Speculative Context: A Pool-Based Cache for SOP-Constrained Voice Agents

Dima (Dan) Sivov
dsivov@gmail.com

June 2026 · Revision 2 (post-review)

Abstract. Real-time voice agents face a latency–intelligence trade-off: a small fast model holds the conversational budget but cannot reason through multi-step procedures or retrieve documents, while a large model can but leaves the user waiting—above all on slow external data. We present a two-system architecture for voice agents under a declared Standard Operating Procedure (SOP): a live agent reads from a session-scoped knowledge *pool* that an off-path *supervisor* populates speculatively during the user's speech-time, moving external-data retrieval off the critical path. We evaluate three components. (i) A **speculatively-populated pool cache**: because mispredicted fetches remain usable pool candidates rather than discarded cache misses, the live agent finds a prefetched item on 96% of *turns* under per-turn curation, while only 11% of speculative *fetches* are consumed under exact-key lookup—two measurements of different quantities (turns vs. fetches), not a single rate. Across our sessions the pool hides a mean of 22.5 s of external-data latency per conversation. (ii) A **cosine-plus-dedup rerank** that curates the pool with no LLM on the critical path; on a matched comparison it reproduces an LLM reranker's curation while cutting p95 rerank latency from 2167 ms to 940 ms (299 ms mean / 483 ms p95 in the locked cross-SOP configuration). (iii) A **speculative-context prompt**: an ablation shows that labelling prefetched items honestly as speculative—rather than "curated, high-relevance"—keeps the agent from over-using low-relevance items on conversation-closing turns. This is a *pilot* validation of the architecture's mechanics and internal consistency—N=5 per cell, simulator users, internal baselines—not a production-performance claim; §8 states the larger-N, common-metric, and human-validation studies that would establish the latter. We also report the end-to-end picture: the rerank adds only ∼300 ms, while total turn time (∼6 s) is dominated by the live agent's own two LLM calls—which a smaller model does not reduce (§5.4)—so this work targets external-data latency, not LLM-inference latency.

1 Introduction

A real-time voice agent makes a choice on every turn that has no good answer under a single model. A small fast model holds the real-time budget but fumbles whenever it must follow a multi-step procedure or retrieve a document; a large model is competent but leaves five to fifteen seconds of silence between utterances, and users hang up. The standard production compromise—a medium model with a thin retrieval layer and a router that escalates hard cases—works for simple flows but not for the regulated, multi-step conversations (insurance renewal, medical scheduling, account servicing) that businesses actually need handled.

We treat the trade-off as a *two-system* problem rather than a one-model problem. A fast live agent responds in real time, reading from a shared blackboard. A slow supervisor runs in the background between user utterances—during the user's own speech-time—predicting what the user will need next and pre-staging it. When the supervisor is right, the response feels instant; when wrong, the live agent falls back to its built-in behaviour, no worse than today. Framed quantitatively, the supervisor solves a budgeted prediction problem: choose, within a fixed speculative-compute budget, which upcoming data dependencies to prefetch so as to minimise the critical-path latency the live agent pays—the *regret* against a perfect-foresight oracle (§2). The general orientation (use the user's speech-time for slow work) appears in AsyncMLD [5] for dialogue recommendation; our contribution is the specific design that makes it work for SOP-constrained voice dialogue.

Contributions.

1. A **speculatively-populated pool cache** (§5.1). Rather than predict an exact cache key, the supervisor appends each speculatively-fetched item to a session-scoped pool; the live agent curates it per turn. Because mispredictions persist as pool candidates instead of being discarded, per-turn curation supplies a prefetched item on 96% of turns, whereas an exact-key cache over the same speculative fetches consumes only 11% of them (two different denominators, reconciled in §5.1). The novel element is the misprediction-tolerant *population* policy over an SOP graph, not the dual-agent prefetch-cache pattern itself (§2).
2. **An empirical case that the per-turn rerank needs no LLM** (§3.5, §6.1). Our curation is cosine ranking plus a thresholded redundancy filter (a form of MMR [12]); the contribution is the measurement that this recovers an LLM reranker's *only* value-add on this data—near-duplicate avoidance—while removing it from the critical path. On a matched comparison this cuts p95 rerank latency from 2167 ms to 940 ms with no loss of curation quality.
3. **A framing-mismatch ablation for speculative**

context (§3.6, §6.2). That LLMs degrade on irrelevant context and that an "ignore irrelevant information" instruction mitigates it is known [13]; what we add is an ablation in an asynchronous prefetch pipeline showing that an over-confident "curated" label (vs. an honest "speculative" one) makes the agent overuse low-relevance items on conversation-closing turns, lowering task success.

We evaluate on two SOPs (outbound car-insurance renewal, inbound medical-appointment booking) using a smart-LLM customer simulator. The present study is deliberately scoped—N=5 per cell, simulator users, internal baselines—and §8 states the controlled experiments (larger N with confidence intervals, published-baseline comparisons, a human study via a live-avatar harness we have built) that complete it.

2 Related work

Voice agents with speech-time speculative prefetch (closest work). The dual-agent shape we adopt—a background agent that prefetches during the user's speech and a foreground agent that reads from the resulting cache—has been explored concurrently in real-time voice systems. Closest is VoiceAgentRAG [1], whose background "Slow Thinker" predicts likely follow-up topics with an LLM and prefetches document chunks into a semantic cache that a foreground "Fast Talker" reads on cache hits; LTS-VoiceAgent [2] ("listen–think–speak") and RelayS2S [3] similarly overlap background reasoning with foreground speech, and predictive-ASR prefetch [4] hides downstream latency by acting on a preliminary ASR hypothesis. We do not claim the dual-agent speech-time-prefetch pattern itself as novel. We differ in two respects that we evaluate: (i) our supervisor predicts over a *structured SOP graph* using empirical transition statistics (with optional MCTS lookahead), rather than free-form topic prediction; and (ii) our pool is *misprediction-tolerant*—a fetch triggered by a wrong prediction remains a curated candidate on later turns instead of becoming a wasted cache entry (§5.1).

Asynchronous LLM work during dialogue. AsyncMLD [5] reduces dialogue-system latency by running database searches and intention understanding in parallel with the agent's spoken response. We share the orientation—do slow work during the user's speech-time—but differ in mechanism: rather than dispatching parallel work for the *current* query, our supervisor maintains a session-scoped pool of speculatively prefetched items, curated per turn for the *next* response.

