MILLISECOND-LEVEL TOOL-CALL PREDICTION FOR LLM AGENT CACHE ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents complete tasks by iteratively invoking external tools, and every call exposes hundreds of milliseconds of network and execution latency. We propose a graph neural network (GNN) next-tool-call predictor that drives cache prefetching: each prediction costs milliseconds on CPU with zero LLM tokens, meeting both the latency budget and the context-sensitivity requirement of online prefetching. Existing predictors satisfy only one of the two. Millisecond trajectory-statistics predictors (k-gram mining, smoothed Markov chains) fit the budget but read only historical call patterns, never the conversation, so they cannot tell apart the next calls implied by the same tool sequence under different intents. Context-aware LLM speculators model the dialogue, but each step costs seconds of GPU inference and every extra candidate needs a fresh generation, making a full-candidate ranking within budget—a prerequisite of tiered prefetching—impossible. Our predictor runs on a global tool graph of tool and semantic-state nodes whose transition edges embed statistical transition probabilities; the conversational state (request, executed calls with truncated results, current user message) is encoded by a single sparse BM25 encoder into a context vector, the historical tool-embedding sequence by a GRU into a state vector, and both are injected into a relational graph convolutional network (R-GCN). Candidates are scored by a context-aware node head and a conditional edge head, additively fused with a Markov prior as low-frequency fallback, and the top-K tools receive tiered prefetch. On public agent trajectories, our method attains $80.20\pm1.61\%$ and $84.89\pm0.49\%$ top-1 on the two τ-bench retail actors (GPT-4o and Claude 3.5) of Speculative Actions and $65.39\pm10.17\%$ on APIGen-MT, corresponding to 1.79–$2.09\times$ cache speedup. Ablations validate the fusion design: transition statistics, Markov prior, and context-conditioned GNN together adapt to both prediction regimes.

1 INTRODUCTION

LLM agents follow an iterative loop of "reason $\rightarrow$ call a tool $\rightarrow$ wait for the result $\rightarrow$ continue reasoning." Each call interacts with external systems and adds non-negligible network-transport and process-execution latency; measurements show tool execution accounts for 45–57% of end-to-end agent latency (Sui et al., 2026), the key bottleneck for responsiveness. *Prefetching* hides this latency: before the LLM finishes the current turn, predict the next tool and start its preparation in parallel, absorbing part of the tool latency inside the LLM's own reasoning window.

The predictor faces a hard latency constraint: prediction must cost far less than tool execution. A single call typically takes 200–800 ms (Sui et al., 2026); against this budget, existing mechanisms emphasize either latency or semantics, rarely both. Millisecond trajectory-statistics predictors—pattern mining (Sui et al., 2026), sandbox prewarming (Zhang et al., 2026), cache management (Zhai et al., 2026)—fit the budget, but their signal derives entirely from historical statistics without modeling the current session, so they cannot differentiate the next call of the same tool sequence under different user intents. LLM-based speculators (Ye et al., 2025; Ji et al., 2026) model the dialogue accurately, but each step needs 1–3 s of GPU inference—more than the latency they hide—and a calibrated low-cost ranking over all candidates remains out of reach.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In short, a fundamental trade-off persists: statistical predictors are fast but context-blind, LLM predictors are semantic but slow. We close it with a lightweight R-GCN predictor that conditions on the full conversational state at 18.1 ms per step on CPU, giving online prefetching low latency and strong context awareness at once.

Our contributions are:

1. **A millisecond conditional-transition predictor.** The Conditional-Transition Graph Neural Network (CT-GNN) combines R-GCN, context-aware heads, transition statistics, and a Markov prior, with a single BM25 encoder fusing multi-source conversational state: zero-LLM-token prediction at 18.1 ms per step on CPU, 65–85% top-1 and 1.79–2.09$\times$ cache speedup on three public scenarios (§4.2).
2. **Controlled fair evaluation.** Under identical test splits and information sets we reproduce four mainstream approaches and additionally give LLM competitors full conversation history. The zero-LLM GNN leads the strongest reproduced competitor by 5.5–13.3 pp at 110–160$\times$ lower cost; even with competitor inference cost fully waived, their cache gains still trail ours—accuracy does not convert into prefetch benefit, ranking does (§4.2).
3. **Boundary-condition analysis.** Whether next-tool transitions are context-conditioned decides the method choice: context-conditioned scenarios let the GNN beat counting by up to +25.6 pp inside restricted candidate sets; near-Markovian scenarios make counting close to Bayes-optimal, yet transition statistics restore the GNN's lead. Our method is never worse than counting in either regime, so deployment requires no prior regime detection (§4.3).
4. **Reverse evaluation of planners.** Training the state-of-the-art planner GTool on our data and injecting its predictions into our prefetch framework shows that one-shot planning cannot convert into prefetch benefit—even with inference cost waived it fails to beat naive counting—establishing the necessity of online step-by-step prediction (§4.4).

