic space, where hard negatives are introduced to enhance the model's sensitivity to non-functional attribute differences. During inference, PrefTool performs personalized tool selection by ranking candidate tools according to dense vector similarity.

3.1 Problem Formulation

3.1.1 Task Definition

Personalized tool selection aims to identify the most appropriate tool for a user u given both the current query q_u and the user's historical interactions $\mathcal{H}_u$. Among them, $\mathcal{H}_u = \{h_u^1, h_u^2, \ldots, h_u^M\}$, where M is the length of the history. Each historical interaction $h_u^i = (q_u^i, c_u^i)$ consists of a past query q_u^i and the corresponding tool invocation $c_u^i = (t_u^i, p_u^i)$, where t_u^i and p_u^i represent the selected tool and its parameters, respectively.

Given a candidate tool set $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$, the goal is to select a tool $t^* \in \mathcal{T}$ that satisfies the functional requirement of the current query while aligning with the user's preferences reflected in their interaction history. We formalize this as a retrieval and ranking problem:

$$t^* = \arg\max_{t \in \mathcal{T}} \; s(q_u, \mathcal{H}_u, t), \tag{1}$$

where $s(\cdot)$ is a personalized scoring function that evaluates the relevance between a candidate tool and the user context.

3.1.2 Interaction History Types

In this paper, we consider three distinct types of interaction histories $\mathcal{H}_u$, each capturing a different form of user preference expression, following the settings of PEToolBench (Xu et al., 2025). Specifically, we define Preferred-only History ($\mathcal{H}^p$), Rating-integrated History ($\mathcal{H}^r$), and Chronological History ($\mathcal{H}^c$).

Preferred-only History ($\mathcal{H}^p$) contains only tool calls that align with the user's preferences, without providing explicit negative feedback. Rating-integrated History ($\mathcal{H}^r$) augments each interaction with a binary rating $r^i \in \{0, 1\}$, explicitly indicating whether the user "liked" or "disliked" the tool invocation. Chronological History ($\mathcal{H}^c$) organizes interactions according to temporal order, where recent interactions more accurately reflect the user's current preferences, while earlier interactions may correspond to outdated ones. For example, in web search scenarios, a user may initially favor the Chinese-language ecosystem of Baidu but later shift toward Google's academic ecosystem.

These heterogeneous history types impose different requirements on preference modeling. Specifically, $\mathcal{H}^p$ requires the model to capture long-term consistency of tool ecosystem attributes. $\mathcal{H}^r$ requires explicit discrimination between positive and negative feedback. $\mathcal{H}^c$ demands sensitivity to temporal preference drift. This observation motivates our type-aware history modeling design (§3.2.2).

3.2 Stage I: Representation Learning Stage

The goal of the representation learning stage is to construct structured vector representations for tools and user histories separately. We first define the schema-based fingerprint for tools (§3.2.1), and then design a type-aware history aggregation method (§3.2.2). Details are presented below.

3.2.1 Schema-based Fingerprint for Tools

In personalized tool selection, user preferences are often reflected through *non-functional attributes*,

such as provider ecosystems, interface conventions, parameter styles, and invocation habits, rather than functional capability alone. However, existing tool representations mainly rely on verbose natural language descriptions, which are dominated by functional semantics (e.g., "retrieve weather information"). As a result, tools with similar functionality but different ecosystems are mapped to highly similar representations, making it difficult for the model to distinguish user-specific preferences.

To address this issue, we propose a *schema-based fingerprint representation* that abstracts each tool into a compact structured form. Instead of encoding verbose natural language descriptions, we preserve only schema-level information that is closely associated with non-functional characteristics. Meanwhile, we retain the function name as a concise indicator of the tool's core functionality. In this way, the proposed representation mitigates the dominance of redundant textual descriptions while preserving essential functional semantics.

Specifically, for each tool t, the schema-based fingerprint consists of two components. First, we extract its meta-information from the hierarchical tool identifier, including the category C_t, provider P_t, and function name F_t:

$$\text{Meta}(t) = \{C_t,\ P_t,\ F_t\}. \tag{2}$$

Next, we construct a parameter signature by extracting the names and types of all input parameters associated with tool t:

$$\text{Schema}(t) = \{(p_1, \tau_1), \ldots, (p_K, \tau_K)\}, \tag{3}$$

where p_k and τ_k denote the name and type of the k-th parameter of tool t, respectively. For tools without parameters, we use a special token `[NoParam]`.

The final schema-based fingerprint of tool t is:

$$S_t = \text{Concat}(\text{Meta}(t), \text{Schema}(t)). \tag{4}$$

Several constructed fingerprints of tools are provided in Appendix C.