Planning-based dialogue and speculative tool execution. ChatSOP [6] introduces Monte Carlo Tree Search (MCTS) over an SOP graph for controllable LLM dialogue. Our supervisor's planner is of this family; the architectural delta is that planning runs off the critical path, contributing predictions to the pool rather than gating the live reply. Most directly related to our prefetch mechanism is PASTE [7], which speculatively pre-executes tools during an agent's reasoning time using patterns mined from execution traces—and finds, as we do, that trace-mined prediction outperforms search-based lookahead for deciding what to fetch; our ablation (§6.3) reaches the same conclusion against MCTS for dialogue data prefetch. We differ in slack window (the user's speech-time vs. the LLM's reasoning-time) and in misprediction handling: PASTE discards mispredictions on a bounded budget, whereas our pool *retains* them as reusable candidates.

Bandits, regret, and budgeted prefetch. Choosing *which* upcoming data dependencies to prefetch under a fixed speculative budget is a budgeted, contextual multi-armed bandit [15, 16, 17, 18]: each candidate dependency is an arm with uncertain payoff (latency saved on a pool hit), the budget caps the pulls per turn, the context is (cohort, state, mood, history), and the objective is to minimise *regret*—the latency a perfect-foresight oracle would have saved but we did not (see [19] for a survey). Our system occupies the exploit-only corner of this space: the empirical predictor prefetches the historically best-paying arms and never explores. We add the two adaptation mechanisms a bandit framing motivates—recency decay (for non-stationarity) and shrinkage toward the SOP marginal (for cold start)—but *not* exploration, and §6.4 shows by a held-out temporal study why: the setting has full-information feedback (the next action is revealed every turn regardless of what we prefetched), so each turn is a free labelled example and exploration is empirically inert. The prefetch problem is thus better read as online supervised learning under a budget than as a true bandit; the bandit vocabulary still names the objective (budgeted regret, measured at 4.5% in §5.2). The remaining bandit-flavoured open direction is parametric context generalisation (LinUCB-style sharing across similar unseen (cohort, state) cells), not exploration (§8).

Caching and reranking. Similarity-based caches such as GPTCache [9] rescue near-miss lookups that exact-key caches discard—the premise our pool builds on, but applied to *speculatively prefetched data* rather than reactively cached responses. Cache-augmented generation [10] and session-scoped vector retrieval reuse accumulated context without re-paying per turn; agentic plan caching [11] reuses cached *plans* across similar tasks (we reuse prefetched *data*). Our per-turn curation is deliberately lightweight: the dedup step—cosine ranking plus cosine-threshold redundancy removal—is a thresholded form of Maximal Marginal Relevance [12], and the empirical contribution of §6.1 is the measurement that this recovers an LLM reranker's only value-add (near-duplicate avoidance) at a fraction of its latency. Spec-

ulative decoding [14] bets on a likely future at the token level and falls back when wrong; our "speculative" is at the data/context layer, but the speculate-then-verify heritage is the same. We also distinguish our usage from Speculative RAG [8], where "speculative" denotes small-model *draft*-and-verify at answer time; ours denotes *prefetch* during the user's speech-time, a different mechanism and layer.

Irrelevant-context robustness. That LLMs degrade on irrelevant context—and that a simple instruction to ignore it mitigates the effect—was shown by Shi et al. [13]. Our speculative-context prompt (§6.2) applies this in an asynchronous prefetch pipeline; what we add is the *framing-mismatch* ablation (an over-confident "curated" label vs. an honest "speculative" one) and the closing-turn failure mode it exposes.

3 Architecture

The live voice agent (`gpt-4o` in our implementation) reads the shared blackboard—a per-session pool of prefetched items, each tagged with provenance and a TTL. In the common case it does not block on external I/O, serving data dependencies from the pool; it falls back to a blocking live fetch only on the rare turn where a needed dependency was not prefetched (9 such fetches in the runs of §5.2). All other slow work—retrieval, planning, reasoning over the conversation graph—is done off-path by the supervisor (Figure 1).

3.1 SOP structure

The conversation is structured by a declared SOP: a directed graph whose nodes are *agent actions* (e.g. `VerifyIdentity`, `PitchDiscount`) and *user states* (e.g. `PriceConcern`, `AgreedToRenew`), with edges encoding allowed transitions. Each agent action declares its external *data dependencies* (a policy lookup, a market-rates query, a discounts retrieval). Cohorts and per-cohort moods describe the user's archetype and disposition and shape both the supervisor's predictions and the live agent's style (Figure 2). The supervisor plans against this graph: from the current state it predicts the next 1–3 likely user states, the agent actions that would follow, and prefetches those actions' data dependencies.

3.2 The supervisor's predictor stack

Two predictors populate the pool, plus a fallback. An **empirical predictor** (the workhorse) looks up accumulated `precedent_traces` for the action distribution that historically follows the current (cohort, state, action), and schedules prefetches for those actions' dependencies; it is cheap, deterministic, and improves as deployment data accumulates. It applies recency decay (to track drift) and shrinkage toward the SOP-level action marginal (for sparse cells), both quantified in §6.4. **Background MCTS pondering** (§3.3) is a more far-sighted predictor: it runs speculative tree search over the SOP during the user's speech-time. We retain it in the architecture but show by ablation (§6.3) that it does not earn its cost on the retrieval task; the generative part of retrieval is instead supplied by a single cheap LLM call (§6.3). Distinct from these is a **blocking live fallback**: when a needed dependency was *not* prefetched, the live agent issues it synchronously on the critical path (these are misses, not predictions, counted separately in §5.2). In our measured configuration the empirical predictor supplies the large majority of pool items.

3.3 Background MCTS pondering

Between turns the supervisor does not yet know what the user will say—but the SOP sharply constrains the possibilities. The next user state is drawn from a small vocabulary, and the empirical transition statistics make some states far more likely than others. Pondering exploits this idle window. When a turn ends, the supervisor (i) predicts the top-K most likely next user states from the empirical transition frequencies over accumulated precedent traces, and (ii) spawns one Monte Carlo Tree Search per hypothesized state, all running in the background while the user is still speaking (Figure 3). None of this is on the critical path; if the next turn arrives first, the searches are simply cancelled.

Each search is an MCTS over the SOP action graph (depth 3 in our configuration), with one twist that matters for prefetch diversity: rollouts are *mood-diverse*. Each cohort carries a small vocabulary of moods—dispositions such as `price_anxious` or `calm_negotiator`—with prior probabilities. At the start of each rollout the simulated user's mood is sampled from that prior and held fixed for the rollout. Parallel rollouts under the same candidate action therefore explore genuinely different user dispositions, which react differently to the same agent move, and the action's value is the reward averaged across those mood draws (Figure 4). Without mood diversity, rollouts collapse onto a single "average" user and the predicted next-state distribution is artificially narrow; mood sampling widens it, which in turn diversifies what the supervisor prefetches.