2 Related Work

2.1 LLM tool planning

Existing work improves LLM tool calling along two paradigms. Prompt-based methods such as ReAct (Yao et al., 2023) and Plan-and-Solve (Wang et al., 2023) steer multi-step reasoning and tool selection through chain-of-thought prompt design; they generalize well zero-shot but hinge on the backbone's intrinsic reasoning ability, and every planning step triggers a full LLM forward pass at high latency. Model-augmented methods attach external modules: GTool (Chen et al., 2026) builds a request-specific tool dependency graph, encodes it into structural representations injected into a frozen 7B LLM to optimize selection and ordering; GNN4Plan (Wu et al., 2024) uses a GNN to assist an LLM with sub-task dependency and selection in task graphs. Both target compositional decomposition and long-horizon planning quality, and their cost places them outside the online-prefetch path: they suit offline decomposition or high-level policy, whereas our predictor fills the per-step low-latency niche with high-accuracy next-call prediction under an extremely small budget.

2.2 Agent acceleration via prediction and speculation

To hide tool-execution latency, recent work follows a *predict-then-prepare* paradigm: before the LLM finishes the current turn, predict the next call and in parallel prewarm sandboxes, establish connections, or execute. According to the source of the prediction signal, systems fall into three classes.

Trace-based external predictors mine recurring patterns from historical trajectories at negligible cost. PASTE (Sui et al., 2026) mines repeated tool-call subsequences and speculatively executes pattern-matched next calls with parameter binding during generation, overlapping execution with text generation and cutting task completion time by 43.5%. SpecBox (Zhang et al., 2026) prewarms MCP sandboxes mid-generation via streaming keyword matching and semantic embedding, with Markov-style stochastic prefetching along a dependency graph. ToolCaching (Zhai et al., 2026) fuses request semantics and system features to decide whether results are worth caching and when to evict. Costs in this class are millisecond-level and fully within budget, but the signal derives

entirely from historical statistics, lacking session-semantic modeling and thus unable to distinguish intent-conditioned successors of the same sequence.

External LLM speculators use a separate small LLM to predict the next action while the authoritative agent reasons. Speculative Actions (Ye et al., 2025) issues predicted calls early, reuses results on a match and rolls back on mismatch, guaranteeing trajectories identical to strict serial execution. Speculate-with-Memory (Li et al., 2026) equips the speculator with contrastive transition tables, episodic retrieval, and confusion tracking, so accuracy improves with experience. Such speculators model context well, but each step needs 1–3 s of GPU inference—exceeding the latency they hide—and yields only a few uncalibrated candidates, making low-cost full-candidate ranking and tiered prefetching difficult.

Self-speculation lets the deployed agent itself predict, without extra model instances. SWYR (Ji et al., 2026) trains one model by joint agent–speculator RL and reuses the prefix KV cache during reasoning; SPORK (Bai et al., 2026), DualSpec (Zhong et al., 2026), and SPAgent (Huang et al., 2025) fork decoding streams early or adapt speculation policy to action type. These methods exploit internal model states but generally require white-box serving-engine modification or forked-decoding support, limiting deployment generality.

In sum, existing methods trade off among latency, semantic modeling, and deployability: statistical methods are fast but blind, LLM methods are accurate but slow, self-speculation is strong but heavy. Our CT-GNN breaks the trilemma—it processes conversational context on a global tool graph and explicitly receives transition-statistics features, achieving millisecond-level context-aware prediction without white-box access or GPU inference.

3 METHOD

We present CT-GNN (Conditional-Transition Graph Neural Network) for next-tool-call prediction, together with the global tool graph it runs on and the tiered prefetch protocol that converts ranking results into cache gains. As shown in Figure 1, the session state is encoded into a context vector (sparse BM25 encoding) and a sequence vector (GRU-based sequence modeling); both jointly drive relational graph convolution over the tool graph, whose transition edges carry counting statistics. A retrieval mechanism with structural expansion narrows the candidate set, and a dual-head scoring module fused with a Markov prior ranks the candidates, enabling tiered prefetching.

3.1 PROBLEM FORMULATION

At step t, given the agent session history

$$H_t = (q,\ a_1, r_1, o_1,\ \ldots,\ a_t, r_t, o_t,\ m_t), \tag{1}$$

where q is the user request, a_i the i-th executed tool call, r_i its returned result, o_i the model output text that triggered a_i, and m_t the current user message. The model predicts the set of tools to be invoked within the next k steps, ranks the candidates, and writes the top-ranked ones into a tiered cache. The supervision window k is a per-dataset hyperparameter; its values and sensitivity are reported in §4.2.