Remark 1. Compared with the raw tool textual description, the proposed one suppresses redundant functional semantics while preserving ecosystem-level structural characteristics. Consequently, tools sharing similar providers or interface conventions are more likely to form coherent clusters in the representation space, enabling the model to better capture personalized non-functional preferences.

3.2.2 Type-Aware User History modeling

The user interaction history $\mathcal{H}_u = \{h_u^1, \ldots, h_u^M\}$ contains rich signals about the user's implicit preferences. However, as discussed in §3.1.2, different history types encode preference information in fundamentally different ways. Therefore, directly concatenating heterogeneous histories into a single sequence would mix incompatible preference semantics and weaken discriminative signals. To address this issue, we propose a type-aware user history modeling strategy that converts heterogeneous historical interaction h_u^i into a unified yet semantically distinguishable structured representation $\texttt{record}_u^i$.

Specifically, each history record $\texttt{record}_u^i$ consists of two components: (1) a schema-aware interaction summary, and (2) a type-aware preference prefix. For the schema-aware interaction summary, we extract the category, provider, function name, and parameter key list using the same schema extraction process as the tool fingerprint representation. In addition, we retain a short intent summary by truncating the original user instruction to its first 50 characters. This design ensures that historical interactions and candidate tools are represented under a shared structural feature space.

To further preserve the semantics of different history types, we introduce a type-aware prefix function $\phi(\cdot)$ that injects explicit preference indicators into each history record:

$$\phi(\text{type}, i, r^i) = \begin{cases} \text{[PREFERRED]}, & \text{if type} = \mathcal{H}^p. \\ \text{[LIKED]}, & \text{if type} = \mathcal{H}^r \text{ and } r^i = 1, \\ \text{[DISLIKED]}, & \text{if type} = \mathcal{H}^r \text{ and } r^i = 0, \\ \text{[Step i]}, & \text{if type} = \mathcal{H}^c, \end{cases} \tag{5}$$

Finally, we prepend the current query q_u and concatenate all structured history records to construct the user preference anchor text:

$$S_u = \text{Concat}(q_u, \texttt{record}_u^1, \ldots, \texttt{record}_u^M). \tag{6}$$

Remark 2. Through the proposed type-aware modeling strategy, heterogeneous preference signals can be represented in a unified textual format while preserving their underlying semantics. Moreover, since historical interactions and candidate tools share the same schema-level representation granularity, the model can directly align user preference patterns with candidate tool characteristics, particularly non-functional attributes such as provider ecosystems and interface conventions.

3.3 Stage II: Preference Modeling Stage

After constructing representations for users and tools, PrefTool employs a dual-encoder framework to align user preferences with candidate tools in a shared semantic space. The model is optimized using contrastive learning with hard negatives, encouraging sensitivity to non-functional attributes.

3.3.1 Dual-Encoder Architecture

As illustrated in Figure 2, PrefTool employs a dual-encoder comprising a user encoder E_u and a tool encoder E_t that share the same parameters. Given the user anchor text S_u and the schema-based fingerprint S_t of a candidate tool, the two encoders map them into dense representations:

$$\mathbf{v}_u = E_u(S_u), \mathbf{v}_{t_i} = E_t(S_{t_i}), \quad t_i \in \mathcal{T}. \tag{7}$$

The preference matching score between the user and tool is computed using cosine similarity:

$$s(q_u, \mathcal{H}_u, t_i) = \cos(\mathbf{v}_u, \mathbf{v}_{t_i}) = \frac{\mathbf{v}_u^\top \mathbf{v}_{t_i}}{\|\mathbf{v}_u\| \|\mathbf{v}_{t_i}\|}. \tag{8}$$

During inference, all candidate tools are ranked according to their similarity scores, and the top-ranked tool is selected as the prediction.

3.3.2 Contrastive Training with Hard Negatives

We train PrefTool using preference pairs from PEToolBench (Chen et al., 2026b; Zhao et al., 2025b). Each training instance contains a user anchor S_u, a preferred tool t^+, and a rejected tool t^-. Importantly, t^+ and t^- are functionally equivalent but belong to different providers or ecosystems. For example, the same `products/list` functionality may belong to Wayfair versus Kohls, and can only be distinguished by non-functional attributes such as provider and parameter style. Thus, t^- naturally serves as a hard negative, compelling the model to attend to non-functional attributes.

