

# EFFICIENT ON-DEVICE AGENTS VIA ADAPTIVE CONTEXT MANAGEMENT

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

On-device AI agents offer the potential for personalized, low-latency assistance, but their deployment is fundamentally constrained by limited memory capacity, which restricts usable context. This reduced practical context window creates a trade-off between supporting rich, stateful interactions with complex tool capabilities and maintaining on-device feasibility. We break this trade-off with a framework for context-efficient on-device agents, driven by three synergistic optimizations (1) a dynamic memory system using specialized LoRA adapters to distill conversational history into a compressed, and structured *Context State Object*; (2) a minimalist serialization format for tool schemas to minimize token overhead per tool; and (3) a just-in-time schema-passing mechanism that loads full tool definitions only upon tool selection. We instantiate this framework by adapting a 3B parameter SLM to context-efficient trajectories and rigorously evaluate it against a conventional baseline on complex user tasks. Our agent matches, or exceeds, the performance of a conventional baseline while dramatically compressing context, achieving more than a 6-fold reduction in initial system prompt context and a 10- to 25-fold reduction in context growth rate based on the interaction verbosity, demonstrating that strategic context management is key to unlocking capable and persistent on-device AI.

## 1 INTRODUCTION

AI agents, powered by Large Language Models (LLMs), have catalyzed productivity gains across numerous domains (Wang et al., 2025b; Ju & Aral, 2025). Their true potential, however, lies in migrating from the cloud to personal devices, where they can leverage private data and control local applications to unlock a new frontier of personalized productivity (Ferev et al., 2024; Chen et al., 2024). This transformative vision, however, is fundamentally limited by the on-device hardware constraints, creating a significant bottleneck for progress (Xu et al., 2024). Central to this bottleneck is the agent’s context window, which serves as its working memory and is essential for sophisticated capabilities like maintaining conversation state and utilizing complex tools. Maintaining a long context on-device creates two key challenges: **(1)** unsustainable memory consumption from the linearly growing ( $O(n)$ ) KV cache (Xiao et al., 2024), and **(2)** degraded reliability with longer inputs (Liu et al., 2023). This context bottleneck is amplified by three core aspects of agentic behavior. First, **conversational history** in the user–agent–environment loop accumulates rapidly through token-heavy tool calls and environment responses, obscuring essential information from earlier turns (Zhou et al., 2025b). Second, the **verbosity of tool schemas** (e.g., OpenAPI (OpenAPI Initiative, 2024) format) floods the context with thousands of tokens. Third, this creates a **tool scalability** problem, limiting the agent’s versatility (Erdogan et al., 2024). The result is a crippled agent which is either forgetful and unable to maintain a coherent dialogue due to frequent context dumping, or simplistic and incapable of performing complex tasks. Existing solutions are ill-suited for this domain as they are designed for inter-session memory management (Packer et al., 2024), incompatible with optimizations like KV-caching (Zhou et al., 2025b), use heuristics for context management (Kim et al., 2025), or are designed for very long contexts of 100K+ tokens Lin et al. (2025). In contrast, on-device agents face tight memory budgets, where even 5–10K tokens can be prohibitive, yet preserving conversational detail and flow is essential.

In this work, we present a framework for building context-efficient on-device agents that targets each source of context bloat. We introduce a dual-adapter memory system that uses a specialized LoRA

054 adapter (Hu et al., 2021) to distill conversational history into a compressed, structured *Context State*  
055 *Object (CSO)*. To tackle tool-related overhead, we propose a two-pronged tool management system:  
056 a minimalist serialization format for schemas and a just-in-time mechanism that loads definitions  
057 only upon selection. The developed models achieve strong task performance at a fraction of the to-  
058 ken cost. We instantiate our framework by fine-tuning a 3B parameter SLM and rigorously evaluate  
059 it against a conventional baseline on complex user tasks. **Our primary contributions are:**

- 060 • A novel on-device agent architecture for local tool execution or routing to cloud LLM.
- 061 • A dual-adapter memory system that maintains a compressed, but interpretable, conversational  
062 state which matches the performance of strong baselines on complex, multi-turn tasks while re-  
063 ducing the context growth rate by a factor of 10 to 25, with larger gains for verbose conversations.
- 064 • An integrated tool management system that combines token-efficient schemas with selective tool-  
065 passing resulting in more than 6x lower initial tool overhead for more capable agents.