3.2 GLOBAL TOOL GRAPH

We build a directed heterogeneous graph $G = (V, E)$ whose node set consists of tool nodes V_{tool} and semantic state nodes V_{sem}, i.e., $V = V_{\text{tool}} \cup V_{\text{sem}}$. Semantic nodes are deduplicated (lowercasing and hashing) from "user request $\|$ step output" texts in the training trajectories, capped at $|V_{\text{sem}}| \le N_{\text{sem}}$ (2000 for τ-bench, 4000 for APIGen), with overflow truncated in first-appearance order. The raw input feature x_j of tool node j concatenates the BM25 vector of "tool name $\|$ tool description" with two statistics $\big[\log(1+\bar{d}_j),\ \log(1+n_j)\big]$, where $\bar{d}_j$ is the tool's mean execution latency and n_j its call count in the training set; a semantic node's feature is the BM25 vector of its text, zero-padded on the statistics dimensions for alignment.

The graph carries three relation types: association edges from semantic nodes to their corresponding tool nodes, forward transition edges between tool nodes, and their reverse edges. Message passing

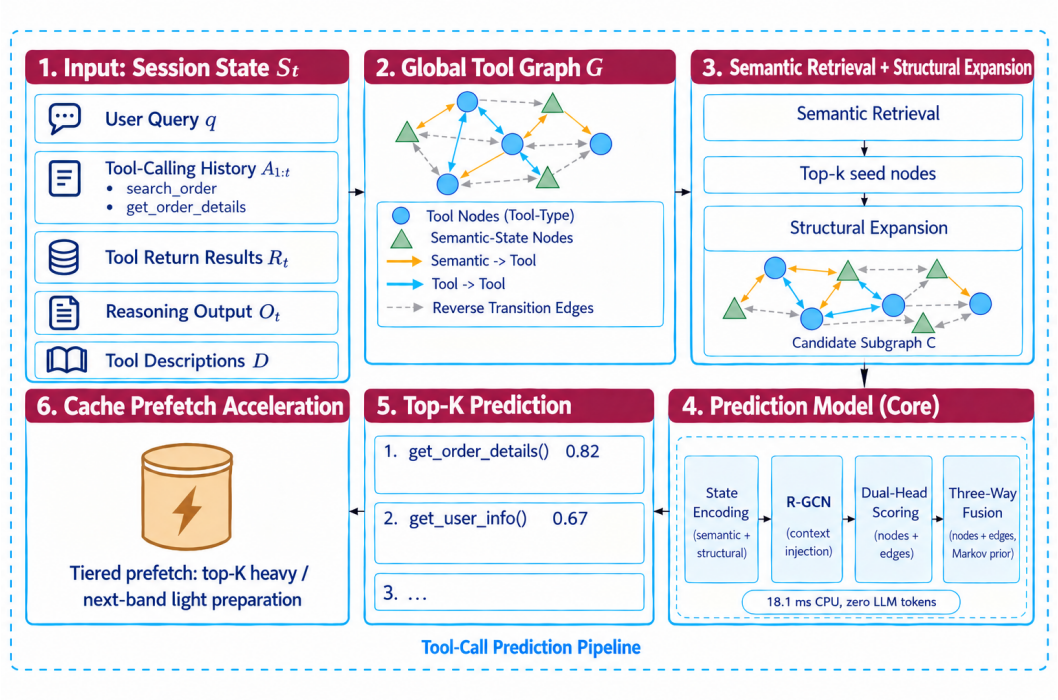

Figure 1: **Pipeline overview.** Conversational state S_t feeds a global tool graph G; semantic retrieval plus structural expansion yields a candidate subgraph; the conditional predictor scores candidates and the top-K tools receive tiered prefetch preparation.

uses relation-specific weight matrices to capture the distinct interaction semantics. Association-edge weights are the per-tool occurrence proportions within the corresponding semantic context; transition edges carry the raw transition probability

$$P(j|i) = \frac{count(i \to j)}{\sum_{j'} count(i \to j')}. \tag{2}$$

All graph statistics (transition counts, tool frequencies, mean latencies) are computed strictly on the training split; an audit confirms the transition counts sum exactly to the number of training steps minus the number of trajectories.

3.3 Conditional-transition predictor

The predictor comprises four parts, detailed in Figure 2: state and sequence encoding, graph encoding with per-layer conditional injection, dual-head scoring, and three-way additive fusion. We describe each below.

State encoding. The conversational state is serialized into a single delimiter-joined text document S_t, containing in order: the user request q, the called-tool name sequence, the model output o_t that triggered the latest call, and, per configuration switches, the current user message m_t, per-step tool results $r_1, \ldots, r_t$ (each truncated to 200 characters), and deduplicated past user messages. A sparse BM25 encoder maps this document to the 256-dim **context vector** $\mathbf{c} = f_{\text{BM25}}(S_t)$: for a term w in the (capacity-256, truncated) vocabulary,

$$c_w = \mathrm{idf}(w) \cdot \frac{\mathrm{tf}(w, S_t)\,(k_1+1)}{\mathrm{tf}(w, S_t) + k_1\left(1 - b + b\,|S_t|/\overline{|S|}\right)}, \qquad \mathrm{idf}(w) = \log\left(1 + \frac{N - \mathrm{df}(w) + 0.5}{\mathrm{df}(w) + 0.5}\right), \tag{3}$$

where tf, df are term and document frequencies, N and $\overline{|S|}$ the corpus document count and mean document length, Okapi parameters $k_1 = 1.5$, $b = 0.75$; $\mathbf{c}$ is not L2-normalized. The vocabulary is