For each training triplet (S_u, S_{t^+}, S_{t^-}), the encoders generate the corresponding representations $\mathbf{v}_u^{(i)}$, $\mathbf{v}_{t^+}^{(i)}$, and $\mathbf{v}_{t^-}^{(i)}$. For the i-th sample, we define the negative set as:

$$\mathcal{N}(i) = \{\mathbf{v}_{t^+}^{(j)} \mid j \neq i\} \cup \{\mathbf{v}_{t^-}^{(i)}\}, \tag{9}$$

which consists of in-batch negatives together with the explicit hard negative from the current sample.

The training objective is formulated as an InfoNCE loss (Oord et al., 2018; Jin et al., 2025):

$$\mathcal{L}_{cl} = -\frac{1}{B} \sum_{i=1}^{B} \log \frac{\exp(\cos(\mathbf{v}_u^{(i)}, \mathbf{v}_{t^+}^{(i)})/\tau)}{\sum_{\mathbf{v} \in \mathcal{N}(i)} \exp(\cos(\mathbf{v}_u^{(i)}, \mathbf{v})/\tau)}. \tag{10}$$

By jointly leveraging in-batch negatives and provider-level hard negatives, the proposed objective explicitly pushes apart functionally similar but ecosystem-different tools, thereby improving the model's ability to capture personalized non-functional preferences.

4 Experiments

4.1 Experimental Setup

Datasets and Scenarios. We evaluate on the public PEToolBench benchmark (Xu et al., 2025). It contains 12,000 instances and 7,454 real-world APIs. The dataset is evenly divided into three personalization scenarios: preferred-only history (P), rating-integrated history (R), and chronological history (C), where each scenario contains 3,000 training and 1,000 testing instances. The evaluation metric is tool selection accuracy (Tool Acc), with the candidate tool set size fixed at 10.

Baselines. We compare against a diverse set of representative models. Generative baselines include the closed-source GPT-4o and GPT-4o-mini (OpenAI et al., 2024), open-source LLaMA-3-8B (Dubey et al., 2024), Qwen2.5-7B (Qwen et al., 2024), Mistral-7B (Jiang et al., 2023), and Vicuna-7B (Chiang et al., 2023), as well as the current state-of-the-art PEToolLLaMA (fine-tuned with SFT and DPO) (Xu et al., 2025). Retrieval baselines include sparse retrieval BM25 (Robertson and Zaragoza, 2009), zero-shot dense retriever GTE (Zhang et al., 2024b), and a Tool Retrieval Bridge (TRB)-style pipeline (Chen et al., 2026a) adapted with GPT-4o-mini for query rewriting. Implementation details are summarized in Appendix A.

4.2 Overall Performance

As shown in Table 1, we compare PrefTool with generative baselines, retrieval baselines, and the current state-of-the-art PEToolLLaMA across the three personalization scenarios of PEToolBench (Xu et al., 2025). Overall, PrefTool achieves the best results in all three scenarios, with an average accuracy of 91.23%, surpassing the previous best method PEToolLLaMA (77.83%) by 13.4%, validating the overall effectiveness of the retrieval

Table 1: Tool selection accuracy comparison on PEToolBench across three personalization scenarios (i.e., preferred-only, rating-integrated, and chronological history). Tool candidate set size is 10. The bolded denotes the best result.

Methods	PREFERRED-ONLY (P)	RATING-INTEGRATED (R)	CHRONOLOGICAL (C)	AVG.
		Generation-based Method		
Vicuna-7B	25.50	10.80	12.70	16.33
Mistral-7B	30.30	15.40	14.10	19.93
Qwen2.5-7B	40.40	24.80	24.80	30.00
LLaMA-3-8B	48.10	26.90	26.60	33.87
GPT-4o-mini	51.80	38.40	31.20	40.47
GPT-4o	53.70	45.70	33.60	44.33
PEToolLLaMA	74.30	78.40	80.80	77.83
		Retrieval-based Method		
BM25	2.00	2.30	2.30	2.20
GTE (zero-shot)	24.50	13.30	8.30	15.37
TRB	23.40	19.40	13.60	18.80
PrefTool (Ours)	**91.30**	**91.20**	**91.20**	**91.23**

Table 2: Error type breakdown: full descriptions vs. schema-based fingerprint (%) across different scenarios (P, R, and C). TFM denotes Tool Functionality Mismatch. TPM denotes Tool Preference Mismatch.

Setting	Configuration	TFM (↓)	TPM (↓)
P	w/ descriptions	10.00	3.50
	w/o descriptions (Ours)	**5.60**	**3.10**
R	w/ descriptions	10.30	3.90
	w/o descriptions (Ours)	**5.20**	**3.60**
C	w/ descriptions	10.20	**3.60**
	w/o descriptions (Ours)	**5.10**	3.70

paradigm for personalized tool selection. Specifically, we make the following observations:

First, generative baselines such as GPT-4o and LLaMA-3-8B achieve below 45% average accuracy, while BM25, GTE, and TRB obtain only 2.20%, 15.37%, and 18.80% average accuracy. This shows that neither autoregressive generation, generic retrieval, nor pipelined query rewriting can effectively capture users' implicit non-functional preferences. In contrast, PrefTool formulates personalized tool selection as a preference-aware retrieval task with a dual-encoder architecture, improving sensitivity to non-functional attributes.