A completed pondering search yields two payoffs, both feeding the live turn that follows. First, a **cached action decision**, keyed by (cohort, predicted state): if the user's actual classified state at turn $N{+}1$ matches a hypothesis, the live agent reuses the cached search and skips live planning entirely. Second, **prefetch seeds**: each rollout generates the natural-language utterance the simulated user would plausibly say, and these predicted utterances drive the supervisor's query-aware data prefetch—populating the pool with data for turns that have not happened yet.

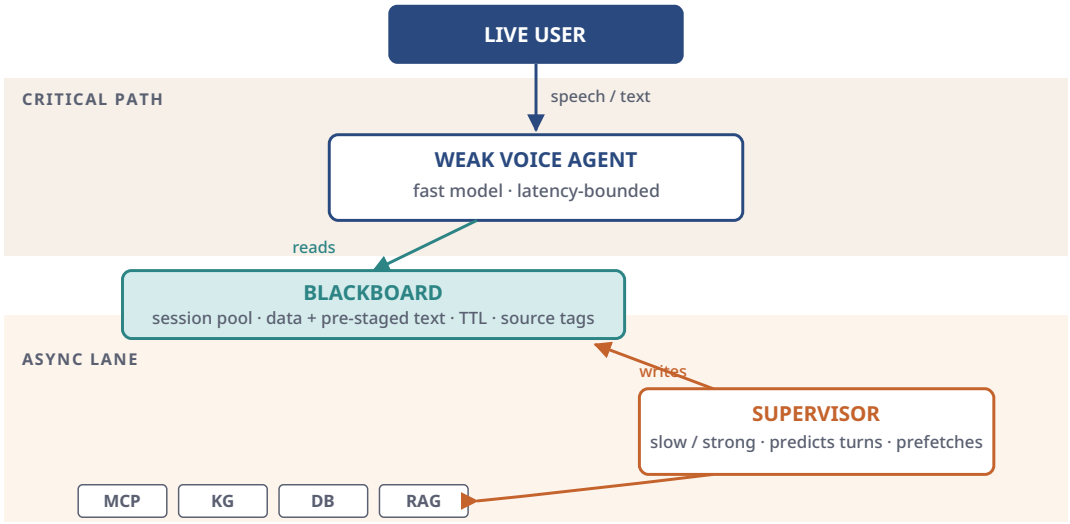

Figure 1. The two-system architecture. The live voice agent talks only to the shared blackboard, never to the supervisor or data sources directly. The supervisor runs slow work (search, retrieval, planning) during the gap between user utterances and writes results to the blackboard.

The second payoff is the one that matters most for our pool, since it converts lookahead into concrete prefetched context rather than only a cached decision.

Pondering is, by design, the mechanism by which the system's planning improves with reflection time—but it is also the most expensive predictor, and §6.3 shows by ablation that for the retrieval task it does *not* earn its cost: a cheap empirical-count predictor predicts the next action far more accurately, and a single small-model call fills the generative query slot at ∼85× less cost. We therefore retire pondering from the retrieval path in our recommended configuration, retaining it only as a cold-start option and for its cached-decision payoff (both deferred to future work). We describe it here because it is the architecture's natural lookahead component and the baseline the ablation is measured against.

3.4 The pool

The pool is a per-session set of items, each carrying a payload, a short text summary, source tags, a pre-computed summary embedding, a TTL, and a confidence score. It is capped at 30 items per session (lowest-confidence evicted on overflow). The summary embedding is computed once at insert time (OpenAI `text-embedding-3-small`), so per-turn curation pays only for cosine arithmetic plus a single embedding of the live message.

3.5 Per-turn curation: cosine rerank with deduplication

At each turn the live agent curates the pool with *no LLM call*: (1) embed the live user message and rank pool items by cosine similarity; (2) a dedup pass drops items by `(dependency_name, summary[:60])` exact-key match and then by pairwise summary-embedding cosine ≥ 0.95; (3) take the top-3 survivors and attach them to the live agent's response prompt. §6.1 shows by ablation that this matches an LLM rerank's curation while removing it from the critical path, and §6.1 justifies the dedup pass specifically as recovering the only discrimination the LLM provided.

3.6 The speculative-context prompt

The attached items are handed to the live agent under a prompt that states their speculative nature explicitly:

```
PREFETCHED CONTEXT (speculatively pre-staged --
may or may not fit this turn):
The supervisor pre-fetched these items based on
predictions about what the user might ask. They
may help your reply, or they may be off-topic --
especially when the user is wrapping up, making
small talk, or asking about something
unanticipated. Evaluate each item against what
the user just said. Use what is relevant; ignore
what isn't. If nothing fits the current message,
reply naturally as if no context was provided --
do NOT force irrelevant data into the response.
```

This is a design component, not a cosmetic choice: §6.2 shows that the natural alternative—labelling the block "curated by supervisor, high relevance"—measurably lowers task success, because the cosine rerank (unlike an LLM) cannot decline to return items on low-relevance turns, and an over-confident label leads the agent to use them anyway.

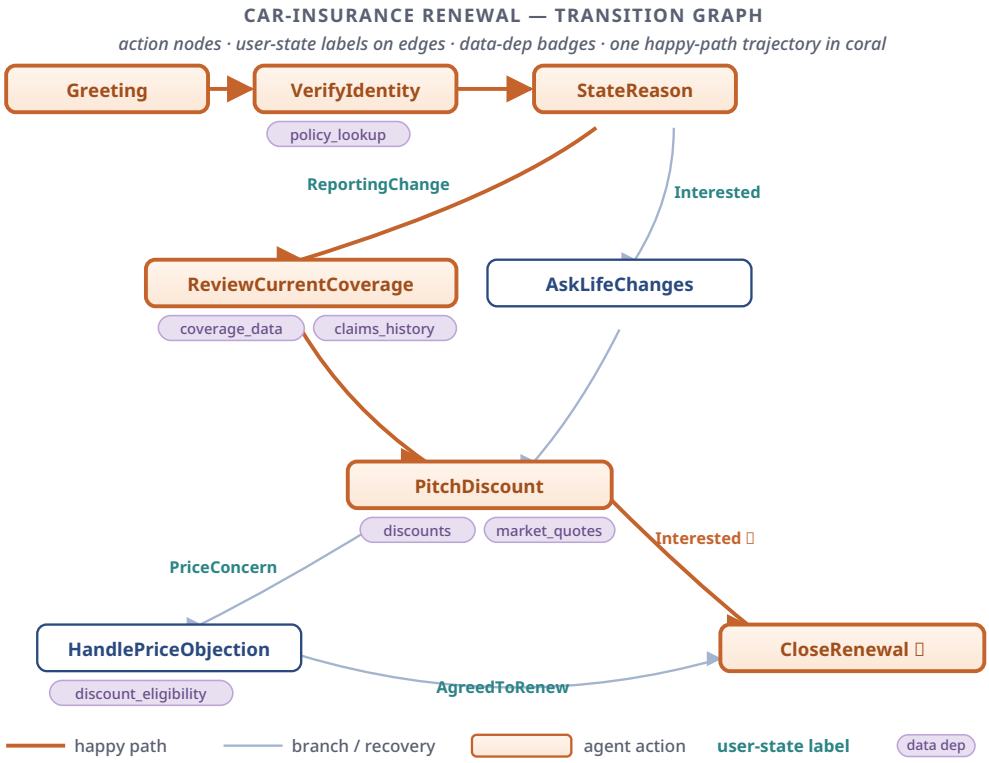

Figure 2. The car-insurance renewal SOP, with a representative happy-path trajectory (coral) and off-trajectory recovery branches (grey). User-state labels sit on edges; data-dependency badges sit under the action that prefetches them. The full SOP has 13 actions, 11 user states, and 7 cohorts × 3–4 moods.

4 Evaluation methodology

Scenarios. Two SOPs of contrasting shape: outbound car-insurance renewal (13 actions, 11 user states; the agent drives) and inbound medical-appointment booking (14 actions, 13 user states; the user drives). Each declares 7 cohorts with 3–4 moods and 6 data dependencies with realistic mocked latencies (0.35–6.5 s).

Users. A strong LLM plays the customer (the "smart simulator"). This is faithful for prediction quality, hit rate, latency-hidden, and tier distribution; it is not faithful for prosody, ASR latency, barge-in, or real-user behaviour. We treat real-call validation as out of scope for the present study and report a concrete plan for it in §8.

Metrics. *Turn supply rate* (the pool's "effective hit rate"): the fraction of *turns* on which curation returns at least one prefetched item (vs. a live fallback fetch). *Fetch-consumption rate* (the exact-key baseline's analogue): the fraction of issued speculative *fetches* ever consumed. These have different denominators (turns vs. fetches) and we never combine them into one number. *Rerank latency*: wall-clock of the per-turn curation step, mean and percentiles. *Session success*: whether the conversation reaches the SOP's success-marker state. *Predictor composition*: the share of pool items contributed by each source.

Scale and statistics. The present results use N=5 sessions per cell with pre-committed thresholds for pass/fail, which we use for coarse-grain confirmation rather than effect-size estimation. §8 specifies the larger-N study (with Wilson/bootstrap confidence intervals) and the published-baseline comparisons required to make each headline number defensible.

5 Results

5.1 The pool cache (hit rate)

When the needed data depends on what the user just said—RAG over policy documents or claims history—exact key prediction is unreliable: even good predictions are only approximately the live query. In an earlier measurement, 60% of predicted RAG queries shared at least one retrieved document with the eventual live query—a strong but inexact signal that an exact-key cache discards as a miss. The pool keeps that signal: each fetched item is appended to the session pool, and curation matches against it per turn rather than against a predicted key (Figure 5).

Two measurements on *different denominators* bound the effect on car-insurance renewal; we report both explicitly rather than as one ratio.

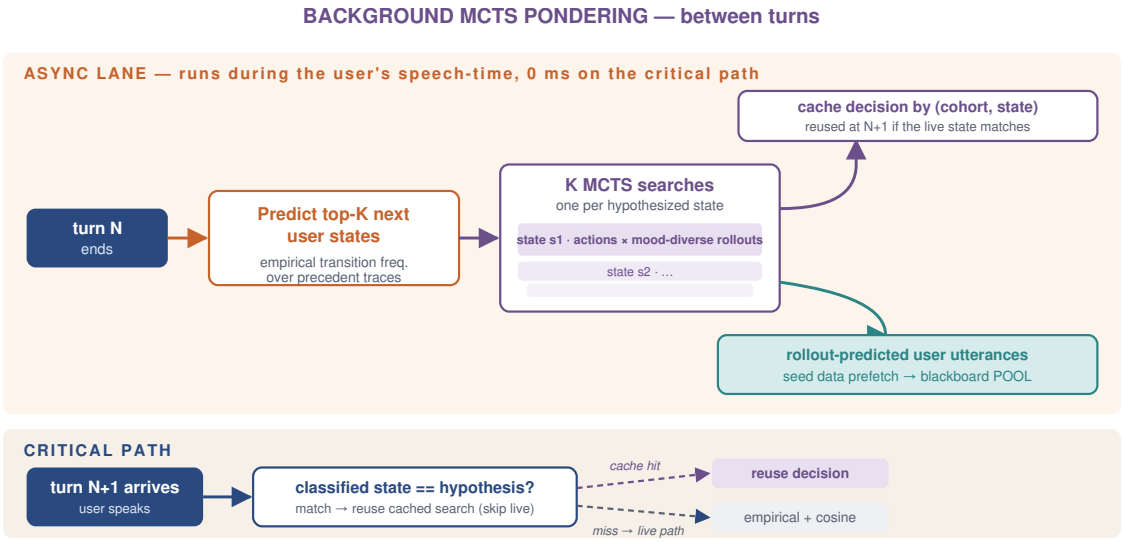

Figure 3. Background MCTS pondering. When turn N ends, the supervisor predicts the top-K likely next user states and runs one MCTS search per state during the user's speech-time. The searches yield two payoffs: a cached action decision (reused at turn $N{+}1$ if the live state matches a hypothesis) and rollout-predicted user utterances that seed data prefetch into the pool. Everything happens off the critical path.

metric (denominator)	exact-key	pool
fetches consumed (of 144)	**11%**	—
turns w/ ≥ 1 item (of 68)	—	**96%**†

†Measured under the earlier LLM-rerank curation, not the final cosine+dedup config.

The two numbers count different things—fetches versus turns—so they are not a single hit-rate that "rises from 11% to 96%." Together, though, they make the structural point: under exact-key lookup most speculative fetches are wasted (89% never consumed), whereas under the pool a mispredicted fetch remains a curated candidate and the live agent still finds a usable item on almost every turn. This compares two variants of our own architecture; §8 specifies the published-baseline comparison (no-cache, naive per-turn RAG) that would isolate the speculative-population contribution on a like-for-like metric.

5.2 Rerank latency, cross-SOP

Under the architecture as described—no LLM on the critical path, cosine + dedup curation, speculative-context prompt—N=5 per SOP:

110 rerank calls; p50 = 246 ms, p99 = 848 ms. The latency profile is near-identical across two SOPs of opposite shape ($\leq$ 4% spread in mean, $\leq$ 8% in p95). We disclose a selection caveat: the car-insurance cell is the 5 successful runs from a batch of 8 (two further runs terminated incompletely and one is reported below as a same-architecture 4/5 comparison); N is small and this is a sampled slice, not an exhaustive batch. A third SOP (credit-card activation) is in the codebase; its earlier exclusion was an action-proposer issue in the SOP-graph gating, fixed in code but *not yet re-benchmarked*—its inclusion is part of the breadth study in §8.

SOP	Succ.	Mean	p95	MCTS path
car-insurance	5/5	293 ms	490 ms	0/60
medical	5/5	304 ms	453 ms	0/60
Agg. (N=10)	**10/10**	**299**	**483**	**0/120**

Table 1. Per-SOP results under the locked configuration (no LLM on the critical path, cosine + dedup curation, speculative-context prompt). Latencies are rerank-step wall-clock. The "MCTS path" column is a design property, not a finding: with `tier3_enabled=false` the router cannot elevate live MCTS, so this is 0 by construction. Source data: `table1_sessions.jsonl` (exported from the run database).

5.3 Pool composition

Across the 97 fetches in the N=10 evaluation:

Source of fetch	Fetches	Share
Empirical predictor (precedent traces)	85	88%
Blocking live fallback (a prefetch *miss*)	9	9%
Background MCTS pondering	3	3%

Only the empirical predictor and pondering are speculative *predictions*; the 9 live-fallback fetches are misses served synchronously on the critical path (§3), reported

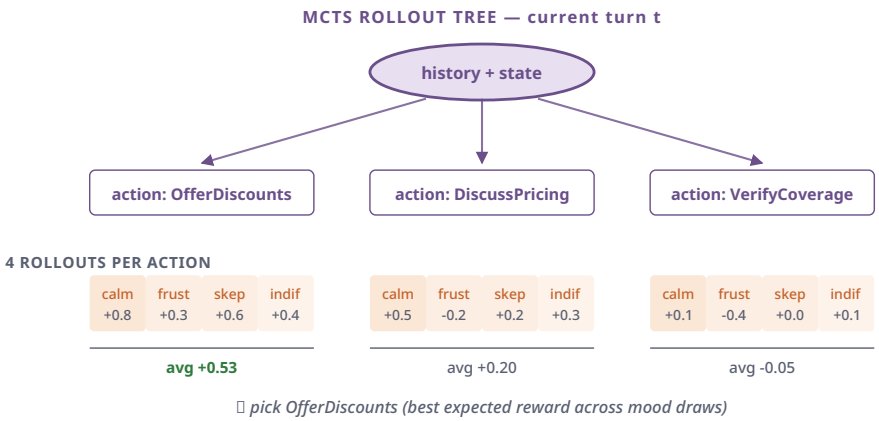

Figure 4. One pondering search for a single hypothesized next state. Candidate agent actions (here three) are each evaluated by mood-diverse rollouts—a draw per cohort mood (calm, frustrated, skeptical, indifferent)—and the action's value is the mean reward across mood draws. The best action is cached for reuse if this hypothesized state is the one the user actually reaches.

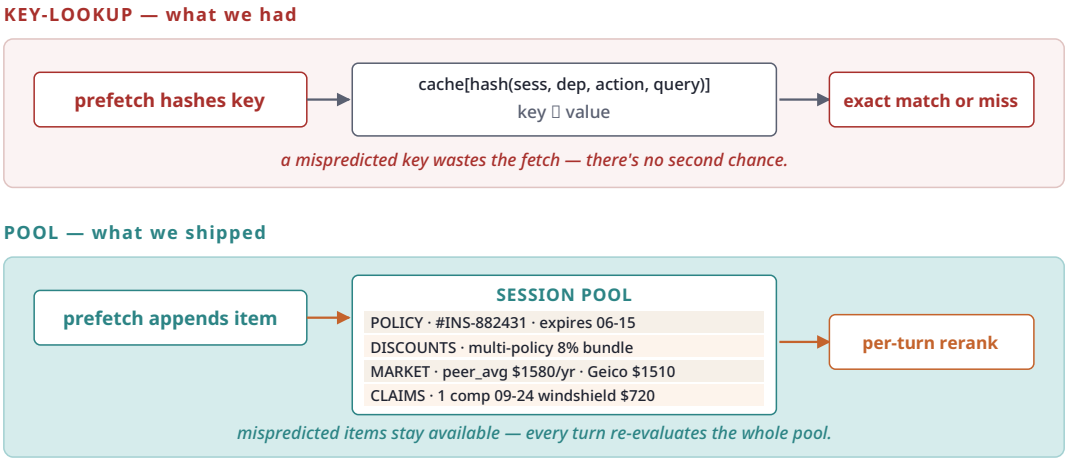

Figure 5. Exact-key lookup (top) vs. the pool (bottom). Under key-lookup a mispredicted key wastes the fetch; under the pool, every fetched item stays available and the live agent curates per turn.

here for completeness. The empirical predictor is load-bearing; the headline numbers do not depend on MCTS pondering, which is under-exercised by the test conditions: of the 213 pondering runs spawned across these sessions, 209 (98%) were cancelled by the next user turn. (Pondering's mean duration of 7.4 s is computed over a larger set of pondering runs than the 213 here; the autopilot's zero-pause turn cycle averages 6.0 s.) Real calls have 2–5 s inter-turn pauses, which would give pondering room to complete; §8 specifies the timing study that measures its contribution under realistic pauses.

Weighting these misses by latency gives a regret view (§2): the 9 misses cost 34.7 s of critical-path time in aggregate (mean 3.86 s each, essentially equal to a pool hit's 3.86 s, so the unweighted miss rate and the latency-weighted regret coincide here). Against the latency the pool removed, that is a realized latency-regret of 4.5% versus a perfect-foresight oracle—the prefetcher captures 95.5% of the available benefit (a converged point estimate, not a regret bound: the oracle is unbudgeted and our predictor is frozen).

5.4 End-to-end latency, and what this work does not address

The rerank step is small, but it is not the whole turn. Over the Table 1 sessions (`gpt-4o` live agent), mean end-to-end agent time per turn is **6.0 s** (p50 5.3 s, p95 10.9 s), decomposed as: classify-and-propose 2.4 s, response generation 1.8 s, situation embedding 0.3 s, and pool rerank 0.3 s. The two live-agent LLM calls (≈4.2 s) dominate; the prefetch machinery this paper contributes adds only the ∼0.3 s rerank.

What the architecture removes from the critical path is *external-data* latency, not LLM latency. The six data dependencies per SOP carry 0.35–6.5 s mocked fetch laten-

cies; across our sessions the pool hides a mean of **22.5 s** of such latency per conversation—data that, fetched synchronously, would otherwise land on the critical path. The ∼300 ms rerank is the price of hiding that 22.5 s.

A smaller live model does not shrink the 4.2 s LLM floor. We re-ran the live agent on `gpt-4o-mini` (N=10, matched config): end-to-end turn time did *not* improve (6.4 s vs. 6.0 s—the structured classify call was in fact 0.8 s *slower* on the smaller model, at identical output length), and session success fell from 10/10 to 4/10 (car-insurance collapsed to 0/5). For these hosted models, model tier did not predict per-request latency, and the weaker model's planning degraded enough to break task completion. Reducing the LLM floor therefore calls for streaming or speculative generation [2, 3]—an orthogonal lever to the external-data problem we target.

6 Ablations

6.1 Why cosine + dedup, not an LLM rerank or a confidence gate

The per-turn curation could be done by an LLM, by cosine ranking, or by cosine ranking gated by a confidence test. We compare them.

An LLM rerank is unnecessary for relevance selection. On a held-out run we computed, per turn, the LLM rerank's picks and the embedding ranking's top-8. 87% of the LLM's picks were already in the embedding top-8; in 77% of turns *all* of the LLM's picks were. The LLM and embedding disagreed on rank within the top-N, not on which items belonged—except in one respect: the LLM avoided returning near-duplicate items, which a raw cosine ranking does not. The dedup pass (§3.5) recovers exactly this. Cosine + dedup therefore matches the LLM's curation at no critical-path LLM cost. The table below reports the *matched* comparison—LLM rerank and cosine + dedup measured under the same architecture and run set—rather than pairing a worst-case LLM run against a best-case cosine run; for reference, the locked cross-SOP configuration (Table 1) holds cosine + dedup at 299 ms mean / 483 ms p95.

rerank latency (matched)	LLM rerank	cosine+dedup
mean	1523 ms	**378 ms**
p95	2167 ms	**940 ms**

A confidence gate does not apply here. A natural optimisation is to skip the LLM only when the embedding ranking is "decisive" (the K-th cosine clears an absolute floor and the gap to the $(K{+}1)$-th exceeds a margin). We set a 0.50 floor / 0.05 margin and, in our notes, expected 60–80% adoption. Adoption was 0%: pool items are conversationally homogeneous (same renewal, same customer), so cosines cluster in a tight low band rather than forming cliffs. The K-th cosine never exceeded 0.421 (Figure 6); the median $K/(K{+}1)$ gap was 0.013. The gate's premise—that some turns have a decisive embedding ranking—is false for this data, which is why we curate unconditionally rather than gate. (Reducing the LLM prompt's input tokens via a top-8 pre-filter is likewise ineffective: it cut input from ∼6–8K to ∼1.8K tokens but left p95 unchanged, because the fast model's cost is dominated by the call itself, not prompt length.)

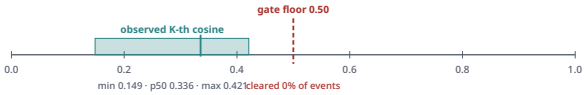

Figure 6. Live-message/pool-item cosine distribution over 44 gate-eligible turns. The observed K-th cosine (teal band, max 0.421) never reaches the 0.50 gate floor (dashed). Topically homogeneous pools do not produce the cliffs a confidence gate needs.

6.2 The speculative-context prompt is necessary

Because the cosine rerank returns top-3 items unconditionally, it cannot decline on a turn where nothing in the pool is relevant—most visibly on conversation-closing turns, where the user winds down ("thanks, I'll get back to you") and the top pool cosine falls to 0.10–0.18. The question is whether the live agent uses those low-relevance items. That depends on the prompt's framing. We hold the architecture fixed and vary only the label on the prefetched-context block:

	Curated	Speculative	Δ
Session success	3/5	**4/5***	recov.
Mean rerank	378 ms	320 ms	–15%
p95 rerank	940 ms	429 ms	–54%

*The remaining non-success was a turn-budget cap-hit on the turn the user agreed to renew—an action-proposer issue, not a curation failure.

Under the "curated, high-relevance" framing the agent treats the items as authoritative and answers about discounts and coverage on turns where the user is leaving; under the speculative framing it ignores the misfits and closes the call. The latency difference is noise at this N; the success difference is the point. We considered a mechanical alternative—a cosine floor that returns zero items below a threshold—but it is a tuned heuristic stacked on a tuned heuristic, and it addresses the symptom (items present) rather than the cause (the prompt over-claims their reliability). The prompt fix is architecture-agnostic and requires no threshold.

Observation, not yet a principle. This is consistent with Shi et al. [13], who showed that instructing

an LLM to ignore irrelevant context mitigates the distraction effect; our addition is the *framing-mismatch* ablation in an asynchronous speculate/consume pipeline—that an over-confident "curated" label, versus an honest "speculative" one, is what tips the agent into over-using low-relevance items—and the closing-turn failure mode it exposes. Whether this hardens into a law for such pipelines ("the consumer's prompt should reflect the speculation's actual reliability") is a hypothesis a single framing-pair on one SOP cannot establish; §8 specifies the multi-framing, multi-SOP study needed.

6.3 What predicts retrieval, and why MCTS does not

The supervisor's most expensive component is the background MCTS pondering of §3.3. We measure whether it earns its cost for the only job it does on the pool—predicting what to prefetch. Retrieval prediction decomposes into three sub-tasks: (i) *which data source* (the next agent action, which names its dependencies); (ii) *structured parameters* (which customer, policy, vehicle)—not predicted at all, since they are session facts already on the blackboard once identity is established; and (iii) the *free-text query* a RAG dependency needs, which depends on the specific question the user will ask and is genuinely generative.

Sub-task (i): which source. We score next-action prediction against ground truth on 50 turn-transitions from logged sessions:

predictor	recall@3	latency	tokens/turn
empirical (count over traces)	**88%**	∼1 ms	**0**
naive LLM (one call)	38%	1.6 s	∼1.6K
MCTS pondering	33%*	10.6 s	∼25.6K

*modal next-action across rollouts (logged); pondering also has a 62% cancellation rate.

Counting wins decisively: predicting an agent's next move is a question about its *habits*, which the empirical predictor memorizes from `precedent_traces`; MCTS and the naive LLM reason about a *hypothetical* future and do worse, at 10^3–$10^4\times$ the token cost.

Sub-task (iii): the generative query. Only RAG dependencies need it. We render the query three ways—a structured stub (cohort/state/action only), a cheap single-LLM next-utterance prediction, and MCTS rollout text—and measure retrieved-document overlap against the query rendered from the *actual* next user utterance, across corpora of increasing granularity:

{user_text} source	coarse (25)	fine (44)	large (63)	tok
structured stub	70%	48%	43%	0
cheap LLM	73%	**62%**	**54%**	∼300
MCTS rollouts	60%	∼60%	∼60%	∼25.6K

The value of generative query prediction *scales with corpus granularity*: when many documents share a cohort/state and differ only by the specific situation, the structured stub collapses (70→43%) while a single small-model utterance prediction holds an 11–14 pp lead—matching MCTS at ∼85× lower cost. (The fine and large corpora are constructed stress-tests with realistic situation-variants, not production data.)

Resolution. Empirical counting for the action and structured parameters; one cheap LLM call for the generative query slot, gated on whether a RAG dependency is in the predicted horizon; MCTS for neither. This retires pondering from the retrieval path—a ∼99% token reduction with *higher* action accuracy—and aligns with concurrent findings that pattern-mined speculation from execution traces (PASTE [7]) outperforms search-based lookahead for tool/data prefetch. We retain pondering in the architecture only as a cold-start option (when no traces have accumulated) and for its cached-decision payoff, both deferred to future evaluation.

6.4 Online adaptation: decay and shrinkage, and why not exploration

The framing of §2 casts prefetch as a budgeted bandit and flags the predictor's two limitations: it weights all history equally (no drift adaptation) and, as a frozen count, can be brittle at cold start. We address both *without* the bandit's exploration machinery, and measure why exploration is not needed.

We add two cheap mechanisms to the empirical predictor. **Recency decay** weights each precedent by $\exp(-\Delta t/\tau)$ (half-life τ; 30 days by default) so the action distribution tracks drift. **Shrinkage** regularises a sparse (cohort, state, mood) cell toward the SOP-level action marginal with pseudo-count κ: $P(a) = (w_a + \kappa\,\pi(a))/(W + \kappa)$, where w_a is the cell's decayed weight, $W = \sum_a w_a$, and π is the marginal. Rich cells ($W \gg \kappa$) are unchanged; sparse cells fall back smoothly to the prior. Both are on by default.

We evaluate on a held-out temporal split: order all one-step transitions chronologically, hold out the most recent 20% as a fixed test set, and grow a training cutoff over the remaining history—the predictor may only count precedents created before the cutoff (a true train-on-past / test-on-future protocol). Figure 7 reports held-out recall@3 as training history accumulates.

Two results. First, **shrinkage is a cold-start win and a warm-state no-op**: +12 pp recall@3 at 10% of history, decaying to within noise of the baseline by 50% and converging at ∼73%. This is exactly the profile that justifies defaulting it on—it helps when data is thin and is harmless once it is not. Second, and more pointed, **Thompson exploration adds nothing over shrinkage**, even in the thin-data regime where exploration is

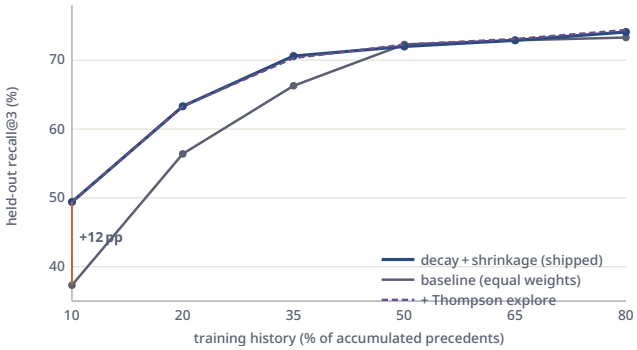

Figure 7. Held-out recall@3 (latest 20% of transitions) vs. training history. Decay + shrinkage gives a large cold-start gain (+12 pp at 10% of history) that fades to neutral as data accumulates and both predictors converge (∼73%). Adding Thompson exploration on top tracks decay + shrinkage exactly—it adds nothing, even where it should help most.

supposed to pay (49.4% vs. 49.4% at 10%). The reason is structural: our setting has *full-information* feedback—the conversation reveals the true next action every turn, whether or not we prefetched its dependency (a miss is still observed, §5.2)—so each turn is a free labelled example. There is no feedback starvation for exploration to relieve; a prior (shrinkage) already covers the only failure mode (sparse cells). The prefetch problem is therefore best read not as a true bandit but as *online supervised learning under a budget*: the bandit vocabulary names the objective (budgeted regret), but exploration—its signature mechanism—is empirically inert here. We keep Thompson sampling implemented but off by default.

7 Discussion

When the pool reframe helps. Whenever prefetch is driven by predictions that are *approximately* right. The pool gives the system a second chance to use approximately-correct fetches; exact-key caching throws them away. The more the needed data depends on the live utterance (RAG, parameterised queries), the larger the gap.

When cosine + dedup suffices. When pool items live in the user's turn space, so embedding similarity tracks relevance (here, the 87% LLM/embedding agreement). For highly heterogeneous pools spanning many domains, an LLM's discrimination may still add value beyond cosine ranking; the dedup pass is the minimal recovery of the LLM's contribution we observed.

Scope of the prompt observation. If the framing-mismatch result generalizes (§6.2), it would apply to any consumer reading from a speculatively-populated surface—blackboards, vector stores, search caches—as, in effect, the asynchronous-systems analogue of accurate API documentation: the interface should describe what it actually guarantees.

8 Limitations and the experiments that complete this work

We are explicit that the present evaluation establishes the architecture's mechanics and internal consistency, not yet its standing against the field. The following experiments are planned to close that gap.

- **Statistical power.** Re-run every headline condition at $N \geq 20$ per SOP with Wilson confidence intervals on rates and bootstrap intervals on latency percentiles. The present N=5 supports coarse confirmation only.
- **Published baselines and a like-for-like metric.** The 11%-consumed / 96%-supplied pair compares two of our own variants on different denominators. The isolating comparison is against (a) no cache (live fetch), (b) naive per-turn RAG over a session vector store with no prefetch, and (c) an exact-key prefetch cache—all scored on one common metric (turns supplied a usable item), separating the value of *speculative population* from the value of the pool data structure.
- **A quality-linked metric.** Session success is coarse. We will add a per-turn data-grounding measure (did the reply correctly use available prefetched data on data-dependent questions?) and redefine effective hit rate as relevant-*and*-used rather than merely returned.
- **The prompt principle as a controlled study.** The speculative-context result is one framing pair on one SOP. A multi-framing grid (curated / neutral / speculative / speculative+examples) across both SOPs, ideally a second task, is needed to claim a general principle.
- **Real users via a live-avatar harness.** We have built a testing harness in which a human speaks to a GPT-Realtime voice avatar steered by the supervisor under full SOP control. A human study (A/B with vs. without prefetch) converts "simulator-only" from a limitation into a validated result and is the single largest credibility upgrade available to us.
- **MCTS pondering under realistic timing.** Re-run with injected 2–5 s inter-turn pauses to measure pondering's completion rate and contribution; the present zero-pause condition cancels 98% of pondering runs and so cannot speak to its value.
- **Context generalisation for unseen cells.** We added recency decay and shrinkage to the predictor and found exploration empirically inert under our full-information feedback (§6.4). The open direction is *parametric* contextual sharing—a LinUCB-style

model [17] over context features so an unseen (cohort, state) cell borrows strength from similar ones, rather than the current tabular counts with hard-coded backoff. Our held-out study is also a single temporal split on one accumulated corpus; a multi-deployment evaluation with injected distribution shift would test the decay mechanism directly.

- **Breadth and cost.** Include credit-card activation (the blocking action-proposer issue is fixed in code; re-evaluation pending) and a fourth domain; and report per-turn LLM-call and token cost for cosine + dedup vs. LLM rerank vs. naive RAG.

9 Conclusion

We presented a two-system architecture for SOP-constrained voice agents that moves external-data retrieval off the critical path: across our sessions the supervisor's speculative pool hides a mean of 22.5 s of external-data latency per conversation while the curation step it adds costs only ∼300 ms. As a pilot validation (N=5, simulator users, internal baselines), we evaluated three components: a misprediction-tolerant pool cache (under exact-key lookup 89% of speculative fetches go unconsumed, whereas under the pool the live agent finds a usable item on 96% of *turns*—different denominators, not a single rate, with the isolating common-metric baseline owed in §8); a cosine-plus-dedup rerank shown by a matched ablation to recover an LLM reranker's only value-add at a fraction of its latency; and a speculative-context prompt whose framing-mismatch ablation keeps the live agent from over-using low-relevance items. We are explicit about what this work does *not* address: total turn time (∼6 s) is dominated by the live agent's own LLM calls, which a smaller model does not reduce (§5.4); reducing that is the domain of streaming and speculative generation [2, 3], orthogonal to the data-latency problem we target. §8 lays out the controlled studies—larger N with confidence intervals, published baselines on a common metric, a quality-linked metric, and a real-user avatar study—that bring the evaluation to full strength.

References

[1] Qiu, J., et al. *VoiceAgentRAG: Solving the RAG Latency Bottleneck in Real-Time Voice Agents Using Dual-Agent Architectures*, 2026. arXiv:2603.02206.

[2] Zou, et al. *LTS-VoiceAgent: A Listen-Think-Speak Framework for Efficient Streaming Voice Interaction via Semantic Triggering and Incremental Reasoning*, 2026. arXiv:2601.19952.

[3] Mai, L. *RelayS2S: A Dual-Path Speculative Generation for Real-Time Dialogue*, 2026. arXiv:2603.23346.

[4] Schwarz, A., He, D., Van Segbroeck, M., Hethnawi, M., and Rastrow, A. *Personalized Predictive ASR for Latency Reduction in Voice Assistants*. Interspeech 2023. arXiv:2305.13794.

[5] Yoshimaru, N., Okuma, M., Iio, T., and Hatano, K. *AsyncMLD: Asynchronous Multi-LLM Framework for Dialogue Recommendation System*, 2023. arXiv:2312.13925.

[6] Li, Z., et al. *ChatSOP: An SOP-Guided MCTS Planning Framework for Controllable LLM Dialogue Agents*, 2024. arXiv:2407.03884.

[7] Sui, Y., et al. *Act While Thinking: Accelerating LLM Agents via Pattern-Aware Speculative Tool Execution*, 2026. arXiv:2603.18897.

[8] Wang, Z., et al. *Speculative RAG: Enhancing Retrieval Augmented Generation through Drafting*. ICLR 2025. arXiv:2407.08223.

[9] Bang, F. *GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings*. Proc. 3rd Workshop for NLP Open-Source Software (NLP-OSS), 2023.

[10] Chan, B. J., Chen, C.-T., Cheng, J.-H., and Huang, H.-H. *Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks*, 2024. arXiv:2412.15605.

[11] Zhang, Q., Wornow, M., Wan, J., and Olukotun, K. *Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM Agents*. NeurIPS 2025. arXiv:2506.14852.

[12] Carbonell, J., and Goldstein, J. *The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries*. SIGIR 1998.

[13] Shi, F., et al. *Large Language Models Can Be Easily Distracted by Irrelevant Context*. ICML 2023. arXiv:2302.00093.

[14] Leviathan, Y., Kalman, M., and Matias, Y. *Fast Inference from Transformers via Speculative Decoding*, ICML 2023. arXiv:2211.17192.

[15] Lai, T. L., and Robbins, H. *Asymptotically Efficient Adaptive Allocation Rules*. Advances in Applied Mathematics 6(1):4–22, 1985.

[16] Auer, P., Cesa-Bianchi, N., and Fischer, P. *Finite-time Analysis of the Multiarmed Bandit Problem*. Machine Learning 47:235–256, 2002.

[17] Li, L., Chu, W., Langford, J., and Schapire, R. E. *A Contextual-Bandit Approach to Personalized News Article Recommendation*. WWW 2010. arXiv:1003.0146.

[18] Badanidiyuru, A., Kleinberg, R., and Slivkins, A. *Bandits with Knapsacks*. Journal of the ACM 65(3):13, 2018 (FOCS 2013). arXiv:1305.2545.

[19] Slivkins, A. *Introduction to Multi-Armed Bandits*. Foundations and Trends in Machine Learning 12(1–2):1–286, 2019. arXiv:1904.07272.