## 067 2 RELATED WORKS

### 069 2.1 ON-DEVICE AGENTS AND FUNCTION CALLING

070 The drive for privacy, low latency, and personalization has spurred a shift towards on-device intel-  
071 ligence, built upon Small Language Models (SLMs) like Gemma-3 and Phi-3-mini that can operate  
072 within device constraints (Xu et al., 2024; Team et al., 2025; Abdin et al., 2024). The frontier for  
073 these SLMs lies in specialized agentic applications, where they act as low-cost orchestrators or di-  
074 rectly interact with the environment via function calling (Chen et al., 2024; Lee et al., 2025). A  
075 common pattern is using an SLM as a router to dispatch tasks to other components (Yue et al., 2025;  
076 Tran et al., 2025). Our work adopts this hybrid on-device/cloud routing paradigm, but focuses on  
077 the context efficiency necessary for practical, long-running interactions.

078 Modern function calling agents, built on foundational frameworks like ReAct and Toolformer (Yao  
079 et al., 2023; Schick et al., 2023), are typically trained with Supervised Fine-Tuning (SFT) on large-  
080 scale synthetic datasets (Prabhakar et al., 2025; Qin et al., 2023). While large models achieve high  
081 accuracy, performance degrades for the SLMs used on-device (Zhang et al., 2025; Abdelaziz et al.,  
082 2024; Lin et al., 2024). A primary driver of this is the token overhead from tool schemas. Recent  
083 work confirms that reducing irrelevant tools improves performance (Paramanayakam et al., 2024),  
084 and systems like TinyAgent use retrieval to select a tool subset before prompting (Erdogan et al.,  
085 2024). Our work builds on this by introducing a two-stage process where the agent first selects a tool  
086 from a lightweight list, allowing it to be fully aware of its capabilities at minimal token overhead,  
087 and only then loads its full, token-optimized schema.

### 089 2.2 CONTEXT AND MEMORY MANAGEMENT

090 Context management has been studied through multiple lenses. Inter-session approaches such as  
091 recursive summarization (Wang et al., 2025a) and MemGPT (Packer et al., 2024) emphasize per-  
092 sistence but often lose fine-grained detail. Intra-session methods like MEM1 (Zhou et al., 2025b)  
093 improve fidelity but remain computationally intensive and incompatible with KV caching (Pope  
094 et al., 2022). Long-context modeling efforts (e.g., retrieval-augmented generation (Lin et al., 2025),  
095 sparse attention (Xu et al., 2025)) extend sequence length to 100K+ tokens, but target different  
096 scales than on-device budgets. Another direction compresses or removes KV cache entries (Kim  
097 et al., 2025; Park et al., 2025), but relies on low-level heuristics, such as attention magnitude and  
098 reconstruction error, which risk discarding key details that may only be needed much later in the  
099 trajectory (e.g., an address or ID). These compressed representations are opaque and hard to debug.  
100 In contrast, we condense history at the semantic level, where a trained model maintains a compact  
101 and rich log, ensuring interpretable, task-relevant memory that remains reliably accessible for tool  
102 use while achieving stronger context reductions.

## 104 3 FRAMEWORK FOR CONTEXT-EFFICIENT ON-DEVICE AGENTS

106 In this section, we detail the architecture and mechanisms of our proposed framework for building  
107 context-efficient on-device agents, describing the overall system architecture, then elaborate on our  
two core context-reduction contributions, and finally describe our data generation pipeline.

### 3.1 SYSTEM ARCHITECTURE

The foundation of our framework is an on-device agent powered by a 3B-parameter SLM. This agent acts as an orchestrator, as illustrated in Figure 1, and performs one of three actions based on user input: (1) **Direct Response** for general conversation; (2) **Local Execution** by calling on-device tools for tasks like checking a calendar; or (3) **Cloud Delegation** for complex queries.

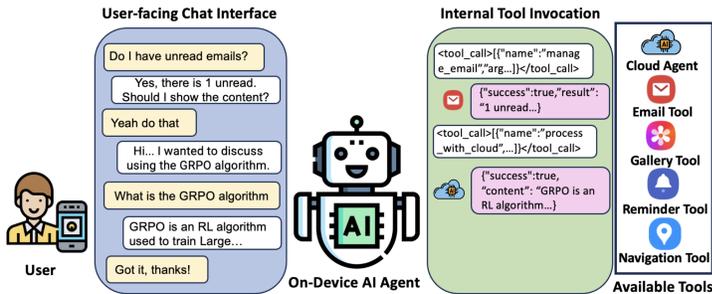


Figure 1: System architecture of the on-device AI agent. The user interacts with the agent through a chat interface. The on-device agent processes the user’s requests and performs internal tool invocations, utilizing a suite of local, on-device tools (e.g., Email, Gallery, Reminders) as well as a more powerful Cloud Agent for complex queries.