Second, PrefTool consistently outperforms the strongest baseline PEToolLLaMA across all settings. In the challenging chronological (C) and rating-integrated (R) scenarios, PrefTool achieves 91.20% accuracy in both cases, surpassing PEToolLLaMA by 10.40% and 12.80%, respectively. The gains mainly come from type-aware prefixes that encode polarity and temporal signals, together with the dual-encoder design that alleviates noise from heterogeneous histories. Moreover, unlike PEToolLLaMA's costly tuning of large generative models, PrefTool only trains lightweight encoders.

Finally, contrastive training is essential for effective preference-aware retrieval. While GTE achieves only 15.37% average accuracy, PrefTool improves to 91.23% after contrastive training on preference data, demonstrating the importance of preference-oriented representation learning for distinguishing fine-grained non-functional attributes.

4.3 Ablation Study

To evaluate the contribution of each component, we conduct ablation experiments:

(*i*) As shown in Table 2 and Figure 3, we replace the schema-based fingerprint with full natural-language tool descriptions. The average accuracy drops from 91.23% to 86.17%, and functionality mismatch (TFM) rises from 5.30% to 10.17%, while tool preference mismatch (TPM) remains consistently around 3%. The stable TPM confirms that user preferences are reliably captured regardless of description style, whereas the increased TFM indicates that redundant natural language introduces functional confusion.

(*ii*) In Figure 3, we remove the type-aware prefixes from the history encoding, treating the three history types as homogeneous sequences. Compared with PrefTool (full), this variant suffers an average accuracy drop of 6.60%, with severe declines of 8.30% and 5.90% in the R and C scenarios, demonstrating that explicit polarity signals are crucial for preference modeling.

(*iii*) In Figure 3, removing hard negatives from the DPO data and relying only on in-batch negatives sharply decreases the average accuracy to about 53.00%. This shows that random negatives are insufficient for distinguishing highly similar tools, while hard negatives are essential for learning fine-grained non-functional preferences.

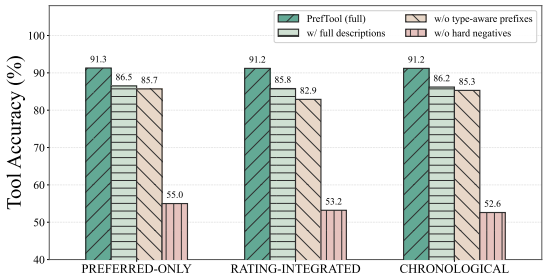

Figure 3: Ablation study results of PrefTool variants across three personalization scenarios.

Table 3: Parameter generation accuracy in the full pipeline. P, R, and C denote the three personalization scenarios. PrefTool + Qwen2.5-7B uses PrefTool-retrieved tools, while Oracle Tool + Qwen2.5-7B directly provides ground-truth tools as an upper bound.

Method	P	R	C	Avg.
Qwen2.5-7B (end-to-end)	63.80	66.50	70.20	66.83
PEToolLLaMA (end-to-end)	87.90	89.70	91.30	89.60
PrefTool + Qwen2.5-7B	74.50	75.10	75.00	74.87
Oracle Tool + Qwen2.5-7B	75.20	75.20	75.20	75.20

4.4 Plug-and-Play Integration with LLMs

We evaluate PrefTool as a plug-and-play retriever in collaboration with frozen LLMs. We concatenate the top-1 tool retrieved by PrefTool with the user instruction and feed them into a frozen Qwen2.5-7B for zero-shot parameter generation. As shown in Table 3, PrefTool + Qwen2.5-7B achieves 74.87% average parameter accuracy, demonstrating the effectiveness of the decoupled framework.

Compared with directly using Qwen2.5-7B for end-to-end tool calling (66.83%), PrefTool yields an 8% improvement. This suggests many generative errors originate from incorrect tool selection rather than parameter generation. By retrieving correct documentation, PrefTool enables frozen LLMs to generate parameters effectively without gradient updates.

We further introduce an Oracle setting providing the ground-truth tool. Oracle Tool + Qwen2.5-7B achieves 75.20%, only 0.33% higher than PrefTool + Qwen2.5-7B, indicating PrefTool incurs almost no retrieval loss on this benchmark. This also suggests parameter generation inherently ceilings around 75%.