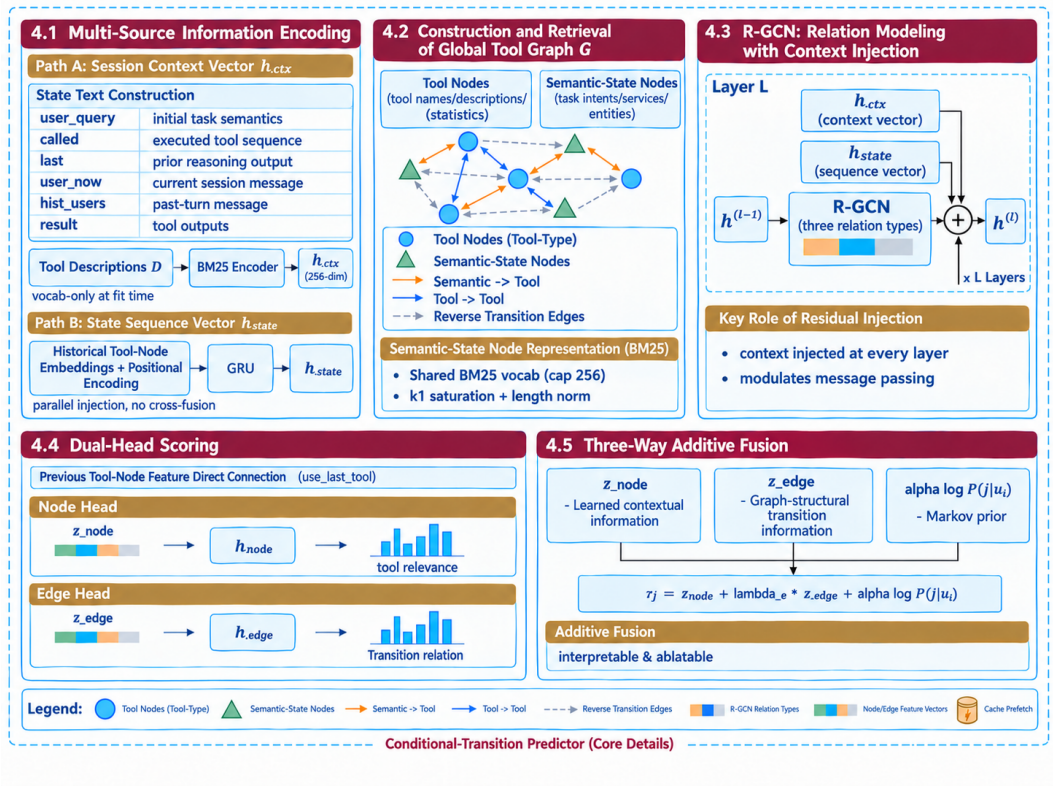

Figure 2: **Core design details.** (4.1) One sparse BM25 encoder maps the state text to the 256-dim context vector (tool descriptions enrich the fit-time vocabulary only); a GRU over position-encoded tool embeddings yields the state vector; the two paths are injected in parallel without pre-fusion. (4.2) Graph construction and subgraph retrieval. (4.3) Per-layer additive injection of both context and sequence vectors into R-GCN. (4.4) Dual-head scoring with a last-tool bypass. (4.5) Three-way additive fusion.

fit only on training-split requests, model outputs, tool results and (if enabled) user messages, including tool names and descriptions—which therefore both widen the encoder's semantic coverage and constitute tool-node input features. Encoding is table lookup plus arithmetic, so inference invokes no LLM at all.

Sequence encoding. The historical tool sequence is encoded by a GRU over the raw input features $x_{a_1}, \ldots, x_{a_t}$ of the called tool nodes; the feature of the ℓ-th tool (ℓ from 0) is concatenated with the normalized position scalar $(\ell+1)/(n-1)$, n being the trajectory length. The final GRU hidden state $\mathbf{h}_{\text{GRU}} \in \mathbb{R}^{128}$ is the dense representation of the sequence.

Graph encoding and conditional injection. We run two layers of relational graph convolution (R-GCN) on G; node embeddings are first projected to a shared 128-dim hidden space, yielding $\mathbf{h}^{(0)}$. The context and sequence signals pass through independent linear layers $\mathbf{W}_c, \mathbf{W}_s$ into the same space, $\widetilde{\mathbf{c}} = \mathbf{W}_c\mathbf{c}$ and $\widetilde{\mathbf{s}} = \mathbf{W}_s\mathbf{h}_{\text{GRU}}$, and are injected additively as a residual into *every* node **after each message-passing layer**: graph convolution only encodes static training-set structure—the current session is not in the graph—so per-layer injection lets nodes keep sensing the session state while updating. For layer m ($m = 1, 2$),

$$\mathbf{h}^{(m)} = \text{Dropout}\Big(\text{ReLU}\Big(\text{R-GCN}_m\Big(\mathbf{h}^{(m-1)}\Big) + \widetilde{\mathbf{c}} + \widetilde{\mathbf{s}}\Big)\Big), \qquad \text{dropout} = 0.2. \tag{4}$$

The two signals are never mixed before addition (independent projections, no early fusion), so textual intent and sequence evolution cannot interfere before the convolutional layers; independent injection also preserves the ablativity of each path.

