ARTS: Autoregressive Tool Selection via Graph Neural Networks

Zhuoqi Hu*
Huazhong University of Science and Technology
Wuhan, China
qizhihuanying@hust.edu.cn

Shiwei Li*
Alibaba Group
Hangzhou, China
Shanghai Jiao Tong University
Shanghai, China
lishiwei909@gmail.com

Haozhao Wang
Huazhong University of Science and Technology
Wuhan, China
hz_wang@hust.edu.cn

Xing Tang
Shenzhen Technology University
Shenzhen, China
xing.tang@hotmail.com

Cheng Wang
Huazhong University of Science and Technology
Wuhan, China
wangcheng20@hust.edu.cn

Xiandi Luo
Huazhong University of Science and Technology
Wuhan, China
togwa_sakiko@hust.edu.cn

Yihao Ouyang
Huazhong University of Science and Technology
Wuhan, China
yihaoouyang@163.com

Yichen Li
Huazhong University of Science and Technology
Wuhan, China
ycli0204@hust.edu.cn

Xiuqiang He
Shenzhen Technology University
Shenzhen, China
he.xiuqiang@gmail.com

Ruixuan Li✉
Huazhong University of Science and Technology
Wuhan, China
rxli@hust.edu.cn

Abstract

As large language models (LLMs) evolve into tool-augmented agents, tool selection becomes a key interface between the model and external resources. However, existing methods struggle to model interactions among tools and generally overlook the causal structure inherent in the tool selection process. In this paper, we propose Autoregressive Tool Selection (ARTS) to explicitly model causal dependencies among tools. Specifically, ARTS formulates toolset construction as a sequential decision-making problem, where the next tool is selected conditioned on the query and the previously selected tools. To further capture rich interactions during this process, ARTS employs a graph neural network (GNN) to jointly model tool–tool and tool–query relationships. In practice, ARTS constructs a graph comprising a query node and candidate tool nodes and explicitly marks the nodes of the selected tools. The resulting marked graph is fed into a GNN to autoregressively predict the next tool until a termination condition is met. Experiments on two public benchmark datasets demonstrate that ARTS consistently outperforms relevant baselines in terms of accuracy and exhibits clear advantages in inference latency. The code is available at https://github.com/qizhihuanying/ARTS_MM.

CCS Concepts

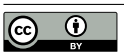
• **Information systems → Retrieval models and ranking**.

Keywords

Autoregressive Tool Selection, Graph Neural Networks

ACM Reference Format:
Zhuoqi Hu, Shiwei Li, Haozhao Wang, Xing Tang, Cheng Wang, Xiandi Luo, Yihao Ouyang, Yichen Li, Xiuqiang He, and Ruixuan Li. 2026. ARTS: Autoregressive Tool Selection via Graph Neural Networks. In *Proceedings of the 34th ACM International Conference on Multimedia (MM '26), November 10–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3767308.3836524

*Both authors contributed equally to this research.



MM '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2213-4/2026/11
https://doi.org/10.1145/3767308.3836524

1 Introduction

As large language models (LLMs) evolve from pure text generators into tool-augmented agents, tool selection becomes a critical interface between the model and external resources [17, 34]. By connecting to external tools, LLMs can take actions in real systems and extend their abilities by using tools for search, data analysis, and planning [20, 27, 37]. Recent domain-specific work further shows that data-driven function-calling strategies can improve LLM-based online financial question answering [31]. Recent work on large-scale tool learning, API selection, and benchmarking [6, 21, 30, 41] collectively points to the same trend: in realistic scenarios, selecting the right tools and composing them effectively has become a core capability for tool-augmented agents [16, 25].

In tool-augmented agents, retrieval often returns a large but noisy set of candidate tools, which must be condensed into a small, usable list for the LLM. Tool reranking addresses this challenge by filtering irrelevant candidates, distinguishing semantically similar tools, and aligning tool priority with user intent [9, 10]. Since LLMs are highly sensitive to the ordering of available tools, reranking critically affects both accuracy and context efficiency. This objective is closely related to performance-driven context compression, which seeks to retain task-useful evidence while removing redundant context [3]. Poor reranking, either due to under-filtering which

may surface misleading tools and lead to incorrect or hallucinated API calls, or due to over-filtering which risks removing rare but necessary tools and thereby weakening multi-step reasoning over tools, can substantially degrade tool-augmented agents [21, 33].

However, current tool reranking and selection methods mainly rely on one-shot decision making. Specifically, retrieval-style pipelines usually estimate each tool's relevance to the query independently, and then keep the top-k tools or those above a fixed threshold. Besides, some work organizes tools by grouping them into categories, or by navigating a precomputed tool-relation graph [6, 15] to reduce the search space, but the final selection is still often decided by this single-stage ranking plus a fixed cutoff. However, tool use during agent execution is naturally causal and sequential [36, 37]. Determining which tool to select and how many tools to use depends not only on the current query but also on the selection history. A purely one-shot formulation has no explicit selection history, making it difficult to use earlier selections to guide later choices. This issue is evident because queries require varying numbers and combinations of tools. Prior studies on LLM behavior analysis have also demonstrated the value of modeling signals beyond surface semantics, including causal interaction patterns and redundant expression structures [38, 39].

To address this issue, we propose **A**uto**R**egressive **T**ool **S**election (**ARTS**), which treats tool selection as a sequential toolset construction process rather than a one-shot ranking problem. Instead of deciding a fixed set of tools in a single pass, ARTS selects tools step by step, where each decision is conditioned on the query and the tools that have already been chosen. This formulation allows earlier selections to directly influence later ones, and naturally supports queries that require different numbers of tools.

To implement this idea, ARTS first builds a graph whose nodes consist of the query and all candidate tools. The graph includes two types of edges: query–tool edges that directly model the interaction between the query and each tool, and tool–tool edges that connect tools with high historical co-occurrence to make their relationships explicit. The graph is then fed into a graph neural network (GNN), which iteratively updates node representations and at each step decides whether to select a new tool or to terminate the selection process. Specifically, at the beginning of each selection step, ARTS injects a query-based state embedding into the nodes of tools that have already been selected. During prediction, it then uses the query node's logit as a dynamic threshold and stops selecting if no remaining tool exceeds this value.

Our contributions can be summarized as follows:

- We cast tool selection as an autoregressive decision-making process, explicitly capturing the causal dependency that each choice should be conditioned on the query and the already selected tools.
- We introduce a GNN to explicitly model both query-tool relevance and tool-tool interactions, enabling structured reasoning over candidate tools.
- We design a query-based state embedding injection to represent the selection history, together with an adaptive stopping rule that uses the query node as a dynamic reference, unifying tool selection and termination in a single autoregressive procedure.
- Experiments on two public benchmarks show that ARTS achieves high NDCG@{1,3,5}, while remaining lightweight and efficient at inference time.