Compared with the fine-tuned PEToolLLaMA (89.60%), the remaining gap mainly comes from the limited zero-shot parameter generation capability of frozen Qwen2.5-7B, since PrefTool already approaches the oracle retrieval upper bound. Moreover, PrefTool only requires training lightweight dual encoders while keeping the generative model frozen, making it more flexible and cost-efficient in dynamic deployment scenarios.

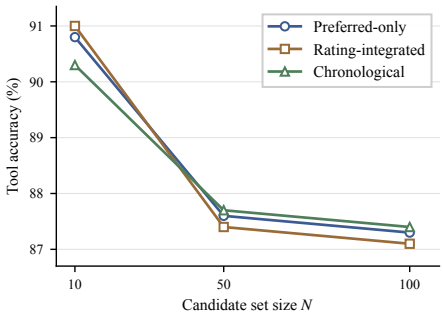

Figure 4: Robustness of PrefTool to candidate set size. Tool selection accuracy is reported across three scenarios as the candidate pool expands from 10 to 100.

4.5 Robustness to Candidate Set Size

To evaluate PrefTool under more challenging conditions, we extend the candidate tool set from the standard 10 to 50 and 100 by retrieving semantically similar tools from the PEToolBench tool pool using a lightweight sentence encoder; implementation details are provided in Appendix B. As shown in Figure 4, PrefTool maintains consistent performance, with accuracy dropping by only 3.4% when expanding from 10 to 100 candidates. This stability confirms the reliability of PrefTool under substantially expanded candidate pools.

5 Conclusion

We proposed PrefTool, a lightweight preference-aware retrieval framework for personalized tool selection. Unlike generative approaches, PrefTool reformulated tool selection as a retrieval task and explicitly modeled users' non-functional preferences over functionally equivalent tools. It introduced a two-stage framework consisting of representation learning and preference modeling. Specifically, we designed a schema-based tool fingerprint to capture compact tool characteristics, and a type-aware aggregation mechanism to model heterogeneous user histories. Besides, we used a lightweight dual-encoder with contrastive learning to distinguish non-functional attributes among similar tools. Experiments demonstrate that PrefTool outperforms strong generative baselines while requiring significantly lower computational and training overhead.

Limitations

While PrefTool demonstrates significant improvements in personalized tool selection, it has a few limitations. First, our schema-based fingerprint deliberately discards natural language descriptions to emphasize non-functional attributes. This design relies heavily on well-structured API metadata and may lead to suboptimal functional matching in open-domain ecosystems where API names or parameters are ambiguous. Second, our type-aware history aggregation employs a relatively rigid rule-based truncation strategy (e.g., limiting instructions to 50 characters). It may struggle to capture deep, multi-turn contexts where user intent evolves over lengthy interactions. Finally, our contrastive modeling inherently relies on interaction history, lacking a dedicated mechanism for the "cold-start" problem, such as making recommendations for new users with zero history or retrieving newly published long-tail APIs.

References

Thierry Blankenstein, Jialin Yu, Zixuan Li, Vassilis Plachouras, Sunando Sengupta, Philip Torr, Yarin Gal, Alasdair Paren, and Adel Bibi. 2025. Biasbusters: Uncovering and mitigating tool selection bias in large language models. In *arXiv.org*.

Kunfeng Chen, Luyao Zhuang, Fei Liao, Juhua Liu, Jian Wang, and Bo Du. 2026a. Tool retrieval bridge: Aligning vague instructions with retriever preferences via bridge model. *Neural Networks*, 202:109037.

Ziyang Chen, Zhenxuan Huang, Yile Wang, Weiqin Wang, Lu Yin, and Hui Huang. 2026b. Sempa: Improving sentence embeddings of large language models through semantic preference alignment. *arXiv.org*, arXiv:2601.05075.

Zihao Cheng, Hongru Wang, Zeming Liu, Yuhang Guo, Yuanfang Guo, Yunhong Wang, and Haifeng Wang. 2025. ToolSpectrum: Towards personalized tool utilization for large language models. In *Annual Meeting of the Association for Computational Linguistics*, pages 20679–20699. Association for Computational Linguistics.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, A. Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Yupu Hao, Pengfei Cao, Zhuoran Jin, Huanxuan Liao, Yubo Chen, Kang Liu, and Jun Zhao. 2025. Evaluating personalized tool-augmented LLMs from the perspectives of personalization and proactivity. In *Annual Meeting of the Association for Computational Linguistics*, pages 21897–21935. Association for Computational Linguistics.

Jie He, Jennifer Neville, Mengting Wan, Longqi Yang, Hui Liu, Xiaofeng Xu, Xia Song, Jeff Z. Pan, and Pei Zhou. 2025. GenTool: Enhancing tool generalization in language models through zero-to-one and weak-to-strong simulation. In *Annual Meeting of the Association for Computational Linguistics*, pages 1097–1122. Association for Computational Linguistics.

Xu Huang, Yuefeng Huang, Weiwen Liu, Xingshan Zeng, Yasheng Wang, Ruiming Tang, Hong Xie, and Defu Lian. 2025. Advancing and benchmarking personalized tool invocation for llms. *arXiv.org*, arXiv:2505.04072.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv.org*, arXiv:2310.06825.

Bowen Jin, Jinsung Yoon, Zhen Qin, Ziqi Wang, Wei Xiong, Yu Meng, Jiawei Han, and Sercan O. Arik. 2025. Llm alignment as retriever optimization: An information retrieval perspective. *International Conference on Machine Learning*, arXiv:2502.03699.

Chaejeong Lee, Jeongwhan Choi, Hyowon Wi, Sung-Bae Cho, and Noseong Park. 2024. Scone: A novel stochastic sampling to generate contrastive views and hard negative samples for recommendation. In *Web Search and Data Mining*, WSDM '25, page 419–428, New York, NY, USA. Association for Computing Machinery.

Xinzhe Li. 2024. A review of prominent paradigms for llm-based agents: Tool use, planning (including rag), and feedback learning. In *International Conference on Computational Linguistics*, pages 9760–9779.

Jianghao Lin, Xinyuan Wang, Xinyi Dai, Menghui Zhu, Bo Chen, Ruiming Tang, Yong Yu, and Weinan Zhang. 2025. Masstool: A multi-task search-based tool retrieval framework for large language models. *arXiv.org*.

Marianne Menglin Liu, Daniel Garcia, Fjona Parllaku, Vikas Upadhyay, Syed Fahad Allam Shah, and Dan Roth. 2025. Toolscope: Enhancing llm agent tool use through tool merging and context-aware filtering. *arXiv.org*, arXiv:2510.20036.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024.

ToolACE: Winning the points of LLM function calling. In *International Conference on Learning Representations*.

Fan Lu, Xiaolong Xu, Haolong Xiang, Lianyong Qi, Xiaokang Zhou, Fei Dai, and Wanchun Dou. 2025a. Cllmrec: Contrastive learning with llms-based view augmentation for sequential recommendation. In *International Joint Conference on Artificial Intelligence*, pages 3153–3161. International Joint Conferences on Artificial Intelligence Organization.

Xuan Lu, Haohang Huang, Rui Meng, Yaohui Jin, Wenjun Zeng, and Xiaoyu Shen. 2025b. Tools are underdocumented: Simple document expansion boosts tool retrieval. In *arXiv.org*.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv.org*.

Josh Achiam OpenAI, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2024. Gpt-4 technical report, 2024. *URL https://arxiv.org/abs/2303.08774*, 2(6):2.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: large language model connected with massive apis. In *Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, X. Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343.

A Yang Qwen, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengpeng Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *Young People's Involvement in Sport*, pages 241–252.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, R. Raileanu, M. Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Neural Information Processing Systems*, NIPS '23, pages 68539–68551, Red Hook, NY, USA. Neural Information Processing Systems Foundation, Inc. (NeurIPS).

Saptarshi Sengupta, Zhengyu Zhou, Jun Araki, Xingbo Wang, Bingqing Wang, Suhang Wang, and Zhe Feng. 2025. ToolDreamer: Instilling LLM reasoning into tool retrievers. In *Conference of the European Chapter of the Association for Computational Linguistics*.

Ekaterina Taktasheva and Jeff Dalton. 2025. TAPS: Tool-augmented personalisation via structured tagging. In *Conference on Empirical Methods in Natural Language Processing*, pages 23530–23555. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. In *Neural Information Processing Systems*, volume 33, pages 5776–5788. Curran Associates, Inc.

Muyan Weng, Yunjia Xi, Weiwen Liu, Bo Chen, Jianghao Lin, Ruiming Tang, Weinan Zhang, and Yong Yu. 2024. Beyond positive history: Re-ranking with list-level hybrid feedback. *arXiv.org*, arXiv:2410.20778.

Qiancheng Xu, Yongqi Li, Heming Xia, Fan Liu, Min Yang, and Wenjie Li. 2025. PEToolLLM: Towards personalized tool learning in large language models. In *Annual Meeting of the Association for Computational Linguistics*, pages 21488–21503. Association for Computational Linguistics.