Division of labor between the two paths. $\widetilde{\mathbf{c}}$ captures the session's *lexical content* (user wording, tool results)—an order-free bag of words; $\widetilde{\mathbf{s}}$ captures the *evolution order* of executed tools (A-then-B $\neq$ B-then-A), to which $\widetilde{\mathbf{c}}$'s tool-name component is exactly invariant, since BM25 depends only on term frequencies and document length. The same tool-name sequence is thus a weighted term set in $\mathbf{c}$ but a position-encoded event stream in $\widetilde{\mathbf{s}}$.

Conditional scoring. Candidate tool j is scored by a node head and an edge head plus the Markov prior. The node head takes a quadruple—the candidate's graph embedding $\mathbf{h}_j^{(2)}$, the context vector $\widetilde{\mathbf{c}}$, the state vector $\widetilde{\mathbf{s}}$, and the raw input feature x_i of the previous tool i as a direct bypass—and outputs the context-dependent tool score $g_{\text{node}}(j)$. The edge head takes the concatenation of the previous-tool and candidate graph embeddings $(\mathbf{h}_i^{(2)}, \mathbf{h}_j^{(2)})$ with the 3-dim transition statistics

$$[\log(1 + \mathit{count}(i \to j)),\ \ P(j|i),\ \ \log(1 + \mathit{count}(j))], \tag{5}$$

outputting the conditional transition score $g_{\text{edge}}(j)$. The final score is a weighted additive fusion with the prior term:

$$\text{score}(j) = g_{\text{node}}(j) + \lambda\, g_{\text{edge}}(j) + \alpha \log P(j|i), \qquad \lambda = 0.3,\ \ \alpha = 0.2. \tag{6}$$

The prior applies uniformly to all tools: for unobserved transitions $P(j|i)$ is floored at 10^{-4}, whose logarithm is a strong negative bias pushing unseen successors into the low-frequency fallback tier. At session start ($t = 0$, no previous tool i), neither the edge head nor the prior applies and scoring is entirely by the node head.

3.4 Training, inference, and prefetch accounting

Training objective. Each training sample is one decision state (a moment in a trajectory). For efficiency and a sharper retrieval signal, the candidate set is generated dynamically rather than taken from the full vocabulary: the unprojected context vector $\mathbf{c}$ (same BM25 space as semantic nodes) retrieves the top-8 semantic nodes by cosine similarity; their tools become seeds, expanded one hop along reverse and two hops along forward transition edges into the candidate subgraph, far smaller than the full vocabulary; the same retrieval serves training and evaluation.

Labels are the tool sets within the next k steps (multi-label); the loss is weighted binary cross-entropy over candidates, positive weight set dynamically from the per-candidate negative/positive ratio (capped at 15), plus a pairwise soft ranking loss $\text{softplus}(\gamma - (z^+ - z^-))$ per positive–negative pair ($z^{\pm}$ the node logits; weight $\beta = 0.2$, margin $\gamma = 0.3$). The optimizer is Adam; learning rate and window size k are per-dataset hyperparameters (see §4.2).

Decision-state sampling differs by dataset but is identical between training and evaluation: τ-bench uses every moment of every trajectory; since APIGen trajectories are longer and evaluation must stay comparable to LLM competitors at affordable scale, we take at most 6 evenly spaced moments per trajectory and cap the test set at 800 trajectories. Both samplings are deterministic.

Model selection and seeds. After each epoch we compute candidate-ranking AUC on the evaluation split, keep the best parameters so far, and early-stop after 5 consecutive non-improving epochs (cap: 32). The two testbeds automatically converge at markedly different epochs, so the training budget is allocated automatically without manual tuning. The data-split seed is decoupled from training randomness: fixed at 42 so all compared methods share one test set; three training seeds (42/43/44) reflect only initialization and optimization variability.

Inference cost. CPU ranking is millisecond-scale: 18.1 ms per step (0.9 ms BM25 + 7.9 ms subgraph retrieval + 9.3 ms GNN forward, measured on τ-bench; APIGen's larger subgraphs remain in the same order of magnitude)—roughly two orders of magnitude below the typical tool-execution budget (our cache simulator's latency model is calibrated within the measured 200–800 ms range).

Tiered prefetch and cost accounting. Ranked candidates are prepared by tier: the top k_{high} tools receive *heavy* prefetch (connection establishment, argument scaffolding, sandbox launch), eliminating 60% of execution latency on a hit; tools ranked between k_{high} and k_{mid} receive *light* preparation, saving 20%. A wrong heavy-prefetch target incurs a 50 ms penalty; light preparation incurs none. Unless stated otherwise, all speedups use $(k_{\text{high}}, k_{\text{mid}}] = (1, 3]$: at this tier a heavy-prefetch hit

Table 1: Dataset statistics. The "Tools" column lists the dataset's raw tool-name count / our modeled vocabulary size; identical values indicate no truncation.

Dataset	Trajectories	Tools	Avg. steps	Regime
ToolBench-30k	30,000	11,100 / 300	3.2	short-context
APIGen-MT	4,894	106 / 92[†]	4.3	context-conditioned
τ-bench retail (GPT-4o)	451	16	7.2	multi-turn
τ-bench retail (Claude 3.5)	919	16	7.7	multi-turn
BUTTONInstruct	7,231	13,134 / 298	3.4	long-tail

Table 2: Prediction accuracy and acceleration. Ours: unified recipe, 3 seeds (mean$\pm$std), speedup at budget $(1, 3]$. Competitor rows are our reruns on the identical testbed, each at its strongest configuration.

Method	τ-retail (GPT-4o)	τ-retail (Claude 3.5)	APIGen-MT	Cost/step
Markov	47.29% / 1.23×	49.89% / 1.46×	39.33% / 1.34×	0.1 ms
PASTE-style k-gram	49.8% / 1.25×	56.2% / 1.55×	44.7% / 1.54×	ms-level
SpecBox (Laplace)	57.0% / 1.32×	56.4% / 1.49×	47.0% / 1.36×	ms-level
SWYR zero-shot	41.56% / 1.27×	43.85% / 1.24×	34.87% / 1.24×	1.6–2.0 s
SWYR + full history	44.11% / 1.29×	45.54% / 1.28×	36.21% / 1.29×	1.6–2.0 s
SwM	71.34% / 1.65×	71.61% / 1.80×	58.45% / 1.48×	2.0–2.9 s
SwM + full history	68.47% / 1.59×	71.19% / 1.79×	59.85% / 1.52×	2.0–2.9 s
SA (strict) [†]	48.4% / 1.38×	55.7% / 1.43×	—	3.7–4.7 s
GTool [‡]	—	—	52.4% / 1.28×	2.02 s/traj
Ours	**80.20$\pm$1.61% / 1.89×**	**84.89$\pm$0.49% / 2.09×**	**65.39$\pm$10.17% / 1.79×**	**18.1 ms**

is exactly tool-name-level top-1, so the efficiency metric coincides with the accuracy metric and compares fairly against competitors' single-shot guessing.

4 Experiments

4.1 Datasets and evaluation setup

To cover different degrees of context dependence, we select datasets that differ markedly in trajectory length, tool-space size, and context conditioning. ToolBench (Qin et al., 2024), the largest public benchmark, has short trajectories whose next call is weakly tied to context—the near-Markovian "short-context" extreme. APIGen-MT (Prabhakar et al., 2025) generates multi-turn trajectories through simulated human–agent interplay, where tool choice is jointly determined by query, business policies, and dialogue state—the context-conditioned regime. Speculative Actions (Ye et al., 2025) provides real multi-turn τ-bench trajectories with competitor prediction files, the basis of our controlled evaluation. BUTTONInstruct (Chen et al., 2025) probes long-tail behavior when the tool space vastly exceeds modeling capacity (13k tools truncated to at most 300). The combination spans from short context to long trajectories and from narrow to extremely wide vocabularies (Table 1).

[†] The APIGen vocabulary includes the pseudo-tool `think` (introduced by the tool-description files, never called in trajectories) and two parse-noise names occurring once; ToolBench and BUTTONInstruct vocabularies are truncated to at most 300 tools by call frequency (BUTTONInstruct retains 298 distinct names in the training split).

4.2 Prediction accuracy and speedup

[†] Strict SA fires only at turn-start states (376/689 steps) under a double-blind setting (no current user message); the two cells use this subset protocol. [‡] GTool plans once per trajectory (extra $+2.02$ s); its speedup waives inference cost at $(1, 3]$; Table 4 gives the $(2, 5]$ breakdown (§4.4).

Table 2 shows advantages along three dimensions. On accuracy, ours leads top-1 on all three testbeds. On efficiency, LLM-family speedups (1.24–$1.80\times$) fall below ours (1.79–$2.09\times$). Against the millisecond peers, k-gram context and Laplace smoothing are necessary upgrades over plain Markov, yet both still trail us by 18–30 pp; their failure modes are complementary (pattern mining

Table 3: Ranking comparison inside the same restricted candidate set (final recipe, three-seed mean).

Testbed	Markov	Ours (R-GCN)	Gap
APIGen-MT	69.50%	94.38%	$+\mathbf{24.9}$ pp
τ-bench (GPT-4o)	61.36%	82.09%	$+\mathbf{20.7}$ pp
τ-bench (Claude 3.5)	62.50%	88.06%	$+\mathbf{25.6}$ pp

Table 4: GTool planner evaluated as a prefetcher.

Method	hit-high	hit-mid	Speedup
GTool	53.8%	0.0%	1.278×
Markov	57.4%	30.7%	1.603×
Ours (R-GCN)	**77.5%**	20.5%	**1.934**×
Oracle	88.6%	11.4%	2.164×

wins the mid-band, smoothed counting wins top-1), and our learned fusion absorbs both signals, sweeping accuracy at millisecond cost.

Per-dataset hyperparameter adaptation is load-bearing, and we quantify it. The unified recipe shares encoder architecture, feature set, graph structure, and early-stopping rule, but adapts the supervision window k and learning rate (τ-bench: $k = 1$, lr $= 5 \times 10^{-4}$; APIGen: $k = 3$, lr $= 2 \times 10^{-3}$). Sensitivity runs show the adaptation is not optional: on τ-bench (GPT-4o), switching to APIGen's values drops three-seed top-1 to $62.79 \pm 1.12\%$ ($k = 3$, -17.4 pp) and $70.91 \pm 2.35\%$ (lr $= 2 \times 10^{-3}$, -9.3 pp), far below the unified recipe's $80.20 \pm 1.61\%$.

4.3 Boundary conditions: context-conditioned vs. near-Markovian tasks

Ranking inside an identical candidate set. To decouple ranking from retrieval, we confine both sides to a common candidate set: the top-5 successors of the previous tool i under training-set transition counting (ties broken lexicographically by tool name, so results are environment-independent); each method merely re-ranks within this set. States whose true successor lies outside the set are dropped (retained: APIGen 2413/2612, τ-GPT-4o 484/538, τ-Claude 1136/1240). Table 3 reports the three-seed mean in-set top-1 of the final recipe.

Even inside the candidate set circumscribed by Markov—where the previous tool has already collapsed the search space—the GNN ranks successors more precisely by exploiting conversational semantics: under context-conditioned transitions, counting alone is far from Bayes-optimal. Conversely, on ToolBench-30k (mean 3.2 steps, weak context dependence), the base GNN without transition-statistics features trails Markov by 8.0 ppin-set; feeding statistics to the edge head reverses the gap to about $+1$ pp(five-way breakdown in Table 5).

The advantage distribution (APIGen final recipe, three-seed mean) shows gains concentrate where counting cannot disambiguate: on the 1,204 states whose true successor has count ≥ 11 we lead by 42.7 pp(94.2% vs 51.5%), and on the 34 states with counts 4–10 by 27.4 pp. The same transition points to different successors under different intents—where counting is blind but context resolves.

4.4 Planners as prefetchers: reverse evaluation of GTool

We assess the efficacy boundary of an LLM-based planner (GTool) inside our prefetch framework: we train GTool on APIGen, feed its predictions into the simulator as prefetch hints, and replay on the same test split under the wide $(2, 5]$ budget.

GTool performs poorly in our online-prefetch setting, far below lightweight predictors, owing to a structural mechanism mismatch: one-shot sequential planning suffers error cascades—an early mistake shifts the hint context of every later step, so mid-band ranks almost never hit—and long trajectories hit the LLM's token-budget truncation. Online prefetching instead requires consuming the true execution history at every step and correcting in real time, not committing to a whole call sequence before any environment feedback arrives.

Table 5: Architecture ablation (ToolBench-30k, in-set top-1 as in §4.3).

Configuration	In-set top-1	vs. Markov
Node head only (edge head off)	36.0%	-19.8 pp
Edge head on, no transition statistics	47.9%	-8.0 pp
Edge head + Markov prior (no statistics)	53.7%	-2.1 pp
Edge head + transition statistics	56.8%	$+\mathbf{1.0}$ pp
Edge head + statistics + prior (final)	**57.0%**	$+\mathbf{1.2}$ pp

Table 6: State information-source ablation ladder (τ-bench GPT-4o, top-1 at budget $(1, 3]$; each row is an independent configuration, additions in parentheses). Rows 4–5 are single-item TF-IDF-side gains (user message $+2.6$ pp, tool descriptions $+1.8$ ppover the preceding row); they branch off row 3's BM25 line and cannot be summed linearly—rows 6 onward are BM25-side full-feature combinations.

Configuration	Top-1
TF-IDF baseline (request + tool-name sequence + last output)	57.48%
+ tool results (truncated to 200 chars)	60.03%
+ encoder switched to BM25	62.10%
+ current user message (TF-IDF-side branch point)	64.65%
+ tool descriptions (TF-IDF-side branch point)	66.40%
BM25 + all features (results + message + descriptions)	74.20%
+ user-message vocabulary repair	74.84%
+ automatic early stopping (final recipe)	**80.20±1.61%**

4.5 Ablation studies

We ablate architectural components (Table 5) and state information sources (Table 6).

The Markov prior gives a stable fallback (gap improving from -8.0 ppto -2.1 pp) but limited gain; explicit transition-statistics features are what teach the edge head to lean on counting when necessary and overtake it ($+1.0$ pp). The full model attains the best 57.0%, 1.2 ppabove Markov.

The result-truncation length follows an inverted-U on both configurations (GPT-4o + BM25, Claude 3.5 + TF-IDF): 100/200/300/400 characters yield $70.5/74.2/71.5/72.8\%$ for the former and $79.1/74.5\%$ at 200/300 for the latter—both peak at 200. Overly long JSON dilutes the state vector, crowds out informative terms in the 256-dim vocabulary, and incurs BM25's length penalty.

5 Conclusion

Our conditional-transition GNN predictor ranks all candidates at 18.1 ms per step on CPU with zero LLM tokens, attaining $80.20\pm1.61\%$, $84.89\pm0.49\%$, and $65.39\pm10.17\%$ top-1 on three public trajectory scenarios and delivering 1.79–$2.09\times$ cache speedup. The experiments further delineate the method's applicability boundaries and underlying regularities: in context-conditioned scenarios, learned prediction leads counting by 20.7–25.6 pp within restricted candidate sets; in near-Markovian scenarios, counting already approaches the Bayes-optimal level, and only explicit transition-statistics features achieve the overtake. The GTool reverse evaluation verifies, at the mechanism level, the irreplaceability of online step-by-step prediction: one-shot planning cannot convert planning quality into prefetch benefit even with inference cost fully waived. Our method remains non-inferior to counting in both regimes, eliminating the need for prior task-type discrimination at deployment.

This study has limitations. The seed variance on APIGen indicates that the current training scale is insufficient to stably capture all context-conditioned dependencies; and the speedup evaluation rests on a unified synthetic latency model, still requiring calibration with measured latency distributions and safety policies on real tool-serving infrastructure.

REFERENCES

Huajun Bai, Weiwei Lv, Huichuan Zheng, Youyou Lu, and Jiwu Shu. SPORK: Self-speculative forking to accelerate agentic LLM inference. *arXiv preprint arXiv:2607.03333*, 2026.

Mingyang Chen, Haoze Sun, Tianpeng Li, et al. Facilitating multi-turn function calling for LLMs via compositional instruction tuning. In *International Conference on Learning Representations (ICLR)*, 2025.

Wenjie Chen, Wenbin Li, Di Yao, Xuying Meng, Chang Gong, and Jingping Bi. GTool: Graph enhanced tool planning with large language model. In *International Conference on Learning Representations (ICLR)*, 2026.

Zixiao Huang, Wen Zeng, Tianyu Fu, et al. Reducing latency of LLM search agent via speculation-based algorithm-system co-design. *arXiv preprint arXiv:2511.20048*, 2025.

Jiabao Ji, Yujian Liu, Li An, Rohit Jain, Gungor Polatkan, Siyu Zhu, and Shiyu Chang. Speculate while you reason: Teaching agents to predict their next tool call via joint agent-speculator RL. *arXiv preprint arXiv:2607.25816*, 2026.

Yu Li, Qinyuan Ye, Prafulla Kumar Choubey, Jiaxin Zhang, and Chien-Sheng Wu. Speculate with memory: Lossless acceleration for LLM agents. *arXiv preprint arXiv:2607.12236*, 2026.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, et al. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, and Yuqing Yang. Parallelizing tool execution and LLM generation for low-latency agent serving. *arXiv preprint arXiv:2603.18897*, 2026.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and Dongsheng Li. Can graph learning improve planning in LLM-based agents? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster agentic systems. *arXiv preprint arXiv:2510.04371*, 2025.

Yi Zhai, Dian Shen, Junzhou Luo, and Bin Yang. ToolCaching: Towards efficient caching for LLM tool-calling. *arXiv preprint arXiv:2601.15335*, 2026.

Yihui Zhang, Tianyu Wo, Jinghao Wang, Xiaoyang Sun, et al. SpecBox: Speculative sandbox scheduling for efficient LLM agent serving. *arXiv preprint arXiv:2607.23933*, 2026.

Shuzhang Zhong, Baotong Lu, Qi Chen, Chuanjie Liu, Fan Yang, and Meng Li. DualSpec: Accelerating deep research agents via dual-process action speculation. *arXiv preprint arXiv:2603.07416*, 2026.

AI USE STATEMENT

Large language models were used in three capacities, all assistive: (i) polishing and editing prose, including translation and wording revisions of this manuscript; (ii) drafting and revising individual sections and the overview figure (an AI image tool produced figure drafts from author-specified content, which the authors then verified against the code and corrected over multiple iterations); and (iii) assisting code debugging and writing analysis scripts during experimentation. All scientific claims, experimental design, data collection, and quantitative results were produced and independently verified by the authors; no LLM generated any experimental number or conclusion.