This model leverages local speed and privacy while ensuring complex problems can be solved effectively with the Cloud Agent. The Cloud Agent, which is a much larger and capable LLM, also prevents common user frustrations, such as incorrectly calling irrelevant tools, hallucinations or denying requests due to lack of capability. However, the verbose cloud responses intensify the need for extreme context efficiency to maintain high performance and low memory usage on-device.

### 3.2 DYNAMIC MEMORY VIA CONTEXT STATE OBJECT

To manage conversational history without high token costs, we introduce a dynamic memory system built on a dual-adapter architecture. This system replaces the raw conversation history with a compressed, structured text log we term the *Context State Object (CSO)*. The CSO serves as an evolving, append-only log that captures the essential state of the interaction in a token-efficient format. This mechanism is powered by two complementary, parameter-efficient LoRA (Hu et al., 2021) modules:

- **The Executor (LoRA<sub>Exec</sub>):** The primary agentic adapter, fine-tuned to perform the core tasks in Section 3.1 by leveraging the CSO for conversational context.
- **The State-Tracker (LoRA<sub>Mem</sub>):** A specialized, lightweight (119M parameters) adapter which generates a concise, append-only CSO update based on the latest turn.

The memory update cycle occurs after each assistant turn, designed for maximum efficiency:

1. **Task Execution:** The `Executor` adapter is active and processes the user query and the state  $CSO_{t-1}$ , to generate a response or tool call.
2. **State Update Generation:** The active adapter is swapped to the `State-Tracker`. It is prompted with  $CSO_{t-1}$  and the latest user-assistant exchange to generate only the new lines to be appended (the delta,  $\Delta_t$ ).
3. **State Concatenation:** The new state object,  $CSO_t$ , is formed by appending the delta to the previous state:  $CSO_t = CSO_{t-1} \oplus \Delta_t$ .

This architecture, as demonstrated in Figure 2a provides several advantages for stateful, on-device agents. First, by generating a small, append-only update once per turn, it maintains a linear growth of context but with a significantly reduced rate. Our experiments show this leads to a 10-fold reduction in the rate of context increase for typical, non-cloud conversations, and a 25-fold reduction with verbose outputs from the cloud-based agent. Over conversations involving 20+ assistant turns, this translates into savings of thousands of tokens, enabling more capable and persistent interactions.

Second, the design is optimized for inference. The lightweight `Executor` and `State-Tracker` adapters can be simultaneously loaded into memory with minimal overhead. Crucially, our custom inference implementation tracks the Key-Value (KV) caches for both adapters separately. Because the CSO is only appended to, the KV cache for the large, static portion of the prompt (system

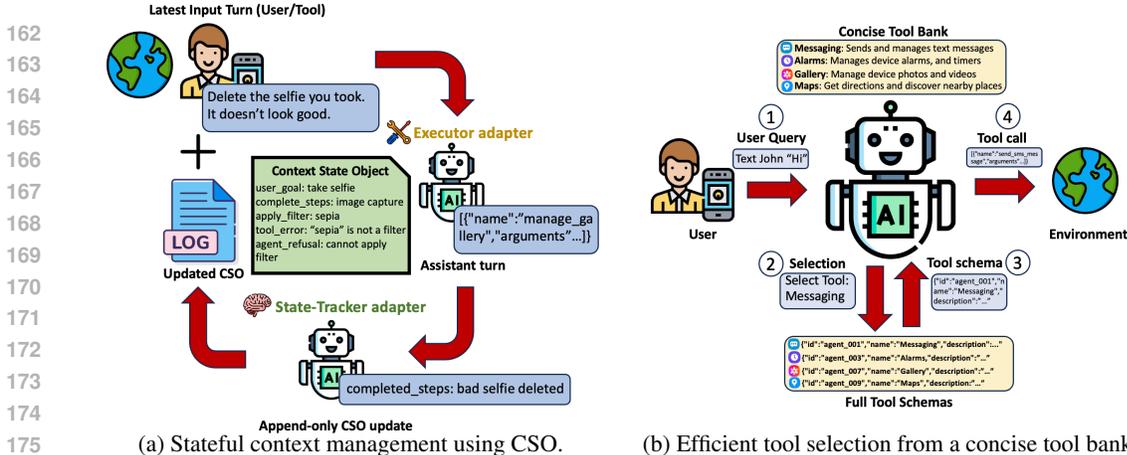


Figure 2: Our two context optimization techniques for on-device agents. (a) A state-tracking system maintains a compact history. (b) A two-step tool-call process reduces the overhead of tool schemas.

instructions, tool schemas (for the `Executor`), and  $CSO_{t-1}$  is preserved for both adapters. Only the small update and new query require processing, dramatically accelerating inference speed.