2 Related Work

2.1 Tool Selection and Tool Retrieval

In large tool libraries containing thousands of tools, tool selection is often formulated as an information retrieval problem. A tool's name, description, input and output schema, and example calls are typically concatenated into a single document, and retrieval is performed by measuring query–document similarity, followed by reranking (e.g., BM25 [23]). Recent work has studied this setting systematically: ToolRet builds a benchmark and shows that standard IR models still leave substantial room for improvement [26]; ToolRerank improves reranking by modeling tool hierarchy and adaptive truncation [42]; Re-Invoke enhances robustness without task-specific training through query rewriting and multi-view matching [2]; and COLT incorporates tool co-usage statistics to capture complementary rather than merely semantically similar tools [22]. However, retrieval-based pipelines usually generate a Top-k list in a single pass, treating tools as independent items. This makes it difficult to model complementarity and conditional dependencies when multiple tools must be composed, since each choice may depend not only on the query but also on previously selected tools. Generative and hybrid frameworks address this issue by producing explicit sequences of tool calls, as shown in Toolformer and ReAct [24, 37]. Yet when the tool library becomes large and diverse, purely generative methods face a rapidly expanding search space and are more prone to hallucinated tool calls, so systems such as Gorilla and ToolLLM prepend a retrieval stage that narrows the full library to a smaller candidate set before the model decides which tools to call [20, 21]. Overall, one-shot retrieval is efficient but cannot capture the causal structure of tool composition, while step-by-step LLM planning models this structure at much higher inference cost. Complementary efficiency research considers parameter-efficient adaptation [14] as well as embedding compression and low- or mixed-precision embedding training [11–13]; however, these directions do not directly address candidate-tool selection. Motivated by this, we propose a small, trainable autoregressive selector that chooses tools step by step based on previously selected tools, aiming to provide an efficient way to model causal dependencies in multi-step tool selection.

2.2 Graph Neural Networks

Graph neural networks (GNNs) are effective models for graph-structured data, where node representations are updated by aggregating information from neighboring nodes. They have been widely used in recommendation, retrieval, and question answering. For example, LightGCN models user-item interactions [8], KGAT incorporates relational information from knowledge graphs [35], and PullNet, GRAFT-Net, and GraphRetriever perform reasoning over question-specific subgraphs [19, 28, 29]. GNN-based rerankers can also improve ranking quality by modeling relations among candidates [7]. This makes GNNs a natural choice for tool selection, where the usefulness of one tool may depend on other tools selected with it. For tool selection, structured organization of the tool space

can provide useful priors over common tool combinations and dependencies. AnyTool leverages the hierarchical category structure of APIs to decompose large-scale retrieval into a smaller search process, while ToolNet explicitly represents tools as a directed graph in which weighted edges capture likely transitions between tools, enabling graph-guided multi-step tool navigation [6, 15]. However, these approaches are still largely static: they do not fully model how the importance of tool relations changes as some tools are selected. This suggests that GNNs are a promising backbone for efficient structured tool selection, but they need to be extended to capture query-dependent and sequential selection dynamics.

3 Methodology

In this section, we introduce ARTS, a graph-based framework for autoregressive tool selection. Throughout this section, $\mathrm{Emb}(x) \in \mathbb{R}^d$ denotes the pretrained text embedding of a query or tool description, and $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denotes a graph with node set $\mathcal{V}$ and edge set $\mathcal{E}$. We use these embeddings to initialize query and tool nodes with task-relevant semantic features, providing a shared semantic space for directly comparing each query with its candidate tools [18].

3.1 Problem Formulation

Given a natural-language query q and a candidate tool set $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$, the objective is to select an optimal subset $\mathcal{Y}^* \subseteq \mathcal{T}$ that satisfies the query. In practice, tools are rarely independent: some tools depend on the outputs of others, some provide complementary functionality when used together, and some are mutually exclusive under the same query. Nevertheless, most existing approaches score each tool independently and apply a one-shot Top-k or threshold-based rule to determine the final selection [42]. Such static, single-pass strategies fail to capture the causal relationships among tools and ignore the sequential nature of tool selection.

To explicitly model these causal relationships, we formulate tool selection as an autoregressive decision process over a discrete action space. Specifically, the problem is cast as a finite-horizon Markov decision process (MDP). At step τ, the state S_τ consists of the query q and selection history $\mathcal{H}_{\tau-1} \subseteq \mathcal{T}$, where $\mathcal{H}_0 = \emptyset$. Conditioned on the current state, a policy $\pi(a_\tau \mid S_\tau)$ selects the next action a_τ from

$$\mathcal{A}_\tau = (\mathcal{T} \setminus \mathcal{H}_{\tau-1}) \cup \{\textsc{Stop}\}. \tag{1}$$

where $\setminus$ denotes set subtraction, i.e., $\mathcal{T} \setminus \mathcal{H}_{\tau-1}$ denotes the set of tools not yet selected in previous steps.

3.2 Bi-Level Graph Structure

To implement the policy π, we require a model that can capture interactions among the query, previously selected tools, and remaining candidate tools, and leverage these interactions to determine the next selection. A straightforward solution is to use an LLM by encoding the query and tool information into a prompt and letting the model select the next tool. However, this approach models query-tool and tool-tool interactions only implicitly through semantic representations, and incurs high inference cost when the selection process is repeated over multiple steps.

Instead, we adopt a graph neural network (GNN) as the decision model. By representing the query and tools as nodes and their relationships as edges, the GNN explicitly models both query–tool relevance and tool–tool associations, while remaining efficient at inference time. Accordingly, for each query, we construct a graph that encodes both the query context and tool relations as the input to the GNN. Next, we introduce a bi-level graph structure for constructing this query-specific graph.

3.2.1 Global Tool Graph. We first construct a global graph over tools, $\mathcal{G}^{\text{global}} = (\mathcal{V}_T, \mathcal{E}_T)$. The node set $\mathcal{V}_T$ contains all tools across queries, and the edge set $\mathcal{E}_T$ encodes tool-tool relations derived from historical co-usage statistics computed only from the training data. Each tool node is initialized with a text embedding produced by an embedding model $\mathrm{Emb}(\cdot)$, computed from the tool documentation. The documentation is formed by concatenating the tool name, API name, natural-language description, and parameter specifications. Edges are constructed using historical tool co-occurrence statistics. For each historical query q, we consider its positive tool set $\mathcal{P}(q)$ and count how often each tool pair (t_i, t_j) appears together, yielding a co-occurrence count c_{ij}. If c_{ij} is at least threshold f, an edge is added between t_i and t_j. This global graph explicitly captures direct relationships among tools and provides global structural guidance shared across all queries.