Xingyao Yang, Guangchao Li, Shuai Ma, Zhilin Li, Hongtao Shen, and Zheng Qi. 2026. *Sequential Recommendation with Hierarchical Filtering and Long- and Short-Term Preference Contrastive Learning*, pages 28–46. Springer Nature Switzerland.

Yejin Yoon, Minseo Kim, and Taeuk Kim. 2026. Latent preference modeling for cross-session personalized tool calling. *Preprint*, arXiv:2604.17886.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. EASYTOOL: Enhancing LLM-based agents with concise tool instruction. In *North American Chapter of the Association for Computational Linguistics*.

Dan Zhang, Yangliao Geng, Wenwen Gong, Zhongang Qi, Zhiyu Chen, Xing Tang, Ying Shan, Yuxiao Dong, and Jie Tang. 2024a. Recdcl: Dual contrastive learning for recommendation. In *The Web Conference*, pages 3655–3666. ACM.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, and 1 others. 2024b. mGTE: Generalized long-context text representation and reranking models for multilingual text retrieval. In *Conference on Empirical Methods in Natural Language Processing*, pages 1393–1412. Association for Computational Linguistics.

Chu Zhao, Enneng Yang, Yuting Liu, Jianzhe Zhao, and Guibing Guo. 2025a. Hard negative sampling via large language models for recommendation. *Preprint*, arXiv:2504.04726.

Pengfei Zhao, Rongbo Luan, Wei Zhang, Peng Wu, and Sifeng He. 2025b. Guiding cross-modal representations with mllm priors via preference alignment. *arXiv.org*, arXiv:2506.06970.

Kai Zhu, Jing Li, Yue He, Mingfeng Wang, Jiaheng Yu, Jun Chang, and Jun Wan. 2025. Jclrec: Joint diffusion model and dual contrastive learning for sequential recommendation. *Knowledge-Based Systems*, page 114888.

A Implementation Details

PrefTool initializes the dual encoders, i.e., the user encoder E_u and tool encoder E_t, with gte-large-en-v1.5 and is trained for 3 epochs using MultipleNegativesRankingLoss (MNRL) (Zhang et al., 2024b). We adopt the AdamW optimizer with a learning rate of 2×10^{-5} and a batch size of 5. The maximum sequence length is set to 1024 tokens during training and extended to 2048 during inference to accommodate long interaction histories. The temperature parameter in MNRL is fixed at $\tau = 0.05$ (scale=20). All experiments are conducted on a single NVIDIA 4090 (24GB) GPU with a fixed random seed of 42 for reproducibility.

For the batch size ablation in Figure 5, we vary the batch size from 2 to 6 while keeping all other training configurations unchanged. Each mini-batch consistently contains one hard negative sampled from the DPO preference data, whereas the remaining negatives are naturally formed as in-batch negatives from other samples within the same batch. This design preserves stable contrastive learning signals under limited batch sizes and strengthens discrimination among functionally similar tools.

Dataset License. PEToolBench is released under the Apache License 2.0. We use the dataset solely for academic research and fully comply with its licensing terms. The permissive license also facilitates reproducibility of the experimental results.

B Candidate Pool Expansion

To evaluate PrefTool under larger and more challenging candidate spaces, we construct expanded candidate pools beyond the default setting. Specifically, we first collect all unique tools appearing in the original test set and encode them using a lightweight SentenceTransformer (all-MiniLM-L6-v2) (Wang et al., 2020) based on tool names and descriptions. For each test query, we encode the user instruction with the same model and compute cosine similarity against all tools in the collected pool. We then iteratively append the most semantically relevant tools that are absent from the original candidate set until reaching target candidate sizes of 50 and 100.

Unlike random candidate augmentation, our strategy introduces functionally relevant yet semantically competitive tools, substantially increasing the difficulty of tool selection while better reflecting realistic tool ecosystems, where numerous APIs often exhibit overlapping functionality.

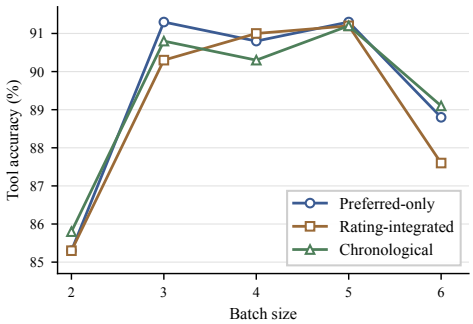

Figure 5: Effect of batch size on tool accuracy. One hard negative is fixed per batch, with remaining negatives drawn as in-batch negatives.

C Schema-based Tool Fingerprint

To demonstrate how the schema-based fingerprint abstracts tool metadata, Table 4 presents two representative tools from PEToolBench. Although both tools expose the same functionality (`products/detail`), they originate from different providers (Wayfair and Kohls). The original tool specifications contain verbose functional descriptions together with parameter details, whereas our fingerprint representation preserves only structured metadata, including the category, provider, function name, and parameter signature.

This representation intentionally removes redundant natural language descriptions while preserving non-functional signals that are critical for personalized tool selection. As a result, the dual encoder can better distinguish tools based on provider-specific characteristics and schema-level patterns, rather than relying on semantically overlapping descriptions of equivalent functionality. All tools in our experiments are represented using this schema-based fingerprint format.

D Prompt Templates

We detail the prompt templates used in our experiments. Figure 6 illustrates the zero-shot parameter generation prompt for our plug-and-play evaluation. It provides a frozen LLM (e.g., Qwen2.5-7B) with the user instruction and the schema of the tool retrieved by PrefTool, guiding the model to extract argument values into a strict JSON format. This setup ensures that the parameter generation performance strictly reflects the retrieval quality without requiring gradient updates.

Furthermore, Figure 7 presents the query rewriting prompt used for the TRB baseline. To con-

Table 4: Comparison of original tool descriptions and our schema-based fingerprints. Both tools share the same function but belong to different providers, illustrating how the fingerprint compresses the verbose documentation and exposes non-functional attributes such as provider identity and parameter signature for preference discrimination.

Representation	Content
Original Tool Description	**Tool name:** `<eCommerce>.<Wayfair>.<products/detail>` **Description:** Get detail information of specific product. **Required parameters:** Name: `sku` Type: `STRING` Description: The value of sku fields returned in `.../products/list` or `.../products/search` endpoint. Default: `W003247135` **Optional parameters:** Name: `wfProductOptions` Type: `STRING` Description: The value of id fields under relevantAttributes JSON object returned right in this endpoint, pass this parameter multiple times for multiple options. Ex: `...&wfProductOptions=1234567890&wfProductOptions=special_offers&...` Default: (empty)
Schema-based Fingerprint for Tools (Ours)	`Category: eCommerce | Provider: Wayfair | Function: products/detail | Params: [sku(STRING), wfProductOptions(STRING)]`
Original Tool Description	**Tool name:** `<eCommerce>.<Kohls>.<products/detail>` **Description:** Get detail information of specific product. **Required parameters:** Name: `webID` Type: `STRING` Description: The value of webID returned in `.../products/list` endpoint. Default: `4201989` **Optional parameters:** None.
Schema-based Fingerprint for Tools (Ours)	`Category: eCommerce | Provider: Kohls | Function: products/detail | Params: [webID(STRING)]`

struct a fair and formidable baseline, we employ GPT-4o-mini as a zero-shot bridge model to rewrite vague queries. Crucially, to ensure strict information alignment with the generation-based baseline (PEToolLLaMA), we feed the prompt with the completely un-truncated interaction history and dynamically inject condition-specific rules (e.g., emphasizing recent tool calls in the chronological setting, or binary feedback in the rating-integrated setting). This rigorous design eliminates prompt-engineering bias, guaranteeing that the observed performance gaps are solely attributable to the inherent architectural differences between the pipelined rewriting paradigm and our end-to-end framework.

Prompt for Zero-shot Parameter Generation

```
## System Message
Your task is to generate the correct input parameters for a given tool based on a user's instruction.

## Instruction Message
User instruction: "{query}"

Selected tool: {tool_name}
Tool description: {tool_description}

{req_desc}
{opt_desc}

Example output #1:
{{"page": 3, "keyword": "schuhe", "sortBy": "recommended", "excludeSponsored": true}}

Example output #2:
{{}}

Rules:
- Required parameters MUST be filled. Find the value from the user instruction. If the value is not found, use the default.
- Include an optional parameter if its value is mentioned or implied in the user instruction. If implied but not specified, use the default.
- Output exactly the JSON object, no extra text. If no parameters are needed, return {{}}.

JSON Output:
```

Figure 6: Prompt template for zero-shot parameter generation.

Prompt for Query Rewriting Assistant

```
## System Message
You are a query rewriting assistant acting as a bridge for a tool retrieval system. Your task is to analyze the user's interaction history, identify their implicit preferences, and rewrite the current vague query into a highly specific tool-invocation query.

Interaction history rule:
{history_rule}

Do NOT output any explanations. Output ONLY the rewritten query.

## Instruction Message
Interaction history:
{full_history_json}

Current User Instruction:
{query}

Task: Rewrite the current instruction to explicitly include the user's preferences.
```

Figure 7: Prompt template for the zero-shot query rewriting bridge model.