Finally, the `State-Tracker` is explicitly trained to produce a rich, semi-structured log that selectively preserves key task-relevant information, including detail extracted from otherwise verbose cloud responses, thereby maintaining fidelity at substantially lower token cost. We use a key-value format as we hypothesize it will leverage the base model’s strong aptitude for structured data, as developed through function-calling training. This format avoids the need for complete sentences and through explicit training allows the `State-Tracker` to log critical details often lost in conventional summarization, such as tool errors, agent refusals, and other environmental feedback. This continuous, detailed log allows the `Executor`, which is co-adapted to this format, to better track implied user intent, key requirements and recover from errors, leading to a more robust and natural conversational flow. A walkthrough is provided in Section A.8. This system incurs a modest overhead of an approximate 80MB memory increase and a 500ms latency per turn for the CSO update cycle (Galaxy S25 CPU). We believe optimized deployments can further decrease these costs.

### 3.3 TOOL SCHEMA MANAGEMENT

To address the context bloat due to tool schemas, we utilize a two-pronged strategy that tackles both the verbosity of individual tool representations and the scaling of tools.

**Token-Efficient Schema Representation** Standard API formats such as the OpenAPI Specification (OpenAPI Initiative, 2024) are notoriously verbose for LLM prompts, in part due to human-oriented formatting and whitespace. Prior work and tooling often employ schema *minification* to reduce token cost, typically by stripping indentation or unused fields. Building on this, we train our models directly with a compact format that retains only the essential fields for function invocation (name, description, parameters) while removing non-essential whitespace. As shown in Table 4 (Appendix), this yields an average **40%** reduction in tokens per tool. Crucially, this optimization is integrated into our broader framework for context and state management, allowing significantly more tools to fit within a given context window and thereby enhancing the agent’s usable capabilities.

**Just-in-Time Schema Passing** While our efficient format reduces the cost of each tool, including all schemas in the prompt remains unscalable and introduces significant prompt bloat with irrelevant context (Paramanayakam et al., 2024). Conversely, a pure retrieval-based approach that injects multiple relevant tool schemas at each turn (Erdogan et al., 2024) presents its own trade-offs wherein it can paradoxically consume context faster than standard methods, and can cause the agent to fail when the retriever returns no tools or irrelevant ones. To address these trade-offs, we implement a hybrid *just-in-time (JIT)* schema passing mechanism (Figure 2b). Our two-step approach ensures the agent is always aware of its full capabilities at a minimal token cost, while only loading relevant full schemas:

1. **Selection Stage:** The system prompt provides the agent with a lightweight list of all available tool names and their concise descriptions. This makes the agent fully aware of its potential actions. Based on the user’s query, its task is to select the most appropriate tool by outputting its name (e.g., Timer tool) along with the reasoning for selection.
2. **Execution Stage:** Upon selection, the full, token-optimized tool schema is provided to the agent in the subsequent observation. Now, the agent’s task is to generate the correct arguments in the tool call based on the user’s query and the provided schema.

This JIT approach provides a highly scalable solution to tool management. The initial context remains small and manageable, approximately **25%** of the baseline size, allowing the agent to be equipped with a diverse set of capabilities without overwhelming the on-device SLM. This drastic reduction in initial prompt size also leads to approximately **5x** reductions in Time-To-First-Token (TTFT) for on-device deployment<sup>1</sup>. Together, our efficient schema format and JIT mechanism form a complete system that dramatically reduces tool-related context overhead.

### 3.4 DATA GENERATION AND TRAINING PROTOCOL

To train our agent to operate effectively within our proposed framework, we develop a comprehensive pipeline to produce a high-quality dataset and fine-tune the agent’s dual-adapter memory system and two-stage tool use, along with the baseline which follows standard fine-tuning practices.

**On-Device Use Case Generation** Our process adapts the APIGen-MT pipeline (Prabhakar et al., 2025) for the on-device domain. To promote generalizability and prevent overfitting, we first create a diverse pool of 100 base on-device tools. For each base tool, we then use an LLM to generate 50 variants with modifications to parameters and descriptions, resulting in 5,000 unique tools. We defined four task categories for generation: multi-tool, cloud delegation, mixed on-device/cloud, and conversational queries. We use Gemini 2.0 Flash (Google Cloud, 2024) for multi-turn trajectory generation. For each scenario, a detailed rubric provides the model with a sampled toolset, an initial user utterance, a persona, and a ground-truth tool sequence (more details in Appendix A.7). Separate instances of Gemini then simulate the entire interaction, playing the roles of a non-deterministic user, the assistant, and a tool execution environment that returns realistic success or failure messages. The generated trajectories undergo a rigorous two-stage curation process. They are first validated against the ground-truth tool sequence for task completion. Successful trajectories are then evaluated by an LLM-as-a-judge (also Gemini 2.0 Flash) to filter out low-quality or unnatural interactions.

**Fine-tuning and Specialized Adapter Training** We use LoRA (Hu et al., 2021) to efficiently fine-tune the 3B xLAM 2 base model (Zhang et al., 2025), adapting it to our on-device framework while preserving its strong foundational capabilities. Additionally, the LoRA approach is also modular, allowing our specialized adapters to be composed with other potential modules, such as safety guardrails or further personalization layers. For training, each example provides a context of 10-14 tools (the correct tool, Cloud tool, and distractor tools) to teach robust selection and delegation. All training is conducted using our token-efficient tool schema format. We also qualitatively test the performance on-device with 4-bit quantized models. For training the State-Tracker adapter, the teacher model is used to generate CSO updates for every assistant turn.

## 4 EXPERIMENTAL SETUP

To rigorously evaluate our context-efficient framework, we design a comprehensive evaluation to assess both generalization and performance across a diverse range of on-device scenarios.

### 4.1 EVALUATION DATASET AND BASELINES

We design a challenging evaluation set of 406 on-device tasks using the generation pipeline from Section 3.4, but with a crucial modification where the test set is built using 19 new, unseen tools for

<sup>1</sup>Evaluated on CPU-based Samsung Galaxy S25 devices and averaged across 3 evaluations. The reduction in TTFT is an expected consequence of the drastically smaller initial prompt size, as prefill processing time is a primary driver of initial latency.

testing generalizability (examples provided in Appendix A.4). This ensures we measure the agent’s ability to apply learned skills to novel APIs, not just memorize tool signatures. We use a simulated user to interact with the trained models over long trajectories, introducing ambiguity, multi-step intents, and naturalistic expression to make the evaluation realistic (Vijayvargiya et al., 2025; Zhou et al., 2024; 2025a). We evaluate five model variants to isolate the impact of our contributions:

- **xLAM 2 3B (Reference):** The model from previous work (Zhang et al., 2025) prompted with full conversational history and OpenAPI tool schemas.
- **Baseline (FT):** The baseline model fine-tuned on our use case, but only with token-efficient schemas and full conversational history.
- **Tool-Efficient:** Fine-tuned with our complete tool schema management system.
- **Memory-Efficient:** Fine-tuned with our dual-adaptor memory system for history management.
- **Combined:** Our full model, integrating both the tool and memory efficiency mechanisms.

## 4.2 EVALUATION METRICS

We evaluate model performance using a suite of automated and qualitative metrics.

- **Rule-Based Tool Call Metrics:** To provide a more nuanced view of the agent’s tool-use capabilities, we measure the quality of its tool calls.
  - *Precision* measures the fraction of tool calls made by the agent that were correct (i.e., part of the ground-truth sequence). It penalizes non-existent, syntactically incorrect, or irrelevant calls.
  - *Recall* measures the fraction of required ground-truth tool calls that the agent successfully made.
  - *F1 Score* is the harmonic mean of Precision and Recall, providing a single, balanced measure of tool-calling accuracy.
- **LLM-as-Judge:** We evaluate the qualitative nature of the trajectories to identify if the user’s goal was easily satisfied with minimal frustration on a scale of 1-5 with 5 being an ideal conversation.

We also track the input context length at each assistant turn across the evaluated models. To mitigate variability from the non-deterministic user simulation, performance on all metrics is averaged across three runs. Results are broken down by task category for a holistic analysis.

## 5 RESULTS

In this section, we present the empirical evaluation of our framework, first analyzing task performance to demonstrate that our efficiency mechanisms preserve or enhance capability, and then quantifying the dramatic reductions in token overhead.

### 5.1 TASK PERFORMANCE

We evaluate our fine-tuned models against the xLAM-2 3B model to contextualize our results. Table 1 presents a detailed performance breakdown across our complex task categories. While the xLAM-2 model excels at generic tool calling, our fine-tuning helps adapt to the nuanced task of cloud orchestration, as shown by the Baseline’s superior performance on cloud-related tasks. Among our proposed methods, the Tool-Efficient model demonstrates the highest precision, a direct result of its deliberate two-stage process, though this rigidity impacts its recall. The Memory-Efficient model, however, emerges as a robust performer. It achieves a perfect F1 score on Cloud Delegation and the highest F1 on the challenging On-Device + Cloud tasks, while also maintaining strong multi-tool performance. Its high qualitative (Q) scores in these categories further suggest a more coherent and effective user interaction. Finally, our Combined model achieves the highest overall tool-calling F1 scores, showing strong decision making and intelligence. Its qualitative performance in tool-calling trajectories lags behind the strongest variants because the multi-step selection and invocation turns lead to a higher ratio of structured tool call turns in the training data relative to user-facing information presentation turns. This imbalance causes the model to prioritize structured tool calls over natural responses, impacting interaction quality. We keep the data sampling method fixed across models to ensure a fair comparison without introducing confounding factors.

Table 1: Performance breakdown on complex, multi-turn tasks. We compare our fine-tuned models and also present the xLAM-2 3B model as a reference. We report F1-Score (F1), Precision (P), Recall (R), and a qualitative score (Q, 1-5 scale). Higher scores are better across all metrics.

Model Variant	Cloud-Tool Only				Multi-Tool				On-Device + Cloud				Convo.
	F1	P	R	Q	F1	P	R	Q	F1	P	R	Q	Q
xLAM-2 3B	0.41	0.53	0.63	2.07	0.87	0.86	0.91	3.59	0.57	0.65	0.63	2.27	2.18
Baseline (FT)	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	3.79	0.83	0.79	0.92	3.17	0.88	0.94	0.87	3.45	2.87
Tool-Efficient	0.93	0.97	0.95	3.55	0.86	<b>0.92</b>	0.88	<b>3.61</b>	0.73	0.85	0.74	3.32	3.21
Mem-Efficient	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>4.11</b>	0.87	0.87	0.90	3.37	0.89	<b>0.95</b>	0.88	<b>3.64</b>	3.32
<b>Combined</b>	0.99	0.98	<b>1.00</b>	2.68	<b>0.93</b>	<b>0.92</b>	<b>0.95</b>	3.32	<b>0.94</b>	0.92	<b>0.92</b>	3.00	<b>3.80</b>

## 5.2 ANALYSIS OF CONTEXT EFFICIENCY

Our framework’s primary motivation is to drastically reduce context overhead. While we have shown our methods perform similarly, or better than standard finetuning, here we examine the initial prompt size and the subsequent rate of context growth to highlight the token cost savings.

### 5.2.1 INITIAL CONTEXT SIZE AND TOOL SCALABILITY

The initial prompt size is critical for on-device performance, directly impacting Time-To-First-Token (TTFT) and determining the maximum number of tools an agent can be aware of. As shown in the starting point (Assistant Turn 0) of Figure 3, the models have vastly different initial footprints.

- The **xLAM-2** model, using standard schemas, starts with an initial context of 3,200 tokens.
- Our **Baseline** and **Memory-Efficient** models, which use our token-efficient schema format, reduce this initial cost to 2,100 tokens.
- Our **Tool-Efficient** model, which includes only tool names and descriptions in the initial prompt and instead relies on JIT loading of tool schemas, requires a mere 400 tokens.
- The **Combined** model leverages the strengths of the Tool-Efficient model in initial context.

This demonstrates that our JIT mechanism and schema format, allow an agent to be aware of an approximately 8x larger toolset for the same token budget and significantly improves TTFT.

### 5.2.2 CONTEXT GROWTH RATE OVER CONVERSATION

We now analyze context accumulation over a conversation. Figure 3 shows that for general Multi-Tool use, the xLAM-2 and Baseline models’ contexts grow linearly to 5,000 tokens. The Tool-Efficient model, while starting lowest, also sees its context expand to comparable sizes on long trajectories. In stark contrast, the Memory-Efficient model’s context remains remarkably flat, increasing by only 100-200 tokens after 25 turns, leading to a 10-fold reduction in context accumulation compared to the Baseline. The Combined model sees slightly higher growth of around 500-700 tokens due to full schemas being injected into its inputs unlike the Memory-Efficient model.

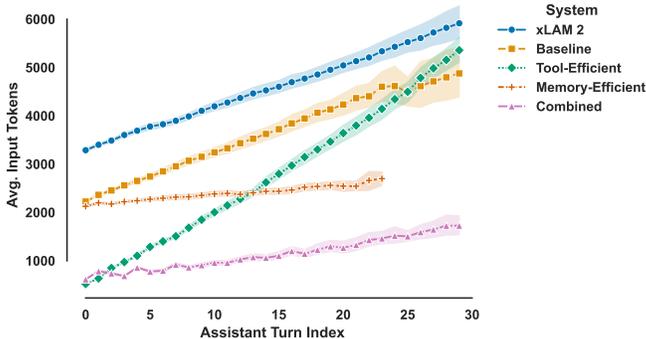


Figure 3: Averaged context input length over assistant turns for the **Multi-Tool** category in the evaluation runs. Shaded regions represent 95% Confidence Intervals.

This advantage becomes even more extreme in Cloud Delegation scenarios (Figure 4). The Baseline’s context explodes to 10,000-12,000 tokens due to verbose cloud responses whereas

378 the Memory-Efficient model distills these responses into concise CSO updates, limiting its con-  
 379 text growth to 500 tokens. We achieve a context growth that is more than 20-fold smaller.  
 380 The Combined model leverages  
 381 the strengths of the Memory-  
 382 Efficient model to maintain low  
 383 context growth rates but at much  
 384 lower absolute token counts.  
 385 The slower context growth of  
 386 the xLAM-2 and Tool-Efficient  
 387 models in the longest cloud tra-  
 388 jectories is a byproduct of their  
 389 failure modes. The former of-  
 390 ten fails to call the cloud tool  
 391 correctly, while the latter oc-  
 392 casionally gets stuck in select-  
 393 loops (unlike the Baseline which  
 394 gets stuck in cloud tool call  
 395 loops), thus avoiding its verbose  
 396 response from cloud. Therefore,  
 397 our Memory-Efficient model is  
 398 unique in that it achieves its extreme context efficiency while achieving the highest success rate.

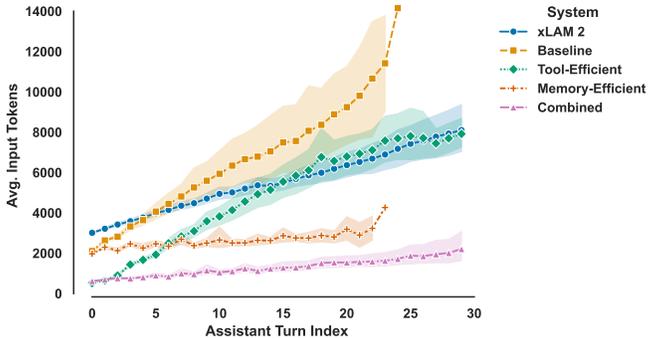


Figure 4: Averaged context input length over assistant turns for the **Cloud Delegation** category in the evaluation runs. Shaded regions represent 95% Confidence Intervals.

## 6 DISCUSSION

401 Our empirical results demonstrate that our context-efficient framework dramatically reduces token  
 402 overhead while maintaining or enhancing agent performance on complex tasks. The strong results of  
 403 our Combined model, in particular, highlight our central hypothesis: for on-device agents, strategic  
 404 context management is a critical enabler of stateful behavior. In this section, we interpret these  
 405 findings and analyze the architectural trade-offs revealed by our experiments, focusing on the Tool-  
 406 Efficient and Memory-Efficient models as the Combined model directly borrows traits from both.

### 6.1 SHORTER CONTEXTS BOOST DECISION-MAKING AND COHERENCE

407 While our quantitative results show a clear performance gap on complex tasks, a qualitative analy-  
 408 sis reveals the underlying reasons. The Context State Object (CSO) enables our Memory-Efficient  
 409 agent to exhibit more robust long-term decision-making. Unlike a simple summary, the CSO func-  
 410 tions as an explicit, structured memory, allowing the agent to effectively track previous actions and  
 411 observations, and recover from errors due to less distractions for attention in the input.

412 A common failure mode for other model variants is repetitive error looping. When a tool call fails,  
 413 the agent often ignores the error from the tool and attempts the same failing action on its next turn,  
 414 especially when the preceding conversational history is long. We hypothesize this is a symptom of  
 415 attention dilution (Liu et al., 2023), where in a long, unstructured context, the critical error feedback  
 416 from the previous turn can be drowned out by less relevant historical tokens. In contrast, the State-  
 417 Tracker adapter is trained to distill critical events, such as tool\_error, into a concise, structured entry  
 418 in the CSO. When prompted with this updated state, the Executor can reliably attend to the logged  
 419 error, and pivot to a new strategy, such as retrying, clarifying with the user or delegating to the cloud.  
 420 This is reflected in the performance metrics, where the recall of Tool-Efficient model is boosted in  
 421 the Combined model due to better error recovery with the CSO.

### 6.2 EFFICACY OF DISTILLED STATE TRACKING

422 A key finding is the effectiveness of the State-Tracker adapter, trained solely via supervised fine-  
 423 tuning. A qualitative analysis of its generated Context State Objects (CSOs) reveals that it learned  
 424 to perform more than simple summarization. We observe three notable properties:

- 425 1. **Task-Oriented State Decomposition:** The State-Tracker learns to decompose ambiguous user  
 426 utterances into a structured state representation. For instance, for a conditional request like *Run*

432 *a scan if battery more than 50%*, it generates distinct entries for the *user\_goal* and the blocker  
 433 (*need\_info*). Crucially, it also logs the Executor’s limitations, such as adding *agent\_limitation*:  
 434 *cannot check battery level* after a failure. This creates a detailed and concise log of events and  
 435 the keys capture the higher level flow, enabling the agent to avoid repeating erroneous actions.

- 436 2. **Adaptive Logging:** The model learns to vary the length of its CSO updates based on the informa-  
 437 tion density of the preceding turn. It generates concise updates for simple turns but more detailed  
 438 updates after verbose cloud responses, suggesting a capability to gauge information salience.
- 439 3. **Alignment with Base Abilities:** The key-value format of the CSO has proved highly effective,  
 440 likely because it aligns with the base xLAM-2 model’s strong training on structured data. We  
 441 hypothesize the keys (e.g., *user\_goal*) function as strong attentional anchors, making it easier for  
 442 the Executor to understand the general flow of the conversation compared to unstructured prose.  
 443

444 While this process is inherently lossy, due to discarding the conversational filler, our results show it  
 445 preserves the task-critical information and conversational flow for on-device deployment.  
 446

### 447 6.3 PRECISION VS. RECALL TRADE-OFF

449 Our study reveals an interesting architectural trade-off between the Tool-Efficient and Memory-  
 450 Efficient models. The Tool-Efficient model, with its two-stage selection process, consistently achiev-  
 451 ing high precision as compared to its recall. Its rigid structure, which requires explicit reasoning to  
 452 justify tool selection, appears to enforce a more deliberate initial choice but comes at the cost of  
 453 lower recall, as the model struggles to pivot to an alternative tool if its first choice is incorrect.  
 454 Further, we observe that the complexity of the two-step protocol can sometimes lead to degraded  
 455 instruction following, making the model less responsive to subtler user requirements.

456 The Memory-Efficient model, conversely, demonstrates greater flexibility and higher overall recall.  
 457 Its access to a rich conversational history via the CSO allows it to better handle ambiguity and  
 458 attempt different strategies if one approach fails, especially as exact previous turns are not visible  
 459 preventing repetitive looping Xu et al. (2022). A precision-focused JIT approach may be preferable  
 460 in high-stakes scenarios where a single incorrect tool call has persistent negative consequences,  
 461 whereas high recall provides the crucial flexibility needed to ensure a user’s ultimate goal is achieved  
 462 in more complex or exploratory interactions. The Combined model benefits from the interaction of  
 463 the two strategies as it achieves high precision due to the 2-step tool calling, and high recall due to  
 464 the ability to attempt different strategies if initially unsuccessful.

## 465 7 CONCLUSION, LIMITATIONS, AND FUTURE WORK

466 By systematically addressing context bloat from both conversational history and tool schemas, we  
 467 introduce a holistic framework that enables agents to be both more efficient and more capable. Our  
 468 Combined agent, with a dual-adapter memory system and integrated tool management protocol,  
 469 supports complex, stateful interactions at a fraction of the token cost of conventional methods. Ex-  
 470 periments show strong performance on challenging multi-turn tasks while reducing context growth  
 471 by over an order of magnitude, making practical on-device deployment far more feasible.  
 472

473 At the same time, our study has natural boundaries. To clearly isolate the effects of our context mech-  
 474 anisms, training was limited to a single epoch on a fixed data distribution, leaving open how broader  
 475 training regimes or diverse domains might impact results. Our method also builds on supervised  
 476 distillation from a teacher model, which constrains the State-Tracker’s quality to the teacher’s ca-  
 477 pabilities. Finally, because existing benchmarks are not designed for dynamic context mechanisms,  
 478 we present our results on a custom evaluation protocol. While this limits direct comparability, we  
 479 ground our findings by benchmarking against a strong function-calling baseline.  
 480

481 These limitations also suggest promising directions for future work. Reinforcement learning could  
 482 enable agents to make more adaptive decisions, overcoming tool-calling and clarification patterns  
 483 learned during supervised fine-tuning. Additionally, evaluating our context management approaches  
 484 across open-domain tasks would test their generalizability. Overall, our results underscore that  
 485 achieving persistent, capable on-device agents will rely on continued advