Semantic Context for Tool Orchestration

Robert Müller [1]

Abstract

Effectively navigating large, changing action spaces is a critical challenge for reinforcement-learning orchestrators that rely on a non-stationary catalogue of APIs, tool endpoints, LLMs, and auxiliary agents. Each tool has a human-readable description — e.g. its input schema, intended functionality, benchmark scores and output format - typically ignored by stanard RL. We introduce and analyse a family of semantic context RL algorithms that embed the semantic context of each tool alongside the state of the environment and learn policies that generalize between tool versions, additions, and removals. We conclude with empirical studies of the role of semantic context in linear bandits and LLM in-context learning.

1. Introduction

The capacity of intelligent systems, particularly Large Language Models (LLMs), is significantly amplified by their ability to orchestrate external tools—such as APIs, auxiliary agents, or specialized functions (Patil et al., 2023; Qin et al., 2023; Schick et al., 2023). This orchestration is a sequential decision-making task: given a user query and a dynamic tool catalogue, an agent must select and use the most appropriate tool. While reinforcement learning (RL) offers a principled framework, naive application (e.g., LLMs generating tool invocations token-by-token) creates intractably large action spaces (V^L with vocabulary size V and sequence length L), hindering learning. A common simplification presents the agent with an explicit list of O available indices or tools, $\mathcal{A}_{avail} = \{a_1, \ldots, a_O\}$, from which to select. However, this often discards valuable semantic descriptions $D(\tau)$ associated with each tool (e.g., API doc strings, capability summaries). Recent works using RL to train LLM to orchestrate tools rely on the provision of tool names and descriptions in the prompts (Feng et al., 2025; Singh et al., 2025; Zhang et al., 2025; Qian et al., 2025). (Lin et al., 2024) improve tool call reliability by random augmentation of tool and argument names, thus pushing the model to rely on tool descriptions. This paper investigates the critical and quantifiable advantages of equipping agents with what we term the *Semantic Context (SC)*—the collection of semantic descriptions for all currently available actions.

This SC is not merely a helpful addition but a fundamental component for effective tool orchestration. Our work establishes this through three core findings.

First, we provide a theoretical and empirical foundation showing that even in **static settings** with a fixed tool set, SC enables more efficient learning. To do this, we develop **SC-LinUCB**, a bandit algorithm, and prove that it achieves favourable regret compared with non-semantic baselines by creating a more parsimonious and accurate reward model (Section 3). Empirical support is provided by SC-LINUCB and in-context learning experiments with LLM.

Second, we demonstrate SC's critical role in **dynamic adaptation**. Our experiments show that as tools are added or removed, an agent leveraging SC adapts gracefully, whereas baselines suffer from catastrophic forgetting and require costly retraining. This highlights SC as a key enabler for continual learning in evolving environments for both, SC-LINUCB and in-context learning LLM.

Finally, we show how SC makes tool orchestration practical at scale through a **FiReAct (Filter-Reason-Act) pipeline**. We demonstrate that semantically filtering a large corpus of tools into a small, relevant set is *essential* for maintaining high accuracy as the number of tools grows into the thousands. This scalable application bridges our theoretical insights with the practical challenges faced by modern LLM agents (SubSection 5.3).

Our research draws from the contextual bandit framework (Li et al., 2010; Chu et al., 2011), with LinUCB (Abbasi-Yadkori et al., 2011) as a cornerstone, and contributes by rigorously analysing features from *a priori semantic embeddings of natural language action descriptions* and quantifying their regret impact. While action representation learning from interaction is common in RL (Chandak et al., 2019; Pathakota et al., 2023), and using natural language for actions has been explored (Tennenholtz and Mannor, 2019), our focus is on leveraging pre-existing, structured semantic information. Addressing dynamic ac-

[1]Aganthos. Correspondence to: Robert Müller <aganthos.ai@gmail.com>.

Computer Use Agents Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada. 2025. Copyright 2025 by the author(s).

tion spaces, central to continual learning, we differ from Chandak et al. (2020) who infer latent action structures; we demonstrate how *explicit, given semantic descriptions* enable robust adaptation without relearning action space representations. This complements continual RL's focus on evolving reward/transition functions (Müller and Pacchiano, 2022; Khetarpal et al., 2020), aiming to furnish principled insights for more sample-efficient, generalizable, and adaptive tool-orchestrating agents that explicitly leverage SC.

When dealing with high dimensional task-/ action spaces there is a variety of approaches to dial down complexity. Examples include learning action elimination networks(Zahavy et al., 2019) to approaches partitioning the task space based on task embeddings (Müller et al., 2020). More recent tool-RAG methods tackle the problem from a retrieval perspective: small LMs learn a function-mask head that suppresses irrelevant APIs at inference time (Lin et al., 2024); completeness-oriented retrievers rank tools so that only a minimal yet sufficient subset is forwarded to the reasoner (Qu et al., 2024; Shi et al., 2025).

2. Problem Formulation

We model the task of selecting an appropriate tool for a given query as a **contextual bandit** problem. This framework allows us to rigorously analyse the decision-making that underpins tool orchestration.

At each discrete time step $t \in \{1, \ldots, T\}$, an agent observes a context (a user query $q_t \in \mathcal{Q}$) and must select an action a_t from a set of currently available tools, $\mathcal{A}_t = \{a_1, \ldots, a_{O_t}\}$ of magnitude O_t. The environment is stochastic: for a given query q_t, each action $a_i \in \mathcal{A}_t$ has a true but unknown probability of success, $p^{\text{eff}}(a_i, q_t)$. After selecting a_t, the agent receives a stochastic binary reward $r_t \in \{0, 1\}$, drawn from a Bernoulli distribution governed by this probability: $r_t \sim \text{Bernoulli}(p^{\text{eff}}(a_t, q_t))$.

The agent's objective is to learn a policy $\pi(a_t|q_t, H_{t-1})$ that maximizes the cumulative reward (or Return), $\sum_{t=1}^{T} r_t$. This is equivalent to minimizing the **Cumulative Expected Regret**, defined as the sum of the per-step differences between the expected reward of the optimal action for a given query and the expected reward of the action the agent actually chose:

$$R_T = \sum_{t=1}^{T} \left(\max_{a \in \mathcal{A}_t} p^{\text{eff}}(a, q_t) - p^{\text{eff}}(a_t, q_t) \right). \quad (1)$$

The central hypothesis of this paper is that an agent's policy can learn more efficiently and adapt faster to changes in the action space $\mathcal{A}_t$ if it explicitly leverages the **Semantic Context**, the rich descriptions associated with each action, rather than treating actions as abstract, opaque indices.

Definition 2.1 (Semantic Context, $C_S(\mathcal{A}_t)$)**.** *Given $\mathcal{A}_t$, the set of available actions at time t, the **semantic context** $C_S(\mathcal{A}_t)$ is the collection of semantic information related to these actions. Specifically, $C_S(\mathcal{A}_t) = \{(a_i, D(a_i))\}_{a_i \in \mathcal{A}_t}$, where for each action a_i, $D(a_i)$ is its natural language description (e.g., docstring). Each description is mapped to a d_{emb}-dimensional **semantic context embedding** $\phi(a_i) = \Xi(D(a_i))$ via an embedding function Ξ. This provides structured, a priori information about the available actions, allowing an agent's policy $\pi(a_t|s_t, C_S(\mathcal{A}_t))$ to leverage both the usual state s_t (which includes the query q_t) and the SC.*

Our analysis begins with the simplest instantiation of this framework: a stationary environment where the set of actions is fixed.

Definition 2.2 (Semantic Context Bandit, SC-Bandit)**.** *An **SC-Bandit** models a single-step decision with a static action space $\mathcal{A}_{avail}$. At each step t, given a query q_t, the agent selects an action a_t based on its policy $\pi(a_t|q_t, C_S(\mathcal{A}_{avail}))$, where the Semantic Context is fixed.*

For SC MDP A.1 and the Lifelong SC MDP A.2 with non-stationary action space we refer to appendix A.2. In all frameworks, the central hypothesis is that explicit incorporation and effective utilization of the Semantic Action Context $C_S(\mathcal{A}_t)$ enable agents to achieve superior learning efficiency, generalization, and adaptability.

3. Theoretical Framework: Semantic LinUCB

We analyse Semantic Contextual Linear UCB (SC-LinUCB), an adaptation of the LinUCB algorithm (Abbasi-Yadkori et al., 2011) that leverages semantic information from action descriptions. Our analysis demonstrates that by incorporating well-structured semantic features, SC-LinUCB can achieve significantly lower regret than LinUCB variants relying on non-semantic action representations. This improvement stems from a more efficient representation of the underlying reward structure, leading to better generalization and reduced exploration complexity.

Our theoretical contribution focuses on how the specific construction of semantic features $\mathbf{x}^{(sem)}$ for SC-LinUCB leads to a more favorable instantiation of this generic bound compared to using non-semantic features.

3.1. Contextual Linear Bandits and Feature Design

We operate within the standard contextual linear bandit framework (detailed in Appendix B.1). At each time step t, given a query (context) embedding $\mathbf{q}_t \in \mathbb{R}^{d_q}$, the agent selects a tool τ_j from the K_t available tools. Each tool τ_j is associated with a semantic description embedding $\phi_j \in \mathbb{R}^{d_{desc}}$. The expected reward $\mathbb{E}[R_t|\mathbf{x}_{t,j}] = \mathbf{x}_{t,j}^T \theta^*$ is linear in the constructed d-dimensional feature vector $\mathbf{x}_{t,j}$.

The SC-LinUCB algorithm itself is Algorithm 2 (Appendix B.2).

The core of our analysis lies in comparing two feature construction strategies:

- **SC-LinUCB Semantic Features ($\mathbf{x}^{(sem)}$):** We construct $\mathbf{x}_{t,j}^{(sem)} = [\mathbf{q}_t; \phi_j; \text{sim}(\mathbf{q}_t, \phi_j); 1]$. The resulting feature dimension is $d_{sem} = d_q + d_{desc} + 1 + 1$. This design explicitly incorporates query attributes, tool semantic attributes, and their direct alignment.
- **LinUCB-NS Non-Semantic Features ($\mathbf{x}^{(non-sem)}$):** As a baseline, we use features $\mathbf{x}_{t,j}^{(non-sem)} = [\mathbf{q}_t; \mathbf{e}_j; 1]$, where $\mathbf{e}_j \in \mathbb{R}^K$ is the one-hot encoding for tool τ_j. The dimension is $d_{non-sem} = d_q + K + 1$. This baseline distinguishes tools by identity but lacks explicit shared semantic information.

The generic regret for LinUCB algorithms stated in Appendix B.3), scaling as $\mathcal{R}_T = \tilde{O}(d \cdot \sigma_{eff} \cdot \sqrt{T})$, where d is the feature dimension and σ_{eff} is the effective noise standard deviation (incorporating observation noise and linear model approximation error).

3.2. Regret Advantage via Efficient Semantic Representation

To formalize the advantage of $\mathbf{x}^{(sem)}$, we introduce an assumption about the nature of the true reward function.

Assumption 3.1 (Semantically Structured Rewards)**.** *The true expected reward function $f^*(\mathbf{q}, \phi)$ is primarily determined by a limited number of underlying semantic interaction patterns between queries and tool semantic properties. Specifically, there exists an optimal linear model in the semantic feature space, $(\mathbf{x}_{t,j}^{(sem)})^T \theta_{sem}^*$, that approximates $f^*(\mathbf{q}_t, \phi_j)$ with a mean squared error $\sigma_{approx,sem}^2$. Further, to achieve a comparable or better linear approximation quality using non-semantic one-hot features, i.e., $(\mathbf{x}_{t,j}^{(non-sem)})^T \theta_{non-sem}^* \approx f^*(\mathbf{q}_t, \phi_j)$ with error $\sigma_{approx,non-sem}^2 \geq \sigma_{approx,sem}^2$, the dimensionality $d_{non-sem}$ (which scales with K) may be significantly larger than d_{sem} if K is large and there is semantic redundancy across tools (i.e., $d_{desc} + 1 \ll K$).*

Theorem 3.2 (Regret Reduction for SC-LinUCB)**.** *Under Assumption B.1 (for both SC-LinUCB with $(d_{sem}, \sigma_{eff,sem}, S_{sem}, L_{sem})$ and LinUCB-NS with $(d_{non-sem}, \sigma_{eff,non-sem}, S_{non-sem}, L_{non-sem})$) and Assumption 3.1: SC-LinUCB achieves a cumulative regret $\mathcal{R}_T(SC)$ that is less than or equal to the regret of LinUCB-NS, $\mathcal{R}_T(NS)$, if its semantic features lead to a more favorable combination of dimensionality and effective noise. Specifically, $\mathcal{R}_T(SC) \leq \mathcal{R}_T(NS)$ if the factor $d_{sem} \cdot \sigma_{eff,sem}$ (ignoring constants and polylog terms from α) is smaller than $d_{non-sem} \cdot \sigma_{eff,non-sem}$. A significant improvement ($\mathcal{R}_T(SC) \ll \mathcal{R}_T(NS)$) is realized if:*

1. ***Parsimonious Representation:*** *$d_{sem} \ll d_{non-sem}$ (achievable if $d_{desc} + 1 \ll K$) while maintaining comparable or better approximation quality ($\sigma_{eff,sem} \lesssim \sigma_{eff,non-sem}$). The regret reduction factor is roughly $d_{sem}/d_{non-sem}$.*
2. ***Superior Fit:*** *Even if $d_{sem} \approx d_{non-sem}$, if semantic features provide a substantially better linear approximation, then $\sigma_{eff,sem} \ll \sigma_{eff,non-sem}$, leading to a regret reduction factor of roughly $\sigma_{eff,sem}/\sigma_{eff,non-sem}$.*

Proof Sketch. The underlying mechanism is that SC-LinUCB learns a single model $\hat{\theta}_{sem}$ over features that encode shared semantic properties. This enables generalization: experience with one tool informs the valuation of other semantically similar tools for similar contexts, leading to more efficient exploration of the (context $\times$ action) space compared to LinUCB-NS, which learns tool-specific parameters more independently via orthogonal one-hot encodings.

The regret for LinUCB (Theorem 19.2 in (Lattimore and Szepesvári, 2020)) is $\mathcal{R}_T = \tilde{O}(\alpha d \sqrt{T})$, where the exploration parameter $\alpha \approx \sigma_{eff}\sqrt{d \log(TK/\delta)} + \text{const} \cdot S_\theta$. Thus, $\mathcal{R}_T \approx \tilde{O}((\sigma_{eff} d + S_\theta \sqrt{d})\sqrt{T})$. For simplicity in comparing dominant effects, we consider the scaling $\tilde{O}(d \cdot \sigma_{eff} \cdot \sqrt{T})$.

Case 1: Parsimony ($d_{sem} \ll d_{non-sem}$ with $\sigma_{eff,sem} \approx \sigma_{eff,non-sem} = \sigma_{eff}$). Assumption 3.1 posits that relevant semantic information can be captured in $d_{sem} = d_q + d_{desc} + 2$ dimensions. If K tools share significant semantic redundancy (e.g., many tools belong to a few archetypes effectively distinguished by d_{desc} features, so $d_{desc} \ll K$), then d_{sem} does not need to scale with K. In contrast, LinUCB-NS requires $d_{non-sem} = d_q + K + 1$ to assign distinct parameters to each tool via its one-hot encoding $\mathbf{e}_j$. If $d_{desc} + 1 \ll K$, then $d_{sem} \ll d_{non-sem}$. Since the regret scales with d, SC-LinUCB directly benefits from operating in a lower-dimensional parameter space, assuming the quality of linear fit (σ_{eff}) is maintained. This reduction in d means SC-LinUCB needs to estimate fewer parameters, and the confidence ellipsoids effectively cover the parameter space with fewer samples. The sum of exploration terms $\sum s_{t,\tau_t}^2$ is bounded by $O(d_{sem} \log T)$ for SC-LinUCB versus $O(d_{non-sem} \log T)$ for LinUCB-NS (Lemma B.3).

Case 2: Superior Fit ($\sigma_{eff,sem} < \sigma_{eff,non-sem}$ with $d_{sem} \approx d_{non-sem}$). If the true reward function $f^*(\mathbf{q}, \phi)$ is more closely aligned with a linear model over semantic features $\mathbf{x}^{(sem)}$ than over non-semantic features $\mathbf{x}^{(non-sem)}$, then the approximation error component of $\sigma_{eff,sem}$ will

be smaller than that of $\sigma_{eff,non-sem}$. Semantic features like direct query-tool similarity ($\mathbf{q}_t^T \phi_j$) can capture critical interaction effects that might be poorly represented by the interactions of $\mathbf{q}_t$ with orthogonal one-hot vectors $\mathbf{e}_j$ in a linear model. A smaller $\sigma_{eff,sem}$ makes the exploration parameter α_{sem} smaller for SC-LinUCB (for the same confidence δ), leading to tighter UCB scores $p_{t,j}$, thus less exploration of suboptimal arms. This means the term $2\alpha s_{t,\tau_t}$ in the instantaneous regret bound is smaller on average.

Generalization Impact on Exploration Dynamics: Beyond the direct impact on d and σ_{eff}, the shared nature of $\hat{\theta}_{sem}$ in SC-LinUCB means that an update from exploring $(\mathbf{q}_t, \phi_a)$ refines weights associated with semantic components (e.g., specific dimensions of ϕ_a or the similarity feature). If another tool ϕ_b shares these semantic components relevant to $\mathbf{q}_t$, the UCB score for $(\mathbf{q}_t, \phi_b)$ is implicitly updated and its uncertainty reduced more effectively than in LinUCB-NS, where the learning for tool a (via $\mathbf{e}_a$) is largely isolated from tool b (via $\mathbf{e}_b$). This leads to a more efficient "pruning" of the (context $\times$ action) space, reducing the cumulative sum $\sum s_{t,\tau_t}$. The combination of these effects results in $\mathcal{R}_T(SC\text{-}LinUCB) \leq \mathcal{R}_T(LinUCB\text{-}NS)$, with significant improvement when semantic features offer substantial parsimony or fit advantages. □

4. SC-LinUCBExperiments

To empirically evaluate the impact of semantic information in contextual bandit settings, we employ two variants of the shared LinUCB algorithm (Abbasi-Yadkori et al., 2011). Both agents aim to learn a single shared parameter vector $\theta^* \in \mathbb{R}^d$ to predict expected rewards $\mathbb{E}[R_t|\mathbf{x}_{t,j}] \approx \mathbf{x}_{t,j}^T \theta^*$. Their core distinction lies in the construction of the feature vector $\mathbf{x}_{t,j}$ for a given query (context) $\mathbf{q}_t$ and tool (action) τ_j.

We compare SC-LinUCB and LinUCB-OneHot using their respective semantic and non-semantic feature constructions detailed in Section 3.1. For this experiment with $K = 6$ tools, $d_{sem} = 6$ and $d_{non-sem} = 9$. Results are averaged over $N_{runs} = 15$ seeds.

We conduct a series of experiments to empirically validate our theoretical findings and demonstrate the practical benefits of using semantic action features. We first focus on an intra-episode setting with a fixed action set, then evaluate adaptation in a continual learning scenario with dynamic action sets. Experiments are run on Colab (free tier CPU).

4.1. Experiment 1: Intra-Episode Efficiency in a Multi-Context Environment

Objective. This experiment validates our theoretical claim that SC-LinUCB achieves lower regret than LinUCB-OneHot by leveraging semantic action features in a multi-context setting with a fixed action set ($K = 6$).

Environment Setup. The environment features $N_Q = 3$ distinct query types (contexts) that cycle periodically over $T = 10000$ timesteps. Each of the $K = 6$ tools τ_j is associated with a 2D toy semantic embedding ϕ_j, derived from one of three underlying archetypes plus noise. Each query type is designed to align semantically with one specific tool archetype. Stochastic rewards $R_t \in \{0, 1\}$ are determined by the semantic alignment between the current query $\mathbf{q}_t$ and the chosen tool's embedding ϕ_j. Full details are in Appendix C.1.2.

Results. Figure 1 presents the average cumulative regret (log scale). SC-LinUCB (orange line) shows substantially superior performance, maintaining an exceptionally low cumulative regret (around 10^0) over 10000 timesteps, indicating rapid convergence to a nearly optimal policy across contexts. LinUCB-OneHot (blue line), while exhibiting sublinear regret (indicating learning), incurs orders-of-magnitude higher regret (exceeding 10^3). This stark difference underscores SC-LinUCB's ability to generalize semantic patterns across different (context, tool) pairings, leading to vastly improved sample efficiency compared to the baseline, which learns tool utilities more independently. Both algorithms used $\alpha = 0.3$. For ablations over the value of α we refer to figure 5.

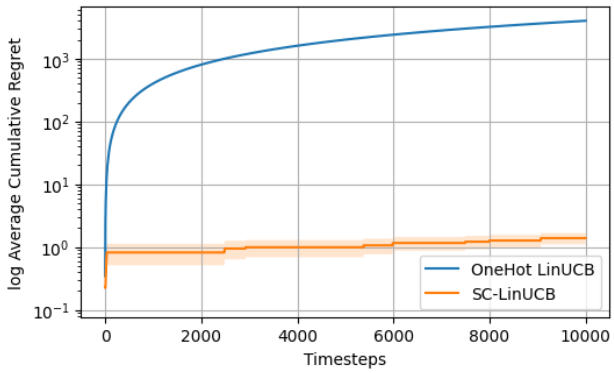

Figure 1. Average Cumulative Regret (log scale) for SC-LinUCB and LinUCB-OneHot in the multi-context (switching) with fixed toolset experiemnt. Time steps $T = 10000$, averaged over 15 runs and $\alpha = 0.3$).

4.2. Experiment 2: Continual Adaptation to Dynamic Toolsets

Objective. This experiment evaluates the agents' ability to adapt to a dynamically changing tool set over four distinct phases ($T_{phase} = 2500$ steps each, for a total of $T = 10000$ steps), involving tool addition, removal, and

the introduction of novel semantic types alongside new relevant queries. The setup tests the robustness and generalization capabilities crucial for lifelong learning. The environment cycles through three base query types ($\mathbf{q}_A, \mathbf{q}_B, \mathbf{q}_C$) for the first three phases, with a fourth query type ($\mathbf{q}_D$) introduced in Phase 4. Full phase details, including specific tool archetype assignments and query cycling, are in Appendix B.8. LinUCB-OneHot re-initializes it's model matrices (A, b) when K changes due to its feature space dependency on K. SC-LinUCB's model matrices and d_{sem} remain fixed. Both agents use an exploration parameter $\alpha = 0.5$ for this illustrative plot (sensitivity to α is explored in Appendix B.8). Results are averaged over $N_{runs} = 15$ independent seeds.

Results. Figure 2 (see figure 6 for corresponding reward plots) illustrates the average cumulative regret on a log scale. The performance of **SC-LinUCB (Semantic, orange line)**

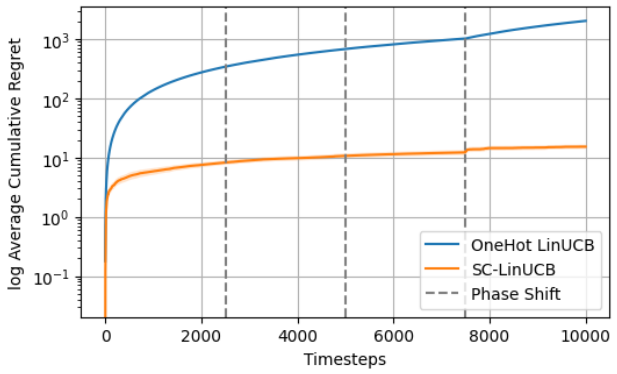

Figure 2. Average Cumulative Regret (log scale) for SC-LinUCB and LinUCB-OneHot in the continual adaptation experiment. Each phase is 2500 steps, changes indicated by dashed lines.

is remarkably robust. Its cumulative regret remains very low, consistently around 10^1 (approximately 10-20 units), across all four phases and 10000 time steps. Crucially, at the phase transitions (dashed vertical lines at $t = 2500, 5000, 7500$), its regret curve shows almost no perturbation. This demonstrates SC-LinUCB's ability to gracefully handle tool removal, leverage its existing semantic knowledge to quickly incorporate new tools with familiar semantic embeddings (Phase 3), and effectively learn about novel semantic types when new queries make them relevant (Phase 4), all without catastrophic forgetting or costly re-learning phases.

In stark contrast, **LinUCB-OneHot (Non-Semantic, blue line)** exhibits significantly higher regret and poor adaptation. Its regret climbs steeply, exceeding 10^3 by the end of the experiment. At each phase transition where the number of tools K changes, its regret curve shows a pronounced upward jump or a sharply increased slope. This is a direct consequence of its model matrices (A, b) being re-initialized due to the change in its feature space dimensionality ($d_{non-sem} = d_q + K + 1$), forcing it to largely relearn the value of tools from scratch for the new configuration.

These results strongly underscore the high cost of adaptation for a non-semantic agent in dynamic environments. SC-LinUCB's fixed-dimensional semantic feature space, combined with its capacity for semantic generalization, provides robust, efficient, and truly continual learning in the face of a changing action landscape.

5. SC in LLM Tool Orchestrators

Using and training LLM to orchestrate across O many tools can be done in a broad variety of methods. As previously mentioned it can be e.g. a classic policy mapping the query to an action (id or name) or a policy taking in the query alongside the semantic context. Crucially there is a variety of training regimes. A popular branch of methods used LLM fine-tuning techniques (full rank or low rank) using supervised fine-tuning (Prabhakar et al., 2025) with RL reasoning (Feng et al., 2025; Zhang et al., 2025) and algorithms like PPO or GRPO. All of these provide semantic context in their implementations. An alternative is to follow the recipe in (Dulac-Arnold et al., 2015) and train a hierarchical policy that predicts in the first step for a given query a text description of the action it wants to take (or an embedding of the action) and performs in the second stage nearest-neighbour search/ softmax over k-nearest neighbours to select the respective action. A third method is to rely on the in-context learning abilities.

We rigorously evaluate how SC impacts LLM in-context learning efficacy for sequential tool selection. We frame this as a multi-armed bandit (MAB) problem: an LLM agent learns to select optimal tools based on query context and interaction history presented via its prompt. Our investigation spans static and dynamic environments, assessing fundamental learning and adaptation.

5.1. Experimental Design

Our experimental design focuses on varying the semantic richness of action representations provided to the LLM. We consider four conditions:

Index Only (IO): Actions are presented as abstract, non-informative indices (e.g., "Action 1", "Action 2"). This baseline tests the LLM's ability to learn solely from correlations in the interaction history.

Name Only (NO): Actions are presented by their names (e.g., "Data Analyzer", "QuickTranslate"). This provides a concise signal, yet it is quite fragile.

Name + Description (ND): Actions are presented with their names and detailed functional descriptions, offering the richest semantic context.

Description Only (DO): Actions are presented as abstract non-informative indices together with detailed functional descriptions.

The LLM for all experiments is `Gemini 2.0 Flash`. Each experiment was conducted for multiple independent trials (5 for static environments, 7 for dynamic environments). The full prompt structure, LLM parameters (temperature 0.5, max output tokens $500 - 1500$), and detailed configurations of arms and queries are provided in appendix C.1. We report the average return over trials, where the expectation is taken over the stochasticity of rewards and LLM responses in figure 3. Average cumulative regrets are presented in figure 7.

We designed four distinct experimental scenarios: **Exp 1** *(fQfA)*: fixed query and fixed tools probes baseline in-context learning of best arm selection; **Exp 2** *(mQfA)*: varied queries with fixed tools test contextual generalisation; **Exp 3** *(fQmA)*: fixed query with evolving tools measures adaptation; **Exp 4** *(mQmA)*: both queries and tools shift, stressing full non-stationary robustness.

5.2. Results and Analysis

The experimental results, depicted by the average cumulative reward curves in 3, reveal a nuanced and significant impact of semantic context on the LLM's in-context learning and adaptation for tool selection. For the corresponding regret plot we refer the reader to figure 7 in the appendix. With the small action gap and the poor performance of the index only, the cumulative reward plot tells the semantic baselines better apart.

Static Environments (fQfA - Exp1; mQfA - Exp2): In environments with fixed action spaces (Exp1 and Exp2 panels in 3), providing richer semantic context generally leads to higher cumulative rewards. The ND condition (green solid line) and the NO condition (orange dash-dot line) both outperform the IO condition (blue dashed line). In Exp1 (fQfA), ND and NO perform very similarly, both achieving near-optimal reward accumulation, indicating that even names are sufficient for the single, repeated query. In Exp2 (mQfA), which involves multiple queries, ND maintains a slight edge over NO, suggesting that descriptions help differentiate tools more effectively as contextual complexity increases. IO consistently lags, demonstrating the LLM's difficulty in accumulating rewards without semantic cues to guide its choices.

Dynamic Environments (fQmA - Exp3; mQmA - Exp4): The introduction of non-stationarity through changing action spaces and/or queries highlights more complex interactions. In Experiment 3 (fQmA: fixed query, moving actions), the reward plot (3, Exp3 panel) shows that the ND condition adapts most effectively to the introduction of a superior tool ("E3_SuperCalc") around turn 17 (phase details in C.1.4). Its reward accumulation rate increases sharply after this point, surpassing NO. The NO condition also shows adaptation and reward growth but appears to either identify or commit to the superior tool with a delay or less consistency. The IO condition is slow in picking up the dynamic reward signal.

Experiment 4 (mQmA: moving queries and actions) presents the most striking results (3, Exp4 panel). In this highly dynamic scenario, the NO condition achieves the highest cumulative reward, notably outperforming ND. This intriguing outcome suggests that when both tasks and tools are frequently changing, the conciseness of tool names might offer an advantage in terms of agile decision-making or reduced risk of misinterpretation compared to longer descriptions. The ND condition still performs well and significantly better than IO, but the simpler NO representation appears more robust or efficient for the LLM in this specific rapidly evolving context. The IO condition again accumulates the least reward, underscoring its inadequacy in complex dynamic settings.

Synthesis of Findings: The results consistently demonstrate that the absence of semantic context (IO) severely limits the LLM's ability to effectively learn and accumulate rewards in these tool-selection bandit tasks. Providing semantic information is crucial. Rich descriptions (ND) are generally powerful, particularly for rapid optimal tool identification in static settings and for adapting to clear improvements (like a new superior tool for a known task) in dynamic environments. However, the superior performance of Names Only in the most complex, fully dynamic scenario (Exp4) is a key finding. It suggests a potential trade-off: while detailed descriptions offer depth, concise and sufficiently indicative names might facilitate greater agility or reduce the processing/interpretation burden on the LLM when faced with rapid and multifaceted environmental changes. This implies that the optimal level of semantic detail for action representation may not be monolithic but rather depend on the stability and complexity of the operational environment. It is also crucial to reiterate the finding of (Lin et al., 2024) that the naming of tools alters much more across developers than descriptions, making this approach more fragile.

Analysis of the Reasoning traces

Reasoning traces from Gemini 2.0 Flash (detailed examples in App. C.3) reveal *how* LLMs leverage semantic context.

```
"Reasoning: The query mentions \"sales
figures\" and \"growth pattern\", which
indicates numerical data and the need to
find trends. The Data Analyzer tool is
```

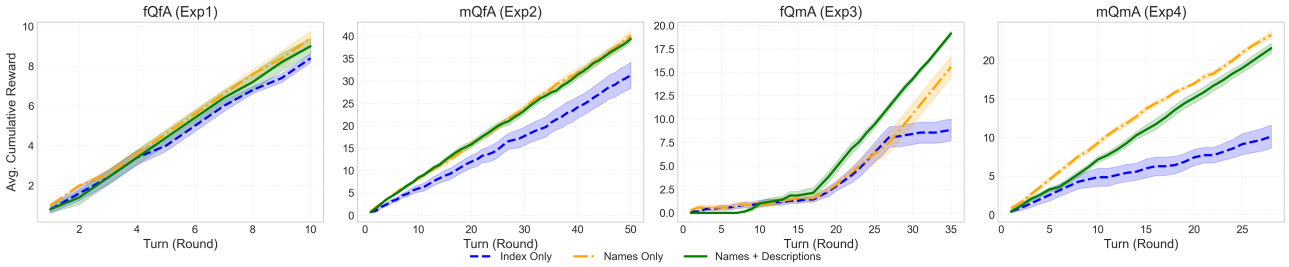

Figure 3. Semantic Context yields higher average return across Experiments 1-4. Subplot titles indicate: f=Fixed, m=Moving, Q=Queries, A=Actions. Shaded regions represent ± 1 standard error of the mean (SEM) across trials. Higher values indicate better performance. Note the varying x and y-axis scales.

```
specifically designed for processing
numerical data arrays to find trends. The
 other tools, Text Formatter and Image
Resizer, are not relevant to this query.
Therefore, the Data Analyzer is the most
likely to yield a reward.\n\n Chosen
Action: Data Analyzer"
```

Two main decision mechanisms are apparent: the reliance on previous experience (ICL exploration (Krishnamurthy et al., 2024)), and semantic matching. Particularly with ND and NO, the LLM often engages in a two-step semantic matching process: 1) analyse the user query to infer the abstract capability required; 2) match this inferred need against the semantic information of available tools, selecting the best aligner. This resembles the two-step action selection (Dulac-Arnold et al., 2015) where the policy maps first to a desired description (proto action) and subsequently selects the most appropriate match. For instance, for a sales growth query, the LLM with ND or NO typically identifies a "Data Analyser" by matching functionality. The richness of ND can lead to more nuanced initial alignments (Exp3), while the conciseness of NO might offer faster, if less precise, matching in dynamic scenarios (Exp4), potentially reducing cognitive load. This relies however on the concise tool naming ability of the tool creator. (Lin et al., 2024) raise that tool and argument naming is more user-sensitive than the function description, making the latter more robust. Crucially, NO and even more ND can enable LLM to prioritize semantic fit over immediate past negative rewards for the best tool. In contrast, IO relies solely on the ICL ability of LLM. The observed two-step reasoning provides a qualitative explanation for SC's quantitative benefits, suggesting that LLMs internalize descriptions for structured decision-making beyond simple index-based pattern matching.

5.3. Semantic Context for Scaling Action Space

In the previous subsection reasoning traces showed a two step of action description and action selection pattern. In all this experiments all tools and descriptions were part of the policy LLM context. To be practical, an orchestrator must scale to large amounts of tools. As the context of LLM runs naturally at some point out, we propose a "filter-then-reason-then-act" (FiReAct) pipeline. Pseudocode of FiReAct is provided in alg 1 and can be thought of as Tool-RAG version of RAG (Lewis et al., 2021).

Algorithm 1 The FiReAct Pipeline

Require: Embedding model ϕ, LLM policy π, query q_t, toolset $\mathcal{A}_t$, num candidates k

1: **Filter:** Retrieve candidate subset $\mathcal{A}_{cand} \subseteq \mathcal{A}_t$ of size k via semantic search using $\phi(q_t)$ and $\{\phi(a^i)|a^i \in \mathcal{A}_t\}$.

2: **Reason & Act:** Select final action $a_{selected} \in \mathcal{A}_{cand}$ using the LLM policy $\pi(q_t, C_S(\mathcal{A}_{cand}))$.

3: **return** $a_{selected}$

We instantiate this pipeline using a `text-embedding-004` retriever and a `gemini-2.0-flash` LLM policy. Firstly query and tools are embedded and the top k tools selected. These are feed (in the respective descriptive format (IO, ND,NO,DO) together with the query to the LLM policy. Based on this, the tool is selected. FireAct can be deployed at both test and train time. We demonstrate its usage at test time in a 0-shot pipeline on a challenging benchmark constructed from the XLAM dataset (Zhang et al., 2024), evaluating 100 queries against a corpus of over 10,000 tools. Figure 4 plots tool selection accuracy for three strategies: pure semantic retrieval ('top 1'), LLM-filtered reasoning ('top 5'), and exhaustive unfiltered reasoning ('all'). The results are unequivocal: without SC, performance is catastrophic. The IO condition yields 0-shot just random pulls, thus $(1/O)$ success rate.

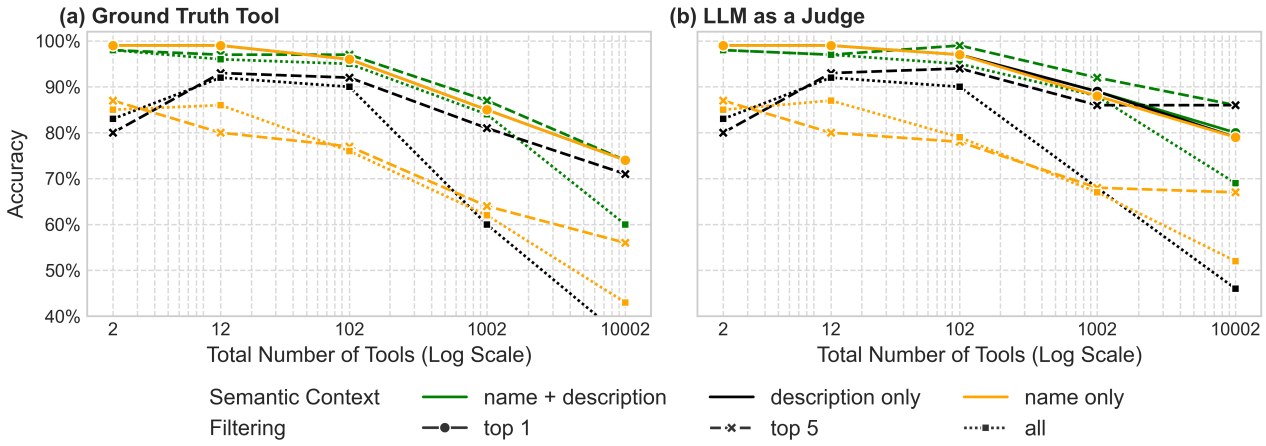

Figure 4. Semantic Context is essential for scalable tool selection with top 5 filtering followed by ND yields the strongest performance for large tool sets. Accuracy is plotted against the total number of tools (log scale). The left plot shows accuracy of identifying the ground truth tool, whereas the right plot uses an LLM as judge to evaluate the tool correctness.

Given SC's necessity, its quality is paramount. Rich ND context (green lines) consistently provides the highest accuracy across all methods, offering a distinct advantage over the weaker 'name only' and 'description only' signals. This shows that while any semantic signal is beneficial, more detailed information provides critical disambiguation power, especially as the number of distractor tools increases. Note however the superiour/ competitive performance of NO with N+D for up to 100 distractor tools. This demonstrates that more detailed semantic information provides critical disambiguation power in complex environments. However less SC (NO) is sometimes simpler, we hypothesize due to the smaller context window.

The most crucial finding, however, reveals how to best leverage SC at scale. While pure retrieval ('top 1') is powerful, its top-1 precision degrades as the tool space grows; with 10,000 distractors, the accuracy for 'name + description' context falls to $75\%(80\%$ with LLM Judge). The retriever's recall within the top 5 remains high, however, creating a vital opportunity for a reasoning step. By having the LLM re-rank these 'top 5' candidates, we restore accuracy to nearly 90%. This 15% accuracy gain validates the FiReAct pipeline as a robust, scalable strategy, where SC is the essential fuel for both initial filtering and final, high-fidelity reasoning.

6. Future Work and Conclusion

This paper establishes that explicit Semantic Context (SC), derived from action descriptions, is a powerful asset for efficient tool orchestration. We demonstrated this principle across two distinct paradigms: for linear contextual bandits, we proved that our SC-LinUCB framework enables more efficient learning and robust adaptation to dynamic action spaces compared to non-semantic baselines. We then showed that these principles translate directly to Large Language Models performing in-context learning. Our experiments confirmed that richer semantic context generally enhances tool selection and adaptability, and we demonstrated that our FiReAct pipeline leverages SC to make this approach scalable to thousands of tools.

Our work has limitations that open clear avenues for future investigation. Theoretically, key steps include developing sharp regret bounds for formally non-stationary toolsets ($\mathcal{A}_t$) and analyzing algorithmic robustness to noisy or imperfect semantic features. While our LLM experiments are indicative, they are specific to the chosen model and prompting strategies; direct theoretical guarantees for in-context tool learning remain a major open challenge. Empirically, extending our experiments to the fine-tuning of LLMs and developing end-to-end trainable retrieval-reasoning pipelines are promising directions.

In conclusion, by formalizing and demonstrating the "semantic advantage," our work shows that leveraging the inherent meaning of actions is a more effective strategy than treating them as opaque indices. The consistent benefits of SC observed across different learning frameworks—from linear models to large transformers—suggest that providing structured, semantic descriptions of actions is a valuable and generalizable design principle. This approach provides a principled path toward developing agents that are more sample-efficient, adaptive, and scalable when interacting with complex and evolving toolsets.

References

Abbasi-Yadkori, Y., Pál, D., and Szepesvári, C. (2011). Improved algorithms for linear stochastic bandits. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 2312–2320. Curran Associates, Inc.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. S. (2019). Learning action representations for reinforcement learning.

Chandak, Y., Theocharous, G., Nota, C., and Thomas, P. S. (2020). Lifelong learning with a changing action set. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 3373–3380. AAAI Press.

Chu, W., Li, L., Reyzin, L., and Schapire, R. E. (2011). Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 208–216. PMLR.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *CoRR*, abs/1512.07679.

Feng, J., Huang, S., Qu, X., Zhang, G., Qin, Y., Zhong, B., Jiang, C., Chi, J., and Zhong, W. (2025). Retool: Reinforcement learning for strategic tool use in llms.

Khetarpal, K., Riemer, M., Islam, R., and Precup, D. (2020). Towards continual reinforcement learning: A review and perspectives. *CoRR*, abs/2012.13490.

Krishnamurthy, A., Harris, K., Foster, D. J., Zhang, C., and Slivkins, A. (2024). Can large language models explore in-context?

Lattimore, T. and Szepesvári, C. (2020). *Bandit Algorithms*. Cambridge University Press.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 661–670. ACM.

Lin, Q., Wen, M., Peng, Q., Nie, G., Liao, J., Wang, J., Mo, X., Zhou, J., Cheng, C., Zhao, Y., Wang, J., and Zhang, W. (2024). Hammer: Robust function-calling for on-device language models via function masking.

Müller, R. and Pacchiano, A. (2022). Meta learning mdps with linear transition models. In Camps-Valls, G., Ruiz, F. J. R., and Valera, I., editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 5928–5948. PMLR.

Müller, R., Parker-Holder, J., and Pacchiano, A. (2020). Taming the herd: Multi-modal meta-learning with a population of agents. In *4th Lifelong Machine Learning Workshop at ICML 2020*.

Pathakota, P., Meisheri, H., and Khadilkar, H. (2023). Dct: Dual channel training of action embeddings for reinforcement learning with large discrete action spaces.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive apis. *CoRR*, abs/2305.15334.

Prabhakar, A., Liu, Z., Zhu, M., Zhang, J., Awalgaonkar, T., Wang, S., Liu, Z., Chen, H., Hoang, T., Niebles, J. C., Heinecke, S., Yao, W., Wang, H., Savarese, S., and Xiong, C. (2025). Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay.

Qian, C., Acikgoz, E. C., He, Q., Wang, H., Chen, X., Hakkani-Tür, D., Tur, G., and Ji, H. (2025). Toolrl: Reward is all tool learning needs.

Qin, Y., Hu, S., Lin, Y., Chen, W., Miao, N., Lu, K., Wang, Y., Wang, J., Wang, S., Hou, W., Li, S., Zhao, Y., Shen, X., Chen, Z., Li, Z., Li, W., Yi, J., Yang, C., Chen, Y., Liu, X., Ma, Z., Chen, X., Cui, G., Qi, F., Fei, Z., Su, J., Ou, Y., Liu, Z., and Sun, M. (2023). Toolllm: Facilitating large language models to master 16000+ real-world apis. *CoRR*, abs/2307.16789.

Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J., and Wen, J. (2024). Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*.

Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *CoRR*, abs/2302.04761.

Shi, Z., Wang, Y., Yan, L., Ren, P., Wang, S., Yin, D., and Ren, Z. (2025). Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. *arXiv preprint arXiv:2503.01763*.

Singh, J., Magazine, R., Pandya, Y., and Nambi, A. (2025). Agentic reasoning and tool integration for llms via reinforcement learning.

Tennenholtz, G. and Mannor, S. (2019). The natural language of actions. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6120–6128. PMLR.

Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., and Mannor, S. (2019). Learn what not to learn: Action elimination with deep reinforcement learning.

Zhang, J., Lan, T., Zhu, M., Liu, Z., Hoang, T., Kokane, S., Yao, W., Tan, J., Prabhakar, A., Chen, H., Liu, Z., Feng, Y., Awalgaonkar, T., Murthy, R., Hu, E., Chen, Z., Xu, R., Niebles, J. C., Heinecke, S., Wang, H., Savarese, S., and Xiong, C. (2024). xlam: A family of large action models to empower ai agent systems.

Zhang, S., Dong, Y., Zhang, J., Kautz, J., Catanzaro, B., Tao, A., Wu, Q., Yu, Z., and Liu, G. (2025). Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning.

A. Background

A.1. Notation at a Glance

Table 1. Notation at a glance

Symbol	Meaning
$\mathcal{A}_t$	action set available at round t of cardinality O_t
$\phi_t(a)$	semantic feature vector of action a
d_{sem}	similarity metric on $\mathcal{X}$
$\theta_\star$	unknown linear reward vector
V_t	design matrix at round t

A.2. Semantic Context MDP

Definition A.1 (Semantic Context MDP, SC-MDP)**.** *An **SC-MDP** describes sequential decision-making with a fixed toolset $\mathcal{A}_{avail}$ and its corresponding fixed Semantic Action Context $C_S(\mathcal{A}_{avail})$. It is an MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where: The state $s_t \in \mathcal{S}$ is typically (h_t, q_t), representing history and current query. The action space $\mathcal{A}$ consists of choices $(a_j, args(a_j))$ where $a_j \in \mathcal{A}_{avail}$. The policy $\pi(a_t|s_t)$ implicitly utilizes the fixed $C_S(\mathcal{A}_{avail})$ (which defines this specific MDP environment) to select a_t. Transitions $P(s_{t+1}|s_t, a_t)$ and rewards $R(s_t, a_t)$ are standard. Tool execution yields an output o_t, forming part of h_{t+1}.*

Definition A.2 (Lifelong Semantic Context MDP, LSC-MDP)**.** *An **LSC-MDP** models scenarios with a dynamically changing tool set $\mathcal{A}_t$. It is an MDP $(\mathcal{S}_{LSC}, \mathcal{A}_{LSC}, P_{LSC}, R_{LSC}, \gamma)$, where the state $s_t \in \mathcal{S}_{LSC}$ is $(h_t, q_t, C_S(\mathcal{A}_t))$, explicitly includes the time-dependent SC $C_S(\mathcal{A}_t)$ that changes as the tool set $\mathcal{A}_t$ evolves. The action space $\mathcal{A}_{LSC}(s_t)$ comprises choices $(a_j, args(a_j))$ where $a_j \in \mathcal{A}_t$. The policy is $\pi(a_t|s_t)$. Transition dynamics $P_{LSC}(s_{t+1}|s_t, a_t)$ determine the next query q_{t+1} and, crucially, the next available toolset $\mathcal{A}_{t+1}$ (and thus $C_S(\mathcal{A}_{t+1})$).*

B. Appendix Semantic Context LINUCB

B.1. Formal Assumptions

For the linear bandit setting we have the following standard assumptions.

Assumption B.1 (Contextual Linear Bandit Setting (Restated))**.** *Over T timesteps, $t \in \{1, \dots, T\}$:*

1. *A context s_t is observed, from which a d_q-dimensional query embedding $\mathbf{q}_t = q(s_t)$ is derived.*

2. *The agent selects an action (tool) a_t from a fixed set of K tools $\mathcal{A} = \{a_1, \dots, a_K\}$.*

3. *Each tool $a_j \in \mathcal{A}$ has a d_{desc}-dimensional semantic description embedding $\phi_j = \phi(D_{a_j})$.*

4. *For each context-tool pair (q_t, a_j), a d-dimensional feature vector $\mathbf{x}_{t,j} = x(\mathbf{q}_t, \phi_j)$ is constructed. We assume $\|\mathbf{x}_{t,j}\|_2 \leq L_x$.*

5. *The expected reward is linear in these features: $\mathbb{E}[R_t(\mathbf{x}_{t,j})|\mathbf{x}_{t,j}] = \mathbf{x}_{t,j}^T \theta^*$ for an unknown true parameter vector $\theta^* \in \mathbb{R}^d$. We assume $\|\theta^*\|_2 \leq S_\theta$.*

6. *Observed rewards are $R_t(\mathbf{x}_{t,j}) = \mathbf{x}_{t,j}^T \theta^* + \eta_{t,j}$, where $\eta_{t,j}$ is conditionally σ-subGaussian noise: $\mathbb{E}[\eta_{t,j}|\mathbf{x}_{t,j}] = 0$ and $\mathbb{E}[e^{\lambda \eta_{t,j}}|\mathbf{x}_{t,j}] \leq e^{\lambda^2 \sigma^2 / 2}$ for all $\lambda \in \mathbb{R}$.*

B.2. SC-LinUCB Algorithm Detail

Algorithm 2 SC-LinUCB (Shared Model) - Appendix Version

Require: Exploration parameter $\alpha > 0$, regularization $\lambda_{reg} > 0$.
Initialize $\mathbf{A} = \lambda_{reg}\mathbf{I}_d$, $\mathbf{b} = \mathbf{0}_d$.
for $t = 1, \ldots, T$ **do**
  Observe query $\mathbf{q}_t$.
  For each tool $a_j \in \mathcal{A}$ (with semantic embedding ϕ_j), construct feature vector $\mathbf{x}_{t,j} = x(\mathbf{q}_t, \phi_j)$.
  Compute $\mathbf{A}^{-1}$.
  Compute $\hat{\theta}_t = \mathbf{A}^{-1}\mathbf{b}$.
  For each tool $a_j \in \mathcal{A}$:
    $s_{t,j} \leftarrow \sqrt{\mathbf{x}_{t,j}^T \mathbf{A}^{-1} \mathbf{x}_{t,j}}$
    $p_{t,j} \leftarrow \mathbf{x}_{t,j}^T \hat{\theta}_t + \alpha s_{t,j}$
  Choose $a_t = \arg\max_{j \in \{1,\ldots,K\}} p_{t,j}$ (break ties randomly).
  Let $\mathbf{x}_t^{chosen} = \mathbf{x}_{t,a_t}$.
  Play tool a_t, observe reward $R_t(\mathbf{x}_t^{chosen})$.
  $\mathbf{A} \leftarrow \mathbf{A} + \mathbf{x}_t^{chosen}(\mathbf{x}_t^{chosen})^T$.
  $\mathbf{b} \leftarrow \mathbf{b} + R_t(\mathbf{x}_t^{chosen})\mathbf{x}_t^{chosen}$.
end for

B.3. Standard Lemmas and Proof for Generic LinUCB Regret

Theorem B.2 (Confidence Set for θ^*, Theorem 2 from Abbasi-Yadkori et al. (2011))**.** *Under Assumption 3.1, let $\delta \in (0, 1)$ and $\lambda_{reg} > 0$. Define*

$$\alpha_t'(\delta) := \sigma\sqrt{2\log(1/\delta) + d\log\left(1 + \frac{tL_x^2}{\lambda_{reg} d}\right)} + \sqrt{\lambda_{reg}} S_\theta$$

(This form of α is closer to the direct statement in Abbasi-Yadkori et al., Theorem 2, which uses $\log(\det(\mathbf{A}_t)/\det(\lambda_{reg}\mathbf{I})) \leq d\log(1 + tL_x^2/(\lambda_{reg} d))$). Then, with probability at least $1-\delta$, for all $t \geq 1$, θ^* lies in the set $C_t = \{\theta \in \mathbb{R}^d : \|\hat{\theta}_t - \theta\|_{\mathbf{A}_t} \leq \alpha_t'(\delta)\}$. This implies that for any $\mathbf{x} \in \mathbb{R}^d$ with $\|\mathbf{x}\|_2 \leq L_x$, $|\mathbf{x}^T(\hat{\theta}_t - \theta^*)| \leq \alpha_t'(\delta)\sqrt{\mathbf{x}^T \mathbf{A}_t^{-1} \mathbf{x}}$. For the main paper, we use a slightly simplified $\alpha \geq \alpha_T'(\delta)$ for clarity, which might incorporate a $\log K$ term for uniform convergence over arms at each step if not absorbed into δ.

Proof. See proof of theorem 2 from Abbasi-Yadkori et al. (2011) for full derivation. □

Lemma B.3 (Elliptical Potential Lemma, Lemma 11 from Abbasi-Yadkori et al. (2011))**.** *Let $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^d$ be a sequence of feature vectors such that $\|\mathbf{x}_t\|_2 \leq L_x$. Let $\mathbf{A}_t = \lambda_{reg}\mathbf{I}_d + \sum_{j=1}^{t-1} \mathbf{x}_j \mathbf{x}_j^T$. Then,*

$$\sum_{t=1}^{T} \min(1, \mathbf{x}_t^T \mathbf{A}_t^{-1} \mathbf{x}_t) \leq 2d\log\left(1 + \frac{TL_x^2}{\lambda_{reg} d}\right)$$

Proof. See proof of Lemma 11 from Abbasi-Yadkori et al. (2011). □

B.4. Elleptical potential lemma

We restate and proof the elleptical potential lemma:

Lemma B.4 (Elliptical Potential Lemma, Lemma 11 from Abbasi-Yadkori et al. (2011))**.** *Let $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^d$ be a sequence of feature vectors such that $\|\mathbf{x}_t\|_2 \leq L_x$. Let $\mathbf{A}_t = \lambda_{reg}\mathbf{I}_d + \sum_{j=1}^{t-1} \mathbf{x}_j \mathbf{x}_j^T$. Then,*

$$\sum_{t=1}^{T} \min(1, \mathbf{x}_t^T \mathbf{A}_t^{-1} \mathbf{x}_t) \leq 2d\log\left(1 + \frac{TL_x^2}{\lambda_{reg} d}\right)$$

If $\lambda_{reg} \geq L_x^2$*, then* $\mathbf{x}_t^T \mathbf{A}_t^{-1} \mathbf{x}_t \leq \mathbf{x}_t^T (\lambda_{reg} \mathbf{I}_d)^{-1} \mathbf{x}_t = \frac{\|\mathbf{x}_t\|^2}{\lambda_{reg}} \leq \frac{L_x^2}{\lambda_{reg}} \leq 1$*, so the* $\min(1, \cdot)$ *can be removed. For a general* λ_{reg}*, the bound still holds with the* min.

Proof. See Lemma 11 and Appendix A.3 in Abbasi-Yadkori et al. (2011). □

B.5. Detailed Argument for Theorem 3.2 (Advantage of SC-LinUCB)

Theorem 3.2 posits that SC-LinUCB achieves lower regret than LinUCB-NS by enabling more efficient exploration and generalization through its semantic features. We elaborate on the two main mechanisms:

1. More Parsimonious Effective Model (Relating to d): The regret bound for LinUCB scales roughly with d, the feature dimensionality. For SC-LinUCB, features $\mathbf{x}_{t,j}^{(sem)} = [\mathbf{q}_t; \phi_j; \text{sim}(\mathbf{q}_t, \phi_j); 1]$ have dimension $d_{sem} = d_q + d_{desc} + 1 + 1$. For LinUCB-NS with one-hot tool encodings, $\mathbf{x}_{t,j}^{(non-sem)} = [\mathbf{q}_t; \mathbf{e}_j; 1]$ has dimension $d_{non-sem} = d_q + K + 1$.

Assumption B.1 implies that the true reward function $f^*(\mathbf{q}_t, \phi_j)$ depends on shared semantic properties encoded in ϕ_j and their interaction with $\mathbf{q}_t$. If the diversity of K tools can be meaningfully captured by d_{desc}-dimensional semantic embeddings such that $d_{desc} \ll K$ (e.g., tools fall into fewer semantic archetypes than K, or their reward-relevant variations are low-dimensional), then d_{sem} can be substantially smaller than $d_{non-sem}$. SC-LinUCB learns a single parameter vector $\hat{\theta}_{sem} \in \mathbb{R}^{d_{sem}}$. This vector effectively models the utility of semantic *attributes* (dimensions of $\mathbf{q}_t$, dimensions of ϕ_j, and their similarity) and how they combine to predict reward. This model is shared across all K tools. LinUCB-NS, on the other hand, needs to learn parameters in $\hat{\theta}_{non-sem} \in \mathbb{R}^{d_{non-sem}}$ where K of these dimensions (from $\mathbf{e}_j$) are dedicated to capturing the unique identity and behavior of each tool. If there is underlying semantic redundancy across tools that LinUCB-NS cannot exploit, it is effectively learning a higher-dimensional model than necessary. Thus, if $d_{sem} < d_{non-sem}$ and both feature sets achieve a comparable quality of linear approximation (i.e., $\sigma_{eff,sem} \approx \sigma_{eff,non-sem}$), the d factor in the regret bound directly favors SC-LinUCB. This represents a reduction in the complexity of the parameter space to be learned.

2. Faster Reduction of Uncertainty for Semantically Similar Options (Relating to $\sum s_{t,a_t}$): The instantaneous regret r_t is bounded by $2\alpha s_{t,a_t} = 2\alpha\sqrt{\mathbf{x}_{t,a_t}^T \mathbf{A}_t^{-1} \mathbf{x}_{t,a_t}}$. The cumulative regret depends on the sum of these exploration terms. Consider the update to the covariance matrix $\mathbf{A}_{t+1} = \mathbf{A}_t + \mathbf{x}_t \mathbf{x}_t^T$. The inverse $\mathbf{A}_{t+1}^{-1}$ shrinks based on the direction of $\mathbf{x}_t$. The exploration term $s_{t',j}^2 = \mathbf{x}_{t',j}^T \mathbf{A}_{t+1}^{-1} \mathbf{x}_{t',j}$ for any arm j at a future step t' will decrease more significantly if $\mathbf{x}_{t',j}$ has a substantial component along the direction of $\mathbf{x}_t$ (the chosen arm's features at time t).

For SC-LinUCB, if tool a_a is chosen at time t (with features $\mathbf{x}_{t,a}^{(sem)}$), the update to $\mathbf{A}_{sem}$ reflects increased certainty along the semantic dimensions present in $\mathbf{x}_{t,a}^{(sem)}$. Now, consider another tool a_b. If a_b is semantically similar to a_a with respect to context $\mathbf{q}_t$ (or a similar context $\mathbf{q}_{t'}$), then their feature vectors $\mathbf{x}_{t,a}^{(sem)}$ and $\mathbf{x}_{t',b}^{(sem)}$ will share many active semantic components (e.g., similar ϕ components, similar interaction terms). Consequently, the exploration term $s_{t',b}^{(sem)}$ for tool a_b will also be reduced due to the information gained from pulling a_a. The agent effectively learns about a "semantic neighborhood" of tools with each pull.

For LinUCB-NS, the feature vectors $\mathbf{x}_{t,a}^{(non-sem)} = [\mathbf{q}_t; \mathbf{e}_a; 1]$ and $\mathbf{x}_{t,b}^{(non-sem)} = [\mathbf{q}_t; \mathbf{e}_b; 1]$ (for $a \neq b$) have orthogonal tool-identity components $\mathbf{e}_a$ and $\mathbf{e}_b$. An update from pulling a_a (involving $\mathbf{e}_a$) primarily reduces uncertainty associated with $\mathbf{e}_a$ and its interaction with $\mathbf{q}_t$. It has minimal effect on reducing the uncertainty associated with the distinct orthogonal direction $\mathbf{e}_b$. Thus, LinUCB-NS learns little about a_b's specific utility from pulling a_a, even if a_a and a_b are semantically very similar.

This implies that SC-LinUCB can "cross off" or gain confidence about larger regions of the (context $\times$ semantic tool property) space with each observation. As a result, the sum of exploration terms $\sum_{t=1}^T s_{t,a_t}$ is expected to be smaller for SC-LinUCB compared to LinUCB-NS over T steps, as it requires fewer "distinctly exploratory" pulls to identify good actions across the spectrum of contexts and tools. While the Elliptical Potential Lemma (Lemma B.3) bounds $\sum s_{t,a_t}^2$ by $O(d \log T)$ for both, the actual sequence of s_{t,a_t} values chosen by SC-LinUCB can be smaller on average due to this generalization, leading to a tighter sum for $\sum s_{t,a_t}$ when applying Cauchy-Schwarz.

Combining a potentially smaller d_{sem} with a more efficient exploration dynamic (leading to a smaller effective sum of exploration bonuses), SC-LinUCB achieves lower cumulative regret.

B.6. SC-LinUCB in the continual setting

Beyond efficiency with a fixed set of tools, SC-LinUCB's semantic feature design offers significant advantages in continual learning scenarios where the set of available tools $\mathcal{A}_t$ (and thus its size K_t) changes over time. This is a critical capability for agents in evolving environments.

Consider a setting with phases, where within each phase p, the toolset $\mathcal{A}^{(p)}$ is fixed, but it can change between phases (e.g., $\mathcal{A}^{(p+1)} = (\mathcal{A}^{(p)} \setminus \mathcal{A}_{removed}) \cup \mathcal{A}_{added}$).

Theorem B.5 (Low-Cost Adaptation of SC-LinUCB to Dynamic Toolsets)**.** *Let SC-LinUCB use semantic features $\mathbf{x}^{(sem)}$ of fixed dimension d_{sem} and LinUCB-NS use one-hot features $\mathbf{x}^{(non-sem)}$ of dimension $d_{non-sem}(K_t) = d_q + K_t + 1$. When the set of available tools changes from $\mathcal{A}^{(p)}$ (size $K^{(p)}$) to $\mathcal{A}^{(p+1)}$ (size $K^{(p+1)}$):*

1. ***SC-LinUCB (Semantic):***
   - *Its feature dimension d_{sem} remains constant.*
   - *Its learned parameter vector $\hat{\theta}^{(p)}_{sem}$ (from phase p) and covariance matrix $\mathbf{A}^{(p)}_{sem}$ remain valid and are directly carried over to phase $p+1$.*
   - *For any newly added tool $a_{new} \in \mathcal{A}_{added}$ with semantic embedding ϕ_{new}, SC-LinUCB can immediately compute its feature vector $\mathbf{x}^{(sem)}_{q,new}$ and estimate its utility using the existing $\hat{\theta}^{(p)}_{sem}$, yielding an informed initial UCB score.*
   - *The "cost of adaptation" is primarily the exploration required for new semantic aspects introduced by $\mathcal{A}_{added}$ that were not sufficiently covered by $\hat{\theta}^{(p)}_{sem}$. If new tools are semantically similar to previously seen optimal tools, adaptation is very fast.*
2. ***LinUCB-NS (Non-Semantic Baseline):***
   - *If $K^{(p+1)} \neq K^{(p)}$, its feature dimension $d_{non-sem}(K_t)$ changes. This necessitates a change in its parameter vector $\hat{\theta}_{non-sem}$ and matrices $\mathbf{A}_{non-sem}, \mathbf{b}_{non-sem}$.*
   - *Common strategies for LinUCB-NS include: (a) Full Re-initialization: $\mathbf{A}_{non-sem}$ and $\mathbf{b}_{non-sem}$ are reset. The agent effectively relearns from scratch for the new toolset $\mathcal{A}^{(p+1)}$, incurring regret similar to starting a new bandit problem of size $K^{(p+1)}$. (b) Heuristic Adaptation: Attempting to adapt $\mathbf{A}_{non-sem}, \mathbf{b}_{non-sem}$ (e.g., adding/removing rows/columns) is complex and typically still treats new tool IDs as completely unknown entities requiring extensive exploration.*
   - *For any newly added tool a_{new}, LinUCB-NS has no prior information derived from other tools about its utility, as its one-hot encoding is orthogonal to others.*
   - *The "cost of adaptation" involves significant relearning for the entire (or substantial parts of) the new toolset.*

Consequently, over a sequence of phases with changing toolsets, SC-LinUCB is expected to achieve substantially lower cumulative regret than LinUCB-NS due to its fixed-dimensional semantic representation, knowledge transfer via $\hat{\theta}_{sem}$, and ability to gracefully incorporate or ignore tools based on their semantic features without model restructuring.

Proof Sketch for Theorem B.5. This theorem's argument builds on the properties of the agents and the implications of Theorem **??** applied piecewise.

For SC-LinUCB: The feature space $\mathbb{R}^{d_{sem}}$ and the parameter vector θ^*_{sem} are defined over semantic properties, not tool identities or the count K_t. Thus, the learned model $(\hat{\theta}_{sem}, \mathbf{A}_{sem})$ retains its validity and utility when the set of available tools $\mathcal{A}_t$ changes.

- *Tool Addition:* When a_{new} (with ϕ_{new}) is added, SC-LinUCB calculates $\mathbf{x}^{(sem)}_{q,new}$ and its UCB score using the current $\hat{\theta}_{sem}$ and $\mathbf{A}_{sem}$. If ϕ_{new} aligns semantically with query features for which $\hat{\theta}_{sem}$ has learned high weights, a_{new} will be explored efficiently. The exploration cost is for resolving uncertainty about this specific $\mathbf{x}^{(sem)}_{q,new}$ within the existing learned model structure. No part of the model needs to be "resized" or "reset."
- *Tool Removal:* If $a_{removed}$ is removed, SC-LinUCB simply no longer considers it for selection. Its learned $\hat{\theta}_{sem}$ and $\mathbf{A}_{sem}$ (which contain information from past pulls of $a_{removed}$) remain valid for evaluating the remaining tools.

The regret within any phase p where $\mathcal{A}^{(p)}$ is fixed is governed by Theorem **??** with $d = d_{sem}$. The transitions between phases incur minimal structural cost.

For LinUCB-NS (OneHot): The feature space $\mathbb{R}^{d_{non-sem}(K_t)}$ explicitly depends on the current number of tools K_t via the one-hot encodings $\mathbf{e}_j \in \mathbb{R}^{K_t}$.

- *Tool Addition (K increases):* $d_{non-sem}$ increases. The matrices $\mathbf{A}_{non-sem}$ and $\mathbf{b}_{non-sem}$ must be expanded. The new dimensions corresponding to the new tool ID have no prior history. Effectively, the agent must learn about this new tool's interaction with all query types from scratch. If the agent fully resets $\mathbf{A}_{non-sem}, \mathbf{b}_{non-sem}$ (as done in our Experiment 2 for a clear baseline), it starts a new learning problem with regret $\tilde{O}(d_{non-sem}(K_{new})\sqrt{T_{phase}})$. Even with more sophisticated matrix adaptation, the components of $\theta^*_{non-sem}$ relevant to the new tool are unknown.
- *Tool Removal (K decreases):* $d_{non-sem}$ decreases. The agent might discard rows/columns from $\mathbf{A}_{non-sem}, \mathbf{b}_{non-sem}$. This is less detrimental than addition if no reset occurs, but the overall problem structure for its features has changed.

The key issue is that LinUCB-NS's learned knowledge is tied to specific tool indices. If these indices change, or new ones appear, extensive relearning is often needed for those affected dimensions. The strategy of re-initializing $\mathbf{A}, \mathbf{b}$ upon change in K (as implemented for LinUCB-OneHot in our Experiment 2) represents a clear case where it incurs a full bandit learning cost for the new configuration.

Comparing Adaptation Costs: The "cost" can be seen as the additional regret incurred during a phase transition compared to an oracle that was already adapted. For SC-LinUCB, this cost is low because $\hat{\theta}_{sem}$ provides immediate, semantically-informed estimates for new tools, and its structure is stable. For LinUCB-NS (with resets on K change), this cost is high, equivalent to the initial regret of a new bandit problem. Thus, over multiple phases of toolset changes, the cumulative regret of SC-LinUCB will be substantially lower due to these significantly reduced adaptation costs at phase boundaries, on top of its potential intra-phase efficiency from Theorem 3.2. □

B.7. Experiment 1: Detailed Setup and Full Results for Intra-Episode Efficiency

This section provides further details for Experiment 1, which evaluates the intra-episode efficiency of SC-LinUCB with semantic features against LinUCB-OneHot with non-semantic features in a multi-context toy environment.

Environment Design. The environment is a contextual bandit task designed to highlight the benefits of semantic generalization.

- **Timesteps (T):** Each experimental run consists of $T = 10000$ timesteps.
- **Tools (K):** There are $K = 6$ tools available throughout each run.
- **Tool Semantic Archetypes and Embeddings (ϕ_j):** Tools are designed around $N_{arch} = 3$ underlying semantic archetypes. Each tool a_j is assigned one of these archetypes. Its $d_{tool_sem} = 2$ dimensional toy semantic embedding ϕ_j is generated by taking the corresponding archetype vector and adding Gaussian noise with zero mean and standard deviation $\sigma_{emb_noise} = 0.05$. This noise is re-generated for each of the N_{runs} independent experimental trials to ensure robustness of results to minor variations in embeddings. The archetype vectors are:
  - Archetype 1 (ϕ_{arch1}): $[0.9, 0.1]^T$ (2 tools assigned this archetype)
  - Archetype 2 (ϕ_{arch2}): $[0.1, 0.9]^T$ (2 tools assigned this archetype)
  - Archetype 3 (ϕ_{arch3}): $[-0.7, -0.7]^T$ (2 tools assigned this archetype, replacing the previous 1 'type3' and 1 'noise' for more symmetry)
- **Queries/Contexts ($\mathbf{q}_t$):** There are $N_Q = 3$ distinct query types, each represented by a $d_q = 2$ dimensional toy embedding. These queries cycle periodically every N_Q timesteps (i.e., $q_A, q_B, q_C, q_A, q_B, q_C, \dots$). The query embeddings are:
  - Query A ($\mathbf{q}_A$): $[1.0, 0.2]^T$, designed to align best with Tool Archetype 1.
  - Query B ($\mathbf{q}_B$): $[0.2, 1.0]^T$, designed to align best with Tool Archetype 2.
  - Query C ($\mathbf{q}_C$): $[-0.8, -0.8]^T$, designed to align best with Tool Archetype 3.

- **Reward Function (R_t):** The reward $R_t \in \{0, 1\}$ is stochastic, drawn from a Bernoulli distribution. The success probability $P(\text{success}|\mathbf{q}_t, \phi_j)$ is determined by the semantic alignment between the current query $\mathbf{q}_t$ and the chosen tool's embedding ϕ_j. Specifically:

$$P(\text{success}) = \text{clip}(P_{base} + C_{sim} \cdot (\mathbf{q}_t^T \phi_j) + B_{align}, P_{min}, P_{max})$$

  where $P_{base} = 0.45$ is a base success rate, $C_{sim} = 0.40$ scales the dot product similarity, and $B_{align} = 0.25$ is a bonus awarded if the chosen tool's true archetype matches the current query's preferred archetype. Probabilities are clipped to $[P_{min} = 0.05, P_{max} = 0.95]$. This structure ensures that tools whose semantic embeddings align well with the current query, especially those of the preferred archetype, have a higher expected reward.

Agent Configurations. Both SC-LinUCB and LinUCB-OneHot are instances of the stanard LinUCB algorithm differing only in their feature construction:

- **SC-LinUCB (Semantic):** Uses $d_{sem} = d_q + d_{tool_sem} + 1(\text{similarity}) + 1(\text{bias}) = 2 + 2 + 1 + 1 = 6$ dimensional features: $\mathbf{x}_{t,j}^{(sem)} = [\mathbf{q}_t; \phi_j; \mathbf{q}_t^T \phi_j; 1]$.
- **LinUCB-OneHot (Non-Semantic Baseline):** Uses $d_{non-sem} = d_q + K + 1(\text{bias}) = 2 + 6 + 1 = 9$ dimensional features: $\mathbf{x}_{t,j}^{(non-sem)} = [\mathbf{q}_t; \mathbf{e}_j; 1]$, where $\mathbf{e}_j$ is the one-hot encoding for tool a_j.

Both agents use $\lambda_{reg} = 1.0$. We evaluate exploration parameters $\alpha \in \{0.3, 0.5, 1.0\}$.

Evaluation Metrics. Results are averaged over $N_{runs} = 15$ independent Monte Carlo runs. We report:

1. **Average Cumulative Reward:** $\frac{1}{N_{runs}} \sum_{run=1}^{N_{runs}} \sum_{t=1}^{T} R_t^{(run)}$.
2. **Average Cumulative Regret:** $\frac{1}{N_{runs}} \sum_{run=1}^{N_{runs}} \sum_{t=1}^{T} (\mathbb{E}[R|\mathbf{q}_t, a_t^*] - \mathbb{E}[R|\mathbf{q}_t, a_t^{(run)}])$. Here, $\mathbb{E}[R|\mathbf{q}_t, a]$ is the true expected reward (success probability) of tool a for query $\mathbf{q}_t$, and a_t^* is the tool with the maximum expected reward for $\mathbf{q}_t$. This uses expected instantaneous regret for smoother non-decreasing cumulative regret curves.

Full Experimental Results. Figure 5 shows both the average cumulative reward and average cumulative regret on logarithmic y-axes for all tested α values.

The results clearly indicate the superiority of SC-LinUCB. In the reward plot (left), SC-LinUCB variants (particularly with $\alpha = 1.0$, purple dashed line) accumulate substantially more reward over time compared to LinUCB-OneHot variants. The log scale emphasizes the sustained higher rate of reward collection.

The regret plot (right) offers the most striking comparison. SC-LinUCB agents maintain an extremely low cumulative regret (primarily between 10^0 and 10^1), indicating rapid convergence to near-optimal policies for the cycling contexts. The SC-LinUCB (Semantic) $\alpha = 0.3$ (blue solid line) shows the lowest regret overall. In stark contrast, all LinUCB-OneHot variants incur regret that is orders of magnitude higher, reaching 10^3. While their regret curves are sublinear (indicating learning), their inefficiency compared to SC-LinUCB is evident. The LinUCB-OneHot agent with $\alpha = 1.0$ (brown solid line) performs best among the non-semantic baselines but is still vastly outperformed.

These empirical findings strongly corroborate our theoretical analysis (Theorem 3.2). The ability of SC-LinUCB to generalize across tools and contexts using a compact semantic feature space ($d_{sem} = 6$) leads to substantially more efficient learning than LinUCB-OneHot, which must learn more independently for each tool ID within its higher-dimensional feature space ($d_{non-sem} = 9$). The semantic features provide a powerful inductive bias that aligns with the problem structure, reducing the effective complexity faced by the learning algorithm.

B.8. Experiment 2: Detailed Results for Continual Adaptation with Varying Exploration

This section provides the full results for Experiment 2, which evaluates the continual adaptation capabilities of SC-LinUCB (Semantic) and LinUCB-OneHot (Non-Semantic) in an environment with dynamically changing toolsets. We present a sensitivity analysis with respect to the exploration parameter $\alpha \in \{0.3, 0.5, 1.0\}$.

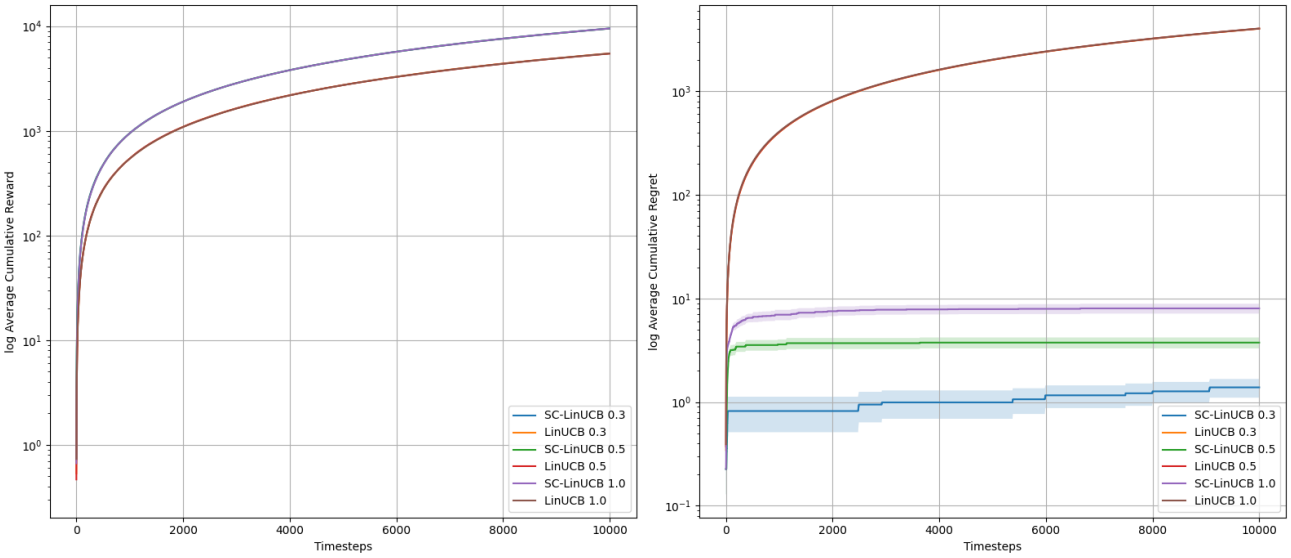

Figure 5. Full results for Experiment 1: SC-LinUCB (Semantic) vs. LinUCB-OneHot (Non-Semantic) in the multi-context toy environment ($T = 10000$, 15 runs). Left: Average Cumulative Reward (log scale). Right: Average Cumulative Regret (log scale). Different line styles/colors within agent types correspond to $\alpha \in \{0.3, 0.5, 1.0\}$.

Experimental Setup Recap. The environment consists of four distinct phases, each lasting $T_{phase} = 2500$ timesteps (total $T = 10000$). The set of available tools (K) and active query types (N_Q) evolve across these phases, involving tool addition (of both semantically familiar and novel types), tool removal, and the introduction of new query types corresponding to novel tools.

- **Phase 1 ($K = 4, N_Q = 3$):** Initial tools: $\{a_{A1}, a_{A2}(\text{type1}); a_{B1}, a_{B2}(\text{type2})\}$. Queries: $\mathbf{q}_A, \mathbf{q}_B, \mathbf{q}_C$.
- **Phase 2 ($K = 3, N_Q = 3$):** Tool a_{A2} (type1) removed. (Starts at $t = 2500$)
- **Phase 3 ($K = 4, N_Q = 3$):** New tool a_{A3} (type1, semantically similar to a_{A1}) added. (Starts at $t = 5000$)
- **Phase 4 ($K = 5, N_Q = 4$):** New tool a_{D1} (novel semantic type4) added; query $\mathbf{q}_D$ (aligning with type4) becomes active. (Starts at $t = 7500$)

LinUCB-OneHot re-initializes its model matrices (A, b) when K changes. SC-LinUCB's core model matrices and semantic feature dimension ($d_{sem} = 6$) remain fixed. Toy embeddings and the reward function are as described in Appendix C.1.2 (or a dedicated Exp2 setup section if it differs significantly). All results are averaged over $N_{runs} = 15$ independent seeds.

Results with Varying Alphas. Figure 6 displays the average cumulative reward (left, log scale) and average cumulative regret (right, log scale) for both SC-LinUCB and LinUCB-OneHot across the three tested values of α.

Cumulative Reward Analysis (Figure 6, Left): SC-LinUCB variants consistently achieve higher cumulative rewards than LinUCB-OneHot variants across all tested α values. For SC-LinUCB, $\alpha = 1.0$ (purple dashed line) yields the highest overall reward, suggesting that with strong semantic features, a reasonably high level of exploration can be beneficial for maximizing long-term reward, even in a changing environment. For LinUCB-OneHot, $\alpha = 1.0$ (brown dashed line) is also its best configuration, but it still lags significantly behind all SC-LinUCB variants. The SC-LinUCB curves maintain a steadier rate of reward accumulation across phase transitions, whereas the LinUCB-OneHot curves show more pronounced slowdowns or changes in slope, indicative of their relearning periods.

Cumulative Regret Analysis (Figure 6, Right): The regret plot starkly illustrates the advantages of SC-LinUCB.

- **SC-LinUCB (Semantic):** All variants (blue $\alpha = 0.3$, green $\alpha = 0.5$, purple $\alpha = 1.0$) maintain exceptionally low cumulative regret, generally staying within the 10^0 to 10^1 range over 10000 steps. The phase transitions cause only

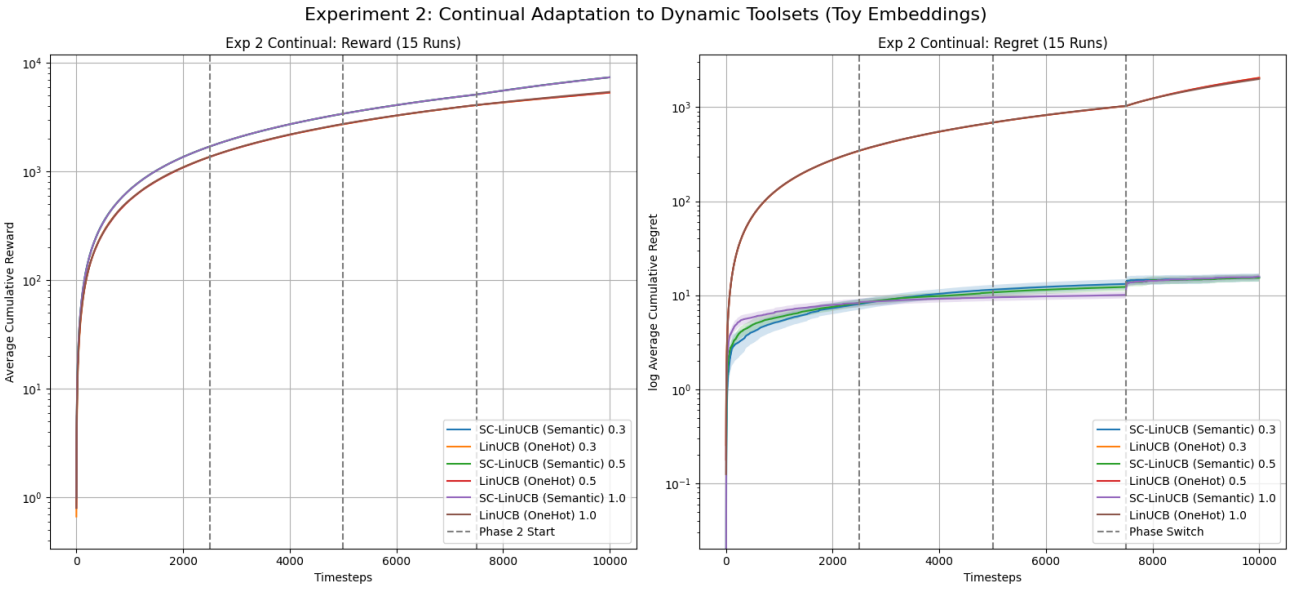

Figure 6. Experiment 2 (Continual Adaptation): Performance of SC-LinUCB (Semantic) and LinUCB-OneHot (Non-Semantic) with varying exploration parameters $\alpha \in \{0.3, 0.5, 1.0\}$. Results over 4×2500 timesteps, averaged over 15 runs. Vertical dashed lines indicate phase shifts. Left: Average Cumulative Reward (log scale). Right: Average Cumulative Regret (log scale).

minor, temporary increases in regret, from which they recover quickly. SC-LinUCB with $\alpha = 0.3$ and $\alpha = 0.5$ show particularly stable and low regret. The $\alpha = 1.0$ variant, while achieving high rewards, exhibits slightly higher regret and notably wider variance (shaded area), especially around phase shifts, likely due to more extensive exploration when the environment changes. This indicates that while higher exploration can find good policies, it might come at the cost of some initial suboptimality if the semantic signal is already strong.

- **LinUCB-OneHot (Non-Semantic):** All variants incur substantially higher regret, ending up in the 10^2 to 10^3 range. Crucially, at each phase transition where K changes (vertical dashed lines), there is a distinct upward turn or steepening of the regret slope. This clearly visualizes the significant cost of adaptation incurred by LinUCB-OneHot as it re-initializes its model and relearns the utility of tools largely from scratch. Increased exploration (e.g., $\alpha = 1.0$, brown line) helps LinUCB-OneHot achieve lower regret compared to its lower α counterparts, but it remains orders of magnitude worse than any SC-LinUCB variant.

Conclusion from Alpha Sensitivity. SC-LinUCB demonstrates robust superiority over LinUCB-OneHot across the tested range of exploration parameters in this continual learning setting. Its ability to leverage fixed-dimensional semantic features allows for graceful adaptation to dynamic toolsets with minimal regret cost. While LinUCB-OneHot does benefit from increased exploration, its fundamental inability to generalize semantically across tools and its need to restructure its feature space when the number of tools changes impose a significant and persistent learning burden. For the main paper, we typically present results for a representative α (e.g., $\alpha = 0.5$) that showcases good performance for SC-LinUCB, as seen in Figure 2. This detailed ablation confirms the general trends.

C. Appendix ICL Experiments

C.1. Experimental Setup Details

This section provides comprehensive details of the configurations used for all experiments discussed in the main paper, ensuring reproducibility.

C.1.1. LLM Parameters and Prompt Structure

The Large Language Model (LLM) utilized across all four experiments was Gemini 2.0 Flash, accessed via the `models/gemini-2.0-flash` API endpoint. Key generation parameters were consistently set as follows:

- Temperature: 0.5
- Maximum Output Tokens: 500 for Experiments 1 & 2; 1500 for Experiments 3 & 4 (to accommodate potentially longer reasoning with dynamic changes).

No specialized safety settings beyond API defaults were applied.

The fundamental prompt structure provided to the LLM comprised a system message defining the task and action presentation, followed by the interaction history and the current query.

System Prompt Template:

Prompt

```
You are an intelligent assistant playing a multi-armed bandit game.
Your goal is to maximize your total reward over many turns.
The available actions (tools) or types of queries may change over time.
In each turn, you are presented with a user query and a list of currently
available actions. Each action, when chosen for a query it is suited for,
has a specific hidden probability of yielding a reward of 1, and 0 otherwise.
If an action is not suited for the query, or no suitable action is available,
it will likely yield a reward of 0.
You must choose one action if suitable options exist.
If no actions are available or suitable, state that.

Available actions: [
  {Formatted list of actions based on experimental condition}
]
```

The placeholder {`Formatted list of actions...`} was populated according to the active experimental condition (Index Only, Names Only, Description Only or Names + Descriptions) for the currently available tools in that phase/turn.

User Message Template per Turn:

Prompt

```
Interaction History (most recent 20 turns shown for LLM context):
{Interaction history string, e.g.,
Turn 1: Query: "Full Query Text 1", Your Choice: ActionName1, Outcome: Reward 0
...
Turn K: Query: "Full Query Text K", Your Choice: ActionNameK, Outcome: Reward R_K
}

Current User Query (Global Turn {current_global_turn}): "{Current Query Text}"

Think step-by-step about which action is best for the current query.
Consider the query, CURRENTLY available action descriptions, and past experiences.
After your reasoning, state your final choice clearly.
```

```
For example: "Reasoning: [...reasons...]. Chosen Action: ActionName Or Index".
If no action is suitable or available, you can state 'Chosen Action: None'.
Which action do you choose?
```

The interaction history provided in the prompt to the LLM contained the full text of the past 20 queries, chosen actions, and their rewards. The experimental framework maintained the complete history for logging and analysis. Each experiment was run for a set number of independent trials: 5 trials for Experiments 1 and 2 (static), and 7 trials for Experiments 3 and 4 (dynamic).

C.1.2. EXPERIMENT 1 (FQFA) CONFIGURATION DETAILS

- **Description**: Single query repeated for $T = 10$ turns, fixed action space.
- **Query (`q_analyze`)**: "I have a list of sales figures for the last quarter, can you help me understand the growth pattern?" (Optimal Arm: `tool_A`)
- **Arm Configurations**:
  - `tool_A` (Data Analyzer): "Processes numerical data arrays to find trends." ($p^{\text{true}} = 0.9, p^{\text{subopt}} = 0.55$)
  - `tool_B` (Text Formatter): "Cleans and formats long text strings." (Designed with $p^{\text{true}} = 0.9$, used with $p^{\text{subopt}} = 0.5$ when chosen for `q_analyze`)
  - `tool_C` (Image Resizer): "Changes the dimensions of image files." (Designed with $p^{\text{true}} = 0.8$, used with $p^{\text{subopt}} = 0.6$ when chosen for `q_analyze`)

C.1.3. EXPERIMENT 2 (MQFA) CONFIGURATION DETAILS

- **Description**: Queries randomly drawn from a fixed set for $T = 50$ turns, fixed action space.
- **Arm Configurations**:
- `tool_translate` (QuickTranslate): "Translates short text snippets between common languages." ($p^{\text{true}} = 0.85, p^{\text{subopt}} = 0.5$)
- `tool_summarize` (BriefSummary): "Creates a one-sentence summary of a paragraph." ($p^{\text{true}} = 0.75, p^{\text{subopt}} = 0.5$)
- `tool_calendar` (EventScheduler): "Adds events to a user's primary calendar." ($p^{\text{true}} = 0.9, p^{\text{subopt}} = 0.55$)
- `tool_filesearch` (DocFinder): "Searches for local documents by keyword." ($p^{\text{true}} = 0.7, p^{\text{subopt}} = 0.6$)
- **Query Configurations (Randomly Sampled from this set)**:
- `q_trans_hello`: "How do you say 'hello' in Spanish?" (Optimal: `tool_translate`)
- `q_sum_paragraph`: "Give me the gist of this: 'The quick brown fox jumps over the lazy dog every day."' (Optimal: `tool_summarize`)
- `q_sched_meeting`: "Schedule a meeting with Jane for tomorrow at 2 PM." (Optimal: `tool_calendar`)
- `q_find_report`: "Find the Q3 sales report document on my drive." (Optimal: `tool_filesearch`)
- `q_trans_bye`: "What is 'goodbye' in French?" (Optimal: `tool_translate`)
- `q_sum_news`: "Briefly, what's this news about: 'Local team wins championship after a dramatic final.'?" (Optimal: `tool_summarize`)

C.1.4. EXPERIMENT 3 (FQMA) CONFIGURATION DETAILS

- **Description**: Single query repeated for $T = 35$ turns (total across phases), action space changes in phases.
- **Query (`Q_ComplexMath`)**: "Solve the integral of x ^ 2 * sin(x) from 0 to pi, and also find the square root of 1764." (Designated Optimal Arm (when available): `E3_SuperCalc`)
- **Master Arm Configurations**:
- `E3_Calculator` (Basic Calculator): "Performs simple arithmetic (+, -, *, /)." ($p^{\text{true}} = 0.7, p^{\text{subopt}} = 0.1$)
- `E3_SciCalculator` (Scientific Calculator): "Advanced math functions: exponents, logs, trig." ($p^{\text{true}} = 0.9, p^{\text{subopt}} = 0.15$)
- `E3_UnitConverter` (Unit Converter): "Converts units (e.g., kg to lbs, meters to feet)." ($p^{\text{true}} = 0.8, p^{\text{subopt}} = 0.05$)
- `E3_Plotter` (Data Plotter): "Generates simple plots from data." ($p^{\text{true}} = 0.6, p^{\text{subopt}} = 0.1$)
- `E3_SuperCalc` (SuperMath Solver): "Handles complex algebra, calculus, and symbolic math. The ultimate math tool." ($p^{\text{true}} = 0.95, p^{\text{subopt}} = 0.2$)
- **Phase Details (Total 35 Turns)**:
  - Phase 1 (`P1_BasicTools`, 7 Turns): Active Arms: {`E3_Calculator`, `E3_UnitConverter`}.
  - Phase 2 (`P2_SciCalc_Added`, 10 Turns): Active Arms: {`E3_Calculator`, `E3_SciCalculator`, `E3_UnitConverter`}.
  - Phase 3 (`P3_SuperCalc_Arrives`, 10 Turns): Active Arms: {`E3_SciCalculator`, `E3_SuperCalc`}.
  - Phase 4 (`P4_SuperCalc_Only`, 8 Turns): Active Arms: {`E3_SuperCalc`, `E3_Plotter`}.

C.1.5. EXPERIMENT 4 (MQMA) CONFIGURATION DETAILS

- **Description**: Both queries (randomly drawn from phase-specific sets) and actions change over $T = 28$ turns (total across phases).
- **Master Arm Configurations**:
- `E4_Translate_EN_DE` (German Translator): ($p^{\text{true}} = 0.9, p^{\text{subopt}} = 0.1$)
- `E4_Summarize_News` (News Summarizer): ($p^{\text{true}} = 0.85, p^{\text{subopt}} = 0.15$)
- `E4_Weather_API` (City Weather): ($p^{\text{true}} = 0.92, p^{\text{subopt}} = 0.1$)
- `E4_Image_Resize` (Image Resizer): ($p^{\text{true}} = 0.8, p^{\text{subopt}} = 0.05$)
- `E4_Code_Python` (Python Code Assistant): ($p^{\text{true}} = 0.75, p^{\text{subopt}} = 0.2$)
- `E4_General_QA` (Knowledge Bot): ($p^{\text{true}} = 0.7, p^{\text{subopt}} = 0.3$)
- **Master Query Configurations**:
- `Q_Translate_Hello_DE` (Optimal: `E4_Translate_EN_DE`)
- `Q_Summarize_Article` (Optimal: `E4_Summarize_News`)
- `Q_Weather_Berlin` (Optimal: `E4_Weather_API`)
- `Q_Resize_Logo` (Optimal: `E4_Image_Resize`)
- `Q_Python_Loop` (Optimal: `E4_Code_Python`)
- `Q_Capital_France` (Optimal: `E4_General_QA`)
- `Q_Weather_Tokyo` (Optimal: `E4_Weather_API`)

- `Q_Python_Function` (Optimal: `E4_Code_Python`)
- **Phase Details (Total 28 Turns)**:
  - Phase 1 (`P1_Lang_Summary`, 8 Turns): Active Arms: {`E4_Translate_EN_DE`, `E4_Summarize_News`, `E4_General_QA`}. Active Queries: {`Q_Translate_Hello_DE`, `Q_Summarize_Article`, `Q_Capital_France`}.
  - Phase 2 (`P2_Weather_Image`, 10 Turns): Active Arms: {`E4_Weather_API`, `E4_Image_Resize`, `E4_General_QA`}. Active Queries: {`Q_Weather_Berlin`, `Q_Resize_Logo`, `Q_Capital_France`, `Q_Weather_Tokyo`}.
  - Phase 3 (`P3_Coding_Focus`, 10 Turns): Active Arms: {`E4_Code_Python`, `E4_General_QA`, `E4_Weather_API`}. Active Queries: {`Q_Python_Loop`, `Q_Capital_France`, `Q_Weather_Tokyo`, `Q_Python_Function`}.

C.2. Additional plots

The following figures illustrate the average cumulative regret accrued by the agent under each condition. These trends generally corroborate the findings from the reward analysis.

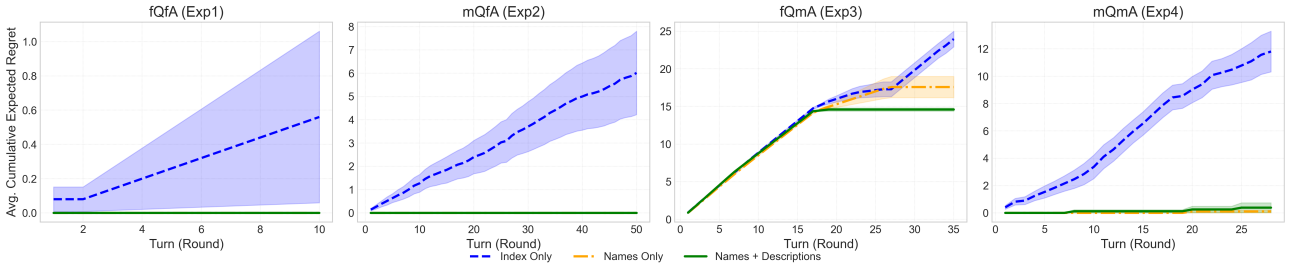

Figure 7. Average Cumulative Expected Regret across Experiments 1-4. Subplot titles use abbreviations: f=Fixed, m=Moving, Q=Queries, A=Actions. Shaded regions represent ± 1 standard error of the mean (SEM) across trials. Note the varying x and y-axis scales across subplots, reflecting different experiment durations and regret magnitudes.

The experimental results, summarized by the average cumulative expected regret curves in figure 7, consistently demonstrate the profound impact of semantic context on the LLM's in-context learning and adaptation for tool selection.

Static Environments (Exp1: fQfA; Exp2: mQfA): In environments with fixed action spaces and query distributions, the provision of rich semantic information via **Names + Descriptions (ND)** yields unequivocally superior performance. As illustrated in 7 (Exp1 and Exp2 panels), the ND condition (green solid line) maintains a cumulative expected regret near zero throughout. This indicates that detailed tool descriptions enable the LLM to rapidly and accurately identify the optimal tool for a given query from the initial turn, effectively bypassing the need for substantial exploration. The LLM, in this condition, behaves as if endowed with strong priors that align well with the task structure.

In stark contrast, the **Index Only (IO)** condition (blue dashed line) results in the highest cumulative regret, which increases approximately linearly. This suggests that in the absence of semantic anchors, the LLM struggles to discern effective query-action mappings, leading to inefficient, near-random exploration or persistent suboptimal choices. The **Names Only (NO)** condition (orange dash-dot line) performs comparably poorly to IO in these static settings, indicating that simple tool names alone provide insufficient semantic grounding for the LLM to reliably infer optimal behavior or differentiate tool efficacies.

Dynamic Environments (Exp3: fQmA; Exp4: mQmA): Non-stationary environments, characterized by changes in the available toolset and/or query distribution, reveal more nuanced interactions between semantic context and adaptability.

In Experiment 3 (fQmA: fixed query, moving actions), the ND condition again demonstrates robust adaptation (7, Exp3 panel). While regret initially accumulates for all conditions due to the unavailability of the globally optimal tool ("E3_SuperCalc"), the ND agent's regret plateaus sharply around turn 17. This event corresponds to a phase change introducing "E3_SuperCalc"

(details in C.1.4), which the ND agent immediately identifies and exploits due to its descriptive alignment with the complex, fixed query. The NO agent also adapts, but with a discernible delay and to a higher regret plateau, suggesting that names offer some utility for identifying new tools but are less efficient than full descriptions. The IO agent fails to adapt effectively, its regret continuing to climb.

Experiment 4 (mQmA: moving queries and actions) presents the most complex scenario and yields a particularly insightful result (7, Exp4 panel). Here, while the ND condition maintains very low regret overall, demonstrating strong adaptability, the **Names Only (NO) condition performs remarkably well, achieving comparable, and at times even slightly lower, cumulative regret than ND**, particularly in later phases. This unexpected outcome suggests that in highly dynamic and complex environments, excessively detailed or numerous descriptions might introduce a cognitive load or risk of misinterpretation that slightly hampers agility. Concise, sufficiently indicative names, under such rapid flux, might allow the LLM to maintain a more flexible mapping, leading to surprisingly robust performance. Nevertheless, both NO and ND vastly outperform the IO condition, which struggles profoundly with the compound dynamism.

C.3. Example Reasoning Traces

Consider, for example, responses to the query:

"query_text": "I have a list of sales figures for the last quarter, can you help me understand the growth pattern?",

- Index only:
  - "Reasoning: The user is asking for help understanding sales figures and growth patterns. This suggests a need for data analysis and visualization. Without knowing the specific functionality of each action, I will assume Action 1 is a general-purpose tool that can handle data analysis tasks. Actions 2 and 3 might be specialized tools that are less likely to be relevant to this general request.\n\n Chosen Action: Action 1\n",
  - "Reasoning: The query is about sales figures and understanding growth patterns. Action 1 has consistently provided rewards for similar queries in the past. Therefore, it is likely the best choice for this query as well.\n Chosen Action: Action 1",
- Names:
  - "Reasoning: The query asks for help understanding a growth pattern in sales figures. This clearly points to data analysis as the appropriate action. The other actions, Text Formatter and Image Resizer, are not relevant to numerical data analysis.\n\n Chosen Action: Data Analyzer",
  - "Reasoning: The query is about understanding the growth pattern of sales figures. The "Data Analyzer" tool has consistently provided a reward of 1 for similar queries in the past. The other tools, "Text Formatter" and Ïmage Resizer,, are not relevant to analyzing numerical data. Therefore, the "Data Analyzer" is the most likely action to yield a reward.\n\n Chosen Action: Data Analyzer",
- Names + Descriptions:
  - "Reasoning: The query mentions s̈ales figuresänd g̈rowth pattern,, which indicates numerical data and the need to find trends. The Data Analyzer tool is specifically designed for processing numerical data arrays to find trends. The other tools, Text Formatter and Image Resizer, are not relevant to this query. Therefore, the Data Analyzer is the most likely to yield a reward.\n\n Chosen Action: Data Analyzer",
  - "Reasoning: The query explicitly mentions s̈ales figuresänd g̈rowth pattern,ẅhich are numerical data-related tasks. The Data Analyzer is designed to process numerical data arrays to find trends. In the previous turn with a similar query, the Data Analyzer yielded a reward of 0. However, this could be due to chance, and the Data Analyzer is still the most appropriate tool for this type of query. The other tools are clearly not relevant.\n\n Chosen Action: Data Analyzer",

C.4. Filter Reason Act Pipeline

We use the following prompt for 0-shot experiment:

Prompt

```
   f"""[BEGIN OF TASK INSTRUCTION]
You are an expert in composing functions. You are given a question and a set of
    possible functions.
Based on the question, you will need to make one or more function/tool calls to
    achieve the purpose.
If none of the function can be used, point it out and refuse to answer.
If the given question lacks the parameters required by the function, also point it
    out.
[END OF TASK INSTRUCTION]
[BEGIN OF AVAILABLE TOOLS]
{actions_prompt_part}
[END OF AVAILABLE TOOLS]
[BEGIN OF FORMAT INSTRUCTION]
The output MUST strictly adhere to the following JSON format,
and NO other text MUST be included.
The example format is as follows. Please make sure the
parameter type is correct. If no function call is needed,
please make tool_calls an empty list []

{{
"tool_calls": [
{{"name": "func_name1", "arguments": {{"argument1": "value1", "argument2": "value2
    "}}}},
... (more tool calls as required)
]
}}

[END OF FORMAT INSTRUCTION]
[BEGIN OF QUERY]
User Query: {query}
[END OF QUERY]
"""
```

where `actions_prompt_part` are the available actions with descriptions in the respective IO, NO, DO or DN format and `query` is the respective task.

The LLM as judge model used was `gemini-2.5-flash-light`.