3.2.2 Query-tool Subgraph. For a given query q with a candidate tool list $\mathcal{T}_q = \{t_1, t_2, \ldots, t_N\}$, we construct a query-tool subgraph $\mathcal{G}_q = (\mathcal{V}_q, \mathcal{E}_q)$. The node set is defined as $\mathcal{V}_q = \{v_q\} \cup \{v_1, \ldots, v_N\}$, which consists of one query node v_q and N candidate tool nodes $\{v_i\}_{i=1}^N$. The query node feature is obtained by embedding the user query. Each tool node feature is inherited from the global graph and corresponds to the text embedding of its tool documentation. The subgraph edges are constructed from two sources. First, we add query-tool edges between the query node v_q and each candidate tool node v_i. This allows the query to serve as a persistent reference during message passing and to directly interact with all candidate tools. Second, we inherit tool-tool edges from the global graph. For any candidate tool pair (v_i, v_j), an edge is added if the corresponding tools (t_i, t_j) are connected in $\mathcal{G}^{\text{global}}$. This enables the model to leverage potential associations among tools. At inference time, candidate tools that are not present in $\mathcal{G}^{\text{global}}$ are still added to $\mathcal{G}_q$ as tool nodes, but are connected only to the query node, since no historical tool-tool edges are available for them.

3.3 Autoregressive Tool Selection

Given the query–tool subgraph constructed above, ARTS builds the final tool set through an autoregressive procedure. At each decision step τ, it (i) encodes the current selection history into node representations, (ii) performs message passing on the subgraph, (iii) predicts logits for all tool nodes, and either selects the next tool or terminates the process according to an adaptive stopping rule. Next, we describe these components in turn.

3.3.1 State Embedding Injection. The query-tool subgraph already contains semantic embeddings for the query and tools, generated using a shared embedding model:

$$\mathbf{e}_q = \mathrm{Emb}(q), \qquad \mathbf{e}_i = \mathrm{Emb}(t_i), \tag{2}$$

where $\mathbf{e}_q \in \mathbb{R}^{d_{\mathbf{sem}}}$ and $\mathbf{e}_i \in \mathbb{R}^{d_{\mathbf{sem}}}$ denote the semantic embeddings of query and the i-th tool, respectively. ARTS further learns a projection matrix $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{(d_{\mathbf{sem}} - d_{\mathbf{st}}) \times d_{\mathbf{sem}}}$, where $d_{\mathbf{st}}$ is the state

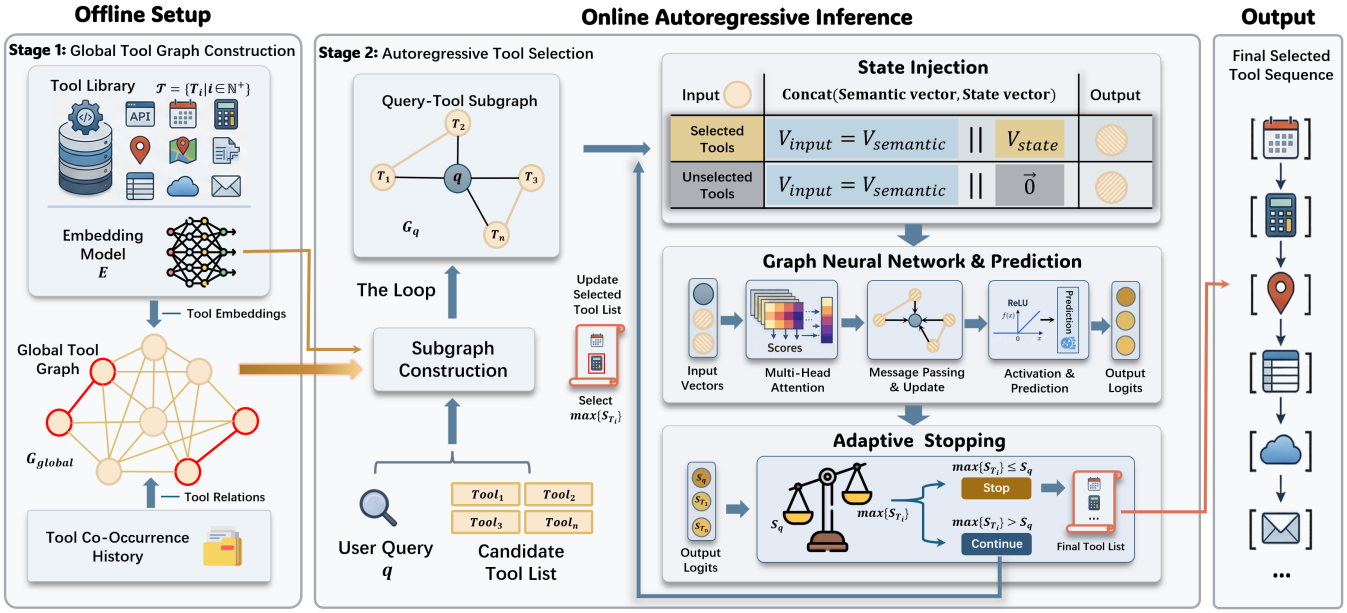


Figure 1: Overview of the ARTS framework. Given a query and a candidate tool list, ARTS constructs a query-tool subgraph from the global tool graph, performs state injection based on the selection history, runs query-aware message passing, and makes autoregressive decisions with an adaptive stopping criterion.

embedding dimension, to map these semantic embeddings into a task-specific space for tool selection.

To enable autoregressive tool selection, the model must be aware of which tools have already been selected at each decision step. To represent the current selection history, we augment node features with an additional state embedding component. For the query node and unselected tool nodes, this component is set to the zero vector. Since tool selection is strongly query-dependent, we generate the state embedding $\mathbf{v}_{\text{state}} \in \mathbb{R}^{d_{\text{st}}}$ from the query representation:

$$\mathbf{v}_{\text{state}} = \mathbf{W}_{\text{state}}\mathbf{e}_q, \tag{3}$$

where $\mathbf{W}_{\text{state}} \in \mathbb{R}^{d_{\text{st}} \times d_{\text{sem}}}$ is a learnable transformation.

At decision step τ, the input representation of each tool node is constructed by concatenating its projected semantic embedding with the state component. Specifically, for a tool node v_i, we define its state-injected representation as:

$$\tilde{\mathbf{h}}_i^{\tau} = \left[\mathbf{W}_{\text{proj}}\mathbf{e}_i \parallel \mathbb{I}[i \in \mathcal{H}_{\tau-1}]\,\mathbf{v}_{\text{state}}\right] \in \mathbb{R}^{d_{\text{sem}}}, \tag{4}$$

where $\mathbb{I}[\cdot]$ is an indicator function that specifies whether tool i has been selected. The query node is processed in the same manner as unselected tool nodes, with state component set to the zero vector.

By injecting the state embedding into node representations, the GNN can explicitly distinguish selected and unselected tools during message passing. This allows the selection history to influence subsequent decisions in a structured and query-conditioned manner.

3.3.2 Message Passing. At decision step τ, the query–tool subgraph together with the state-injected node representations is fed into a graph neural network to perform message passing. The set of node representations is denoted as

$$\tilde{\mathbf{H}}^{\tau} = \{\tilde{\mathbf{h}}_q^{\tau}, \tilde{\mathbf{h}}_1^{\tau}, \ldots, \tilde{\mathbf{h}}_N^{\tau}\}. \tag{5}$$

In this paper, we adopt a graph attention network (GAT) [32] as the message-passing backbone, which aggregates relational information from neighboring nodes via attention mechanisms:

$$\mathbf{H}^{\tau} = \text{GAT}\left(\mathcal{G}_q, \tilde{\mathbf{H}}^{\tau}\right), \tag{6}$$

where $\mathbf{H}^{\tau} = \{\mathbf{h}_q^{\tau}, \mathbf{h}_1^{\tau}, \ldots, \mathbf{h}_N^{\tau}\}$ are output node representations. Through message passing, each node integrates information from the query, neighboring tools, and the encoded selection history.

3.3.3 Logit Prediction and Adaptive Stopping. The output node representations are used to produce logits that guide the next selection decision. Specifically, we learn a shared linear prediction head $\mathbf{W}_p \in \mathbb{R}^{1 \times d_{\text{sem}}}$ that maps node representations to scalar logits:

$$s_q^{\tau} = \mathbf{W}_p\mathbf{h}_q^{\tau}, \qquad s_i^{\tau} = \mathbf{W}_p\mathbf{h}_i^{\tau}. \tag{7}$$

The tool node with the highest logit value is selected as the next tool. Since the selection process should not continue indefinitely, we further introduce an adaptive stopping mechanism. After message passing, the query node aggregates information from the query itself, the previously selected tools, and the remaining candidate tools. This enables the query node to assess whether additional tool selection is necessary under the current selection history. Accordingly, we use the query node logit s_q^{τ} as a dynamic confidence baseline. Tool selection terminates when the maximum logit among the remaining candidate tools is lower than that of the query.

3.4 Training Data Construction

3.4.1 Multi-round Data Construction. To train the autoregressive generation process described above, we expand each original supervised sample into multiple state-specific training steps. Given a sample $(q, \mathcal{T}, \mathcal{Y}^*)$, where q denotes the query, $\mathcal{T}$ is the set of all

candidate tools, and $\mathcal{Y}^* \subseteq \mathcal{T}$ is the set of ground-truth relevant tools, we define $N = |\mathcal{Y}^*|$ as the number of relevant tools. We then construct $N+1$ training instances $\{D_\tau \mid \tau \in \{1, \ldots, N+1\}\}$, each corresponding to a state in which 0 to N relevant tools have already been selected. Each training instance is defined as

$$D_\tau = \{q, \mathcal{H}_{\tau-1}, \mathcal{P}_{\tau-1}, \mathcal{N}\}, \tag{8}$$

where q is the original query, $\mathcal{H}_{\tau-1}$ denotes the tools already selected before step τ, $\mathcal{P}_{\tau-1}$ represents the relevant but unselected tools, and $\mathcal{N}$ contains all irrelevant tools:

$$\mathcal{N} = \mathcal{T} \setminus \mathcal{Y}^*, \qquad \mathcal{P}_{\tau-1} = \mathcal{Y}^* \setminus \mathcal{H}_{\tau-1}. \tag{9}$$

We initialize $\mathcal{H}_0 = \emptyset$. For $\tau \in \{1, \ldots, N\}$, we choose $t_\tau \in \mathcal{P}_{\tau-1}$ and update $\mathcal{H}_\tau = \mathcal{H}_{\tau-1} \cup \{t_\tau\}$. Thus, the final instance D_{N+1} uses $\mathcal{H}_N = \mathcal{Y}^*$ and $\mathcal{P}_N = \emptyset$ to supervise stopping.

3.4.2 Training Objective. To implement the query-based adaptive stopping mechanism, we compute an advantage value for each remaining candidate tool, as follows

$$\Delta_i^\tau = s_i^\tau - s_q^\tau. \tag{10}$$

The advantage measures how much the tool's logit exceeds the confidence threshold implied by the query. We then apply a binary cross-entropy loss to the advantage:

$$\mathcal{L}_i = -\, y_i^\tau \log p_i^\tau - (1 - y_i^\tau) \log\left(1 - p_i^\tau\right), \tag{11}$$

where $p_i^\tau = \sigma\left(\Delta_i^\tau\right)$ and $\sigma(\cdot)$ denotes the sigmoid function. The supervision label y_i^τ is defined such that $y_i^\tau = 1$ if the tool belongs to $\mathcal{P}_{\tau-1}$, and $y_i^\tau = 0$ otherwise. The final training objective aggregates the losses over all candidate tool nodes at the current step.

4 Experiments

4.1 Datasets and Experimental Settings

4.1.1 Datasets. We evaluate tool selection performance on two public datasets: ToolBench [21] and ToolRet [26]. For ToolBench, we adopt the G1 (single-tool, intra-category), G2 (multi-tool, intra-category), and G3 (multi-tool, inter-category) splits. Here, intra-category means that all relevant tools for a query come from the same tool category, whereas inter-category queries require composing tools across different categories within the global API collection. Here, single-tool refers to the source tool rather than the evaluation unit: tools may expose multiple APIs, and all ToolBench metrics are computed at the API level. For ToolRet, we use a processed version of the ToolRet-Training set, where each query is converted into a ranking instance consisting of a candidate set and its relevant tools. For each query, we pool the ground-truth positive tools and the negative tools to form the candidate tool set $\mathcal{T}_q$, and treat the positive tools as the relevant set $\mathcal{Y}^*$. Consequently, each sample in our evaluation is formalized as a tuple $(q, \mathcal{T}_q, \mathcal{Y}^*)$.

4.1.2 Embedding Models. ARTS primarily employs two open-source embedding models: Qwen3-Embedding-0.6B [40] and BGE-M3 [1]. Both are used to construct initial semantic vectors for queries and tools with an embedding dimension of 1024, and provide retrieval features for the Dense Retrieval baseline.

4.1.3 Comparative Methods. We compare ARTS against the following baselines: (1) **BERT (fine-tuned)**: a supervised BERT-based [5] dense retriever trained on the training split to estimate query-tool relevance and produce a ranking over the candidate set; (2) **Qwen3 (dense retrieval)**: a zero-shot bi-encoder baseline that embeds the query and each candidate tool with Qwen3-Embedding-0.6B [40] and ranks tools by cosine similarity, without task-specific training. (3) **Qwen3-Embed+Reranker**: a strong neural pipeline that uses Qwen3-Embedding-0.6B for dense retrieval and Qwen3-Reranker-0.6B for candidate reranking; (4) **BGE-M3 hybrid**: a hybrid baseline that combines dense, sparse, and multi-vector matching with BGE-M3 [1]; (5) **COLT**: a tool retrieval method that enhances semantic matching with tool collaboration signals through graph propagation, improving completeness and reducing redundancy [22]; (6) **Re-Invoke**: a training-free retrieval baseline that expands tool descriptions with synthetic queries and matches user intent under multiple views [2]; (7) **ToolLLaMA**: a tool-use LLM baseline fine-tuned on ToolBench, evaluated under the same candidate-set selection protocol as our other methods [21]; (8) **LLM (DS-V3.2-1shot)**: a large language model baseline (based on DeepSeek-V3.2 [4]) without training that generates a ranking of candidate tools by relevance and additionally outputs a selected subset, constrained to be a prefix of the ranking; (9) **LLM (DS-V3.2-Multi)**: at each round, the LLM selects one previously unselected tool or decides to stop; upon stopping, it also produces a ranking over all remaining tools; (10) **ARTS (Ours)**: our autoregressive GNN with history-state injection and adaptive stopping. We instantiate ARTS with different embedding initializers, denoted as **ARTS(BGE-M3)** and **ARTS(Qwen3-Embedding-0.6B)**; the model architecture and training procedure are identical, and only the embedding encoder differs.

4.1.4 Implementation Details. ARTS uses GAT (4 heads, residual connection) with layer count $L = 5$, hidden dimension $H = 1024$, and state dimension of 64. We use a batch size of 64, AdamW with a learning rate of 1×10^{-4}, 10k warmup steps. We train for a maximum of 100 epochs and perform early stopping (patience=10) based on the performance on the validation set. BERT is implemented by fine-tuning bert-base-uncased with learning rate 2×10^{-5} for 5 epochs. COLT follows the released ToolBench setting with learning rate 10^{-3}, weight decay 10^{-7} and the default collaboration hyperparameters. Re-Invoke is implemented with Qwen3-8B as the local generation model and Qwen3-Embedding-0.6B as the embedding model. ToolLLaMA is implemented using LLaMA-2-7B fine-tuned on ToolBench. All other hyperparameters use the default settings.

4.1.5 Evaluation Metrics. We evaluate performance from two complementary perspectives. First, we use NDCG@{1,3,5} as ranking metrics, computed per query and averaged over all queries. These metrics measure how well a method orders candidate tools, independent of how the final tool set is determined, and thus provide a fair basis for comparing ranking quality across methods. Table 1 reports these results. Second, we evaluate the final tool set using Set F1 and Exact Match. Let $\hat{\mathcal{Y}}$ and $\mathcal{Y}^*$ denote the predicted and ground-truth tool sets, respectively. We define

$$P = \frac{|\hat{\mathcal{Y}} \cap \mathcal{Y}^*|}{|\hat{\mathcal{Y}}|}, \qquad R = \frac{|\hat{\mathcal{Y}} \cap \mathcal{Y}^*|}{|\mathcal{Y}^*|}, \qquad \text{Set F1} = \frac{2PR}{P+R}. \tag{12}$$

Table 1: Performance of various methods on ToolBench and ToolRet. All results are averaged over three runs. AVG denotes (NDCG@1 + NDCG@3 + NDCG@5)/3. The best results are highlighted in bold, while the second-best results are underlined.

Method	ToolBench G1				ToolBench G2				ToolBench G3				ToolRet			
	@1	@3	@5	AVG	@1	@3	@5	AVG	@1	@3	@5	AVG	@1	@3	@5	AVG
BERT (fine-tuned)	0.9272	0.9109	0.9413	0.9265	0.8656	0.8602	0.9168	0.8809	0.9159	0.8760	0.9314	0.9078	0.8880	0.9096	0.9258	0.9078
Qwen3 (dense retrieval)	0.8924	0.8862	0.9224	0.9003	0.8482	0.8054	0.8823	0.8453	0.8074	0.7085	0.8162	0.7774	0.7961	0.8142	0.8419	0.8174
Qwen3-Embed+Reranker	0.9461	0.9149	0.9437	0.9349	0.9149	0.8684	0.9265	0.9033	0.8990	0.7903	0.8774	0.8556	0.8663	0.8842	0.9025	0.8843
BGE-M3 hybrid	0.9235	0.8984	0.9272	0.9164	0.8955	0.8353	0.9021	0.8776	0.8674	0.7532	0.8356	0.8187	0.8097	0.8366	0.8584	0.8349
COLT	0.9486	0.9478	0.9660	0.9541	0.8988	0.8707	0.9195	0.8963	0.8899	0.8441	0.9002	0.8781	0.8390	0.8581	0.8800	0.8590
Re-Invoke	0.9549	**0.9497**	0.9559	0.9535	**0.9443**	0.9318	0.9434	0.9398	0.9417	0.9253	0.9378	0.9349	0.9032	0.9165	0.9206	0.9134
ToolLLaMA	0.9466	0.9402	0.9467	0.9445	0.9332	0.9289	0.9401	0.9341	0.9249	0.9162	0.9398	0.9270	0.9016	0.8945	0.9137	0.9033
LLM (DS-V3.2-1shot)	0.9423	0.9079	0.9343	0.9282	0.9201	0.9049	0.9336	0.9195	0.8749	0.8423	0.8937	0.8703	0.8896	0.9260	0.9368	0.9175
LLM (DS-V3.2-Multi)	**0.9573**	0.9492	**0.9673**	**0.9579**	0.9359	**0.9331**	**0.9570**	**0.9420**	0.8991	0.8750	0.9335	0.9025	0.9018	**0.9293**	**0.9431**	0.9247
ARTS(BGE-M3)	0.9565	0.9412	0.9597	0.9525	0.9401	0.9167	0.9471	0.9346	0.9450	0.9112	0.9456	0.9339	0.9080	0.9219	0.9338	0.9212
ARTS(Qwen3)	0.9527	0.9398	0.9598	0.9508	0.9366	0.9216	0.9486	0.9356	**0.9817**	**0.9426**	**0.9610**	**0.9618**	**0.9202**	0.9275	0.9420	**0.9299**

Table 2: Tool set quality on ToolBench and ToolRet. Set F1 measures partial overlap between the predicted and ground-truth tool sets, while Exact Match requires the two sets to be identical. These metrics better reflect practical tool selection quality.

Method	ToolBench G1		ToolBench G2		ToolBench G3		ToolRet	
	Set F1	Exact Match	Set F1	Exact Match	Set F1	Exact Match	Set F1	Exact Match
BERT (fine-tuned)	0.8241	0.6010	0.7628	0.5224	0.7465	0.4144	0.7725	0.5450
Qwen3 (dense retrieval)	0.7866	0.5358	0.7049	0.4753	0.5984	0.3685	0.7104	0.4611
Qwen3-Embed+Reranker	0.8151	0.6077	0.7309	0.5154	0.6065	0.4128	0.7378	0.5210
BGE-M3 hybrid	0.7471	0.4990	0.6508	0.4689	0.5877	0.3364	0.6229	0.4729
COLT	0.8422	0.6211	0.8014	0.6079	0.8175	0.5092	0.7210	0.6268
Re-Invoke	0.8699	0.6491	0.8523	0.6533	0.8465	0.5657	0.7845	0.6835
ToolLLaMA	0.8572	0.6257	0.8413	0.6399	0.8368	0.5459	0.7691	0.6631
LLM (DS-V3.2-1shot)	0.8381	0.5843	0.7974	0.5538	0.7171	0.3945	0.8040	0.5736
LLM (DS-V3.2-Multi)	**0.9131**	0.6888	0.8561	0.6783	0.8133	0.5138	0.7926	0.6028
ARTS(BGE-M3)	0.8825	**0.7089**	0.8601	0.6545	0.8311	0.5520	0.8438	0.6851
ARTS(Qwen3)	0.8786	0.6951	**0.8667**	**0.6806**	**0.8583**	**0.5872**	**0.8658**	**0.7035**

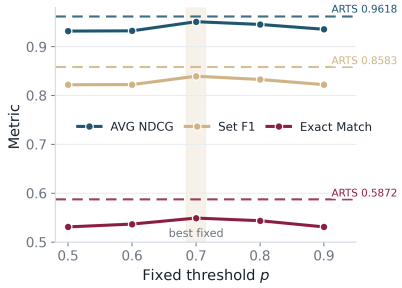
Table 3: Inference latency on ToolBench G3. Time/API reports the per-selection-step latency, while Time/Query sums the end-to-end latency over all rounds for each query (thus reflecting the accumulated cost of multi-round methods). We report Mean and P90 (which means 90% of queries/steps finish within this time) to characterize both average and tail latency.

Metric	Models																	
	BERT		Qwen3+Rerank		BGE-M3		COLT		Re-Invoke		ToolLLaMA		LLM-1shot		LLM-Multi		ARTS(Qwen3)	
	Mean	P90	Mean	P90	Mean	P90	Mean	P90	Mean	P90	Mean	P90	Mean	P90	Mean	P90	Mean	P90
Time/API (s)	0.046	0.062	0.366	0.467	0.148	0.168	0.010	0.018	8.110	14.134	6.372	10.826	13.819	19.527	13.276	19.177	0.014	0.016
Time/Query (s)	0.046	0.062	0.366	0.467	0.148	0.168	0.010	0.018	8.110	14.134	21.792	27.137	13.819	19.527	39.889	59.980	0.050	0.067

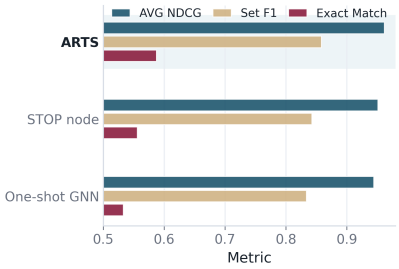
Exact Match is 1 if $\hat{\mathcal{Y}} = \mathcal{Y}^*$ and 0 otherwise. These metrics reflect whether the predicted tool set is suitable for downstream use, since a method with strong ranking quality may still select too many or too few tools. Table 2 reports these set-level results. For ranking baselines that do not natively output a tool set, we truncate the ranked list at K_{avg}, the average number of ground-truth tools per query in the corresponding dataset, and use the top-K_{avg} tools as the final prediction, which approximates the typical tool set size well. For adaptive methods such as ARTS, we directly use the predicted tool set. We additionally report inference latency on ToolBench G3 in Table 3, including both per-step latency (Time/API) and end-to-end latency per query (Time/Query), with Mean and P90 used to characterize average and tail latency, respectively.

4.2 Main Results

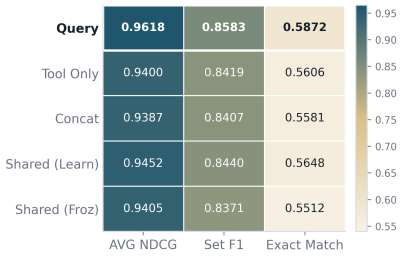
Table 1 reports ranking quality on ToolBench and ToolRet. On the relatively easier ToolBench G1 and G2 splits, LLM (DS-V3.2-Multi) achieves the best average NDCG, with 0.9579 and 0.9420, respectively. On G1, strong retrieval baselines are already highly competitive: COLT reaches 0.9541 and Re-Invoke reaches 0.9535, while both ARTS variants remain close to the top. A similar pattern holds on G2, where Re-Invoke (0.9398) and the two ARTS variants (0.9346/0.9356) stay competitive with the best LLM result. These results suggest that when the relevant tools are still relatively structured and easier to separate, several strong baselines can already produce very accurate rankings. ARTS(Qwen3) performs best on ToolRet, reaching an average NDCG of 0.9299.



(a) Ablation of the stopping threshold.



(b) Selection and stopping mechanisms.



(c) Ablation of state injection strategies.

Figure 2: Ablation studies of ARTS on ToolBench G3. We evaluate how key design choices affect overall performance: (a) the stopping threshold design (dynamic query-adaptive threshold vs. fixed threshold), (b) the selection and stopping mechanism (adaptive stopping vs. an explicit STOP node and a one-shot GNN), and (c) different state embedding injection methods used to encode which tools have already been selected under the current query.

On G3, ARTS(Qwen3) achieves the best average NDCG, reaching 0.9618. This is particularly meaningful because G3 requires multi-tool, inter-category composition rather than selecting tools from a single category. In this setting, the model must not only identify individually relevant tools, but also rank complementary tools correctly under stronger cross-category interference. This is exactly the scenario where explicitly modeling tool relations and selection history becomes especially important.

Table 2 evaluates the final selected tool set. Here the advantage of ARTS becomes clearer. ARTS(Qwen3) achieves the best Set F1 and Exact Match on G2, G3, and ToolRet, while ARTS(BGE-M3) obtains the best Exact Match on G1. Although LLM (DS-V3.2-Multi) gives the highest Set F1 on G1, the ARTS variants are more consistent on Exact Match. The contrast between Table 1 and Table 2 is important: A strong ranked list does not always produce a strong final tool set. Methods with a fixed cutoff often return too many or too few tools. In contrast, ARTS decides which tool to add next and when to stop based on the query and the tools already selected.

Table 3 reports inference latency on ToolBench G3. COLT is the fastest method overall, with 0.010s per query. ARTS(Qwen3) also remains highly efficient, with 0.014s Time/API and 0.050s Time/Query, while retaining the strongest overall results on G3. It is substantially faster than Re-Invoke (8.110s) and Qwen3-Embed+Reranker (0.366s), and far below the LLM-based baselines, whose per-query latency ranges from 13.819s to 39.889s. Its tail latency is also low, with a P90 Time/Query of 0.067s. Overall, ARTS achieves strong ranking results, better final tool sets, and low inference cost. This is consistent with our original goal of improving compositional tool selection without sacrificing efficiency.

4.3 Ablation Studies

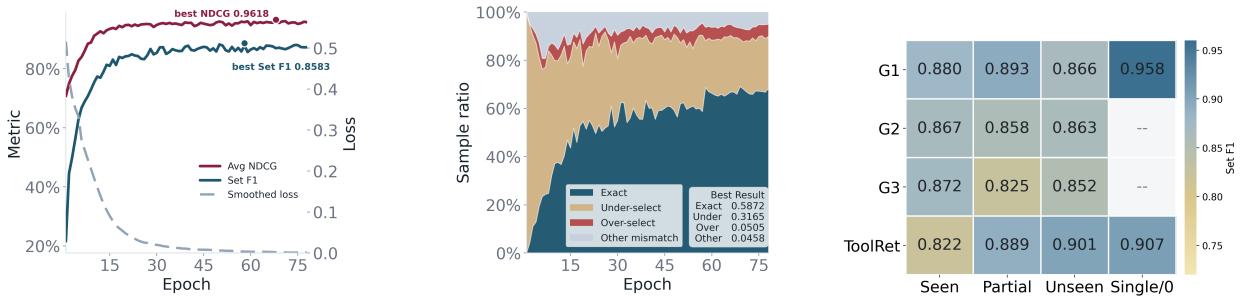
We further analyze ARTS on ToolBench G3 from several aspects, including key design choices, training behavior, and performance under different overlap settings. Unless otherwise specified, these experiments use the same model configuration and training setup as in the main experiments.

4.3.1 Graph Construction and Structure. Table 4 shows that the graph structure used in ARTS is important. In the upper block, the default co-occurrence graph gives the best results on all three metrics. Removing tool-tool edges, replacing them with random connections, or using a fully connected graph all hurts performance. The *Co-occurrence only* baseline, which ranks tools only by historical co-occurrence counts without training or using query semantics, performs much worse, indicating that co-occurrence statistics alone are not sufficient for this task. The lower block studies the minimum co-occurrence frequency used to build the graph. The best results are obtained with f=2. When the threshold is increased further, performance drops, indicating that some useful tool-tool relations are being removed. We therefore use f=2 in the main experiments.

Table 4: Ablation of graph construction on ToolBench G3. The upper block compares graph-structure variants and a statistic-only baseline. The lower block studies the minimum co-occurrence frequency within the default co-occurrence graph. Best results within each block are highlighted in bold.

Method	AVG NDCG	Set F1	Exact Match
Graph structure variants			
ARTS (co-occurrence, default)	**0.9618**	**0.8583**	**0.5872**
No tool-tool edges	0.8651	0.7512	0.4373
Random graph	0.8538	0.7604	0.4694
Fully connected graph	0.8317	0.7338	0.4190
Co-occurrence only	0.6035	0.4953	0.2324
Minimum co-occurrence frequency			
ARTS (co-occurrence, f=1)	0.9494	0.8547	0.5612
ARTS (co-occurrence, f=2)	**0.9618**	**0.8583**	**0.5872**
ARTS (co-occurrence, f=3)	0.9390	0.8234	0.5688
ARTS (co-occurrence, f=5)	0.9352	0.8165	0.5581

4.3.2 Threshold Mechanism. Figure 2(a) compares two threshold designs for deciding when to stop selecting tools. In the fixed-threshold variants, a candidate tool is selected whenever its predicted score exceeds a global cutoff p, and the same cutoff is used for all queries and all decision steps. In ARTS, by contrast, the threshold is dynamic: the query node provides the stopping baseline, and the model stops when no remaining tool exceeds that query-conditioned value. Among the fixed settings, p=0.7 performs



(a) Performance. **(b) Selection behavior.** **(c) Overlap buckets.**

Figure 3: Further analysis of ARTS(Qwen3). (a) and (b) show the learning dynamics and selection behavior on ToolBench G3, while (c) reports Set F1 across overlap buckets on G1, G2, G3, and ToolRet.

best, reaching 0.9512 AVG NDCG, 0.8396 Set F1, and 0.5489 Exact Match, but it still remains below ARTS on all three metrics. Lower cutoffs and higher cutoffs both degrade performance, indicating that a single global threshold is sensitive to the chosen value and cannot robustly balance ranking quality with final tool selection. This is expected because different queries require different numbers of tools and different stopping boundaries. By contrast, the dynamic threshold adapts the stopping decision to the current query state, yielding consistently better ranking and final tool set quality.

4.3.3 Autoregressive Selection and Stopping Mechanism. Figure 2(b) studies two design choices in ARTS. In the *STOP node* variant, we add an explicit virtual STOP node into the graph and terminate selection once the model chooses that node. This lowers performance to 0.9513 AVG NDCG, 0.8427 Set F1, and 0.5566 Exact Match, suggesting that explicit termination through a separate node is less effective than using the query-conditioned threshold directly. In the *One-shot GNN* variant, we remove the multi-round autoregressive procedure and predict the whole tool set in a single pass. This degrades performance further to 0.9444, 0.8335, and 0.5321. The largest drop appears in Exact Match, suggesting that autoregressive selection and adaptive stopping are especially important when the model must assemble the full tool combination precisely rather than merely rank relevant tools near the top. Overall, ARTS benefits from both sequential selection, where later choices can depend on earlier ones, and adaptive stopping, which keeps termination aligned with the current query state.

4.3.4 State Injection Method. Figure 2(c) compares several ways to construct the injected state vector for already selected tools. In *Query*, the state is derived from the query representation and used in ARTS. In *Tool Only*, the state is built only from the selected tool embedding; in *Concat*, it is built from the concatenation of query and tool embeddings; in *Shared (Learn)*, all selected tools share a single learnable state vector; and in *Shared (Froz)*, this shared vector is fixed. Query-based state injection performs best on all three metrics, while all alternatives remain clearly below it. This pattern is consistent with the role of the state signal in ARTS: it should encode the current selection history with respect to the query, rather than merely adding tool-specific semantics or using a globally shared vector.

4.3.5 Further Analysis. Figure 3(a) shows the training dynamics of ARTS on ToolBench G3. As the loss decreases, both ranking quality and tool set quality improve steadily, suggesting that the GNN gradually learns more informative query-tool and tool-tool interactions rather than relying only on initial co-occurrence patterns. Figure 3(b) further shows how the composition of prediction outcomes changes during training. A consistent pattern is that *Under-select* remains much more common than *Over-select*, which is expected because ARTS selects one tool per round and stops once no remaining candidate passes the current threshold, so missing a needed tool already leaves the final tool set incomplete, whereas over-selection would require repeatedly choosing irrelevant tools. Figure 3(c) analyzes Set F1 under different train-test overlap buckets, where test queries are grouped by whether their required tool combinations are unseen, partially seen, or fully overlapped with combinations observed during training. Across G1, G2, G3, and ToolRet, performance on the *Unseen* bucket remains competitive, suggesting that ARTS does not simply memorize observed co-occurrence patterns. The pattern varies by dataset: *Single/0* bucket is strongest in G1 and ToolRet because queries without tool-pair composition are easier, whereas in G2 and G3 this bucket is absent since both of them are multi-tool settings. Overall, the co-occurrence graph helps not only for previously observed relations, but also for partially seen and unseen tool combinations, suggesting that ARTS can still work well when the required tool relations are not fully covered in training.

5 Conclusion

In this paper, we propose ARTS, an autoregressive framework for tool selection in tool-augmented language model agents. By modeling tool selection as a sequential decision-making process, ARTS explicitly captures the causal dependency between successive tool choices. The proposed graph-based formulation enables joint reasoning over query–tool relevance and tool–tool interactions, while a query-based state injection and adaptive stopping unify tool selection and termination within a single process. Extensive experiments on ToolBench and ToolRet demonstrate that ARTS significantly outperforms existing baselines, particularly in complex scenarios requiring complementary tool sets. These results highlight the importance of capturing tool relationships and selection history, helping to build more reliable and efficient tool-using agents.

Acknowledgments

This work is supported by the National Key Research and Development Program of China under grant 2024YFC3307900; the National Natural Science Foundation of China under grants 62376103, 62302184, 62436003 and 62206102; Major Science and Technology Project of Hubei Province under grant 2024BAA008; Hubei Science and Technology Talent Service Project under grant 2024DJC078; Ant Group through CCF-Ant Research Fund. The computation is completed in the HPC Platform of Huazhong University of Science and Technology.

References

[1] Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 2318–2335. doi:10.18653/v1/2024.findings-acl.137

[2] Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024. Re-Invoke: Tool Invocation Rewriting for Zero-Shot Tool Retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 4705–4726. doi:10.18653/v1/2024.findings-emnlp.270

[3] Ziqiang Cui, Yunpeng Weng, Xing Tang, Peiyang Liu, Shiwei Li, Bowei He, Jiamin Chen, Yansen Zhang, Xiuqiang He, Rui Zhang, et al. 2026. Less Is More: Elevating RAG via Performance-Driven Context Compression. In *Forty-third International Conference on Machine Learning*.

[4] DeepSeek-AI. 2025. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. arXiv:2512.02556 [cs.CL] https://arxiv.org/abs/2512.02556

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. doi:10.18653/v1/N19-1423

[6] Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. AnyTool: self-reflective, hierarchical agents for large-scale API calls. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 470, 18 pages.

[7] Andrea Giuseppe Di Francesco, Christian Giannetti, Nicola Tonellotto, and Fabrizio Silvestri. 2024. Graph Neural Re-Ranking via Corpus Graph. arXiv:2406.11720 [cs.IR] https://arxiv.org/abs/2406.11720

[8] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 639–648. doi:10.1145/3397271.3401063

[9] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research* (2022). https://openreview.net/forum?id=jKN1pXi7b0

[10] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6769–6781. doi:10.18653/V1/2020.EMNLP-MAIN.550

[11] Shiwei Li, Huifeng Guo, Lu Hou, Wei Zhang, Xing Tang, Ruiming Tang, Rui Zhang, and Ruixuan Li. 2023. Adaptive low-precision training for embeddings in click-through rate prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4435–4443.

[12] Shiwei Li, Huifeng Guo, Xing Tang, Ruiming Tang, Lu Hou, Ruixuan Li, and Rui Zhang. 2024. Embedding compression in recommender systems: A survey. *Comput. Surveys* 56, 5 (2024), 1–21.

[13] Shiwei Li, Zhuoqi Hu, Xing Tang, Haozhao Wang, Shijie Xu, Weihong Luo, Yuhua Li, Xiuqiang He, and Ruixuan Li. 2024. Mixed-precision embeddings for large-scale recommendation models. *arXiv preprint arXiv:2409.20305* (2024).

[14] Shiwei Li, Xiandi Luo, Haozhao Wang, Xing Tang, Ziqiang Cui, Dugang Liu, Yuhua Li, Xiuqiang He, and Ruixuan Li. 2025. Beyond Higher Rank: Token-wise Input-Output Projections for Efficient Low-Rank Adaptation. In *Advances in Neural Information Processing Systems*.

[15] Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. 2024. ToolNet: Connecting Large Language Models with Massive Tools via Tool Graph. arXiv:2403.00839 [cs.AI] https://arxiv.org/abs/2403.00839

[16] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: plug-and-play compositional reasoning with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1882, 32 pages.

[17] Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented Language Models: a Survey. *Transactions on Machine Learning Research* (2023). Survey Certification. https://openreview.net/forum?id=jh7wH2AzKK

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'13)*. Curran Associates Inc., Red Hook, NY, USA, 3111–3119.

[19] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2020. Knowledge Guided Text Retrieval and Reading for Open Domain Question Answering. arXiv:1911.03868 [cs.CL] https://arxiv.org/abs/1911.03868

[20] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 126544–126565. doi:10.52202/079017-4020

[21] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=dHng2O0Jjr

[22] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards Completeness-Oriented Tool Retrieval for Large Language Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management* (Boise, ID, USA) *(CIKM '24)*. Association for Computing Machinery, New York, NY, USA, 1930–1940. doi:10.1145/3627673.3679847

[23] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Text Retrieval Conference*. https://api.semanticscholar.org/CorpusID:41563977

[24] 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* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2997, 13 pages.

[25] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: solving AI tasks with chatgpt and its friends in hugging face. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1657, 27 pages.

[26] Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval Models Aren't Tool-Savvy: Benchmarking Tool Retrieval for Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 24497–24524. doi:10.18653/v1/2025.findings-acl.1258

[27] Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023. RestGPT: Connecting Large Language Models with Real-World RESTful APIs. arXiv:2306.06624 [cs.CL] https://arxiv.org/abs/2306.06624

[28] Haitian Sun, Tania Bedrax-Weiss, and William Cohen. 2019. PullNet: Open Domain Question Answering with Iterative Retrieval on Knowledge Bases and Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 2380–2390. doi:10.18653/v1/D19-1242

[29] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational

Linguistics, Brussels, Belgium, 4231–4242. doi:10.18653/v1/D18-1455

[30] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases. arXiv:2306.05301 [cs.CL] https://arxiv.org/abs/2306.05301

[31] Xing Tang, Hao Chen, Shiwei Li, Fuyuan Lyu, Weijie Shi, Lingjie Li, Dugang Liu, Weihong Luo, Xiku Du, and Xiuqiang He. 2026. Data-Driven Function Calling Improvements in Large Language Model for Online Financial QA. In *Proceedings of the ACM Web Conference 2026*. 7821–7832.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

[33] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research* (2024). https://openreview.net/forum?id=ehfRiF0R3a

[34] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Front. Comput. Sci.* 18, 6 (March 2024), 26 pages. doi:10.1007/s11704-024-40231-1

[35] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 950–958. doi:10.1145/3292500.3330989

[36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.

[37] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=WE_vluYUL-X

[38] Xiaoquan Yi, Haozhao Wang, Yining Qi, Wenchao Xu, Rui Zhang, Yuhua Li, and Ruixuan Li. 2025. ChatbotID: Identifying Chatbots with Granger Causality Test. *Advances in Neural Information Processing Systems* 38 (2025), 80817–80842.

[39] Xiaoquan Yi, Haixing Wu, Haozhao Wang, Yichen Li, Yuhua Li, Rui Zhang, and Ruixuan Li. 2026. UMPIRE: Unveiling LLM-generated Posts via Redundant Expressions. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 30901–30913.

[40] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. arXiv:2506.05176 [cs.CL] https://arxiv.org/abs/2506.05176

[41] Lirui Zhao, Yue Yang, Kaipeng Zhang, Wenqi Shao, Yuxin Zhang, Yu Qiao, Ping Luo, and Rongrong Ji. 2024. DiffAgent: Fast and Accurate Text-to-Image API Selection with Large Language Model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 6390–6399.

[42] Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024. ToolRerank: Adaptive and Hierarchy-Aware Reranking for Tool Retrieval. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. 16263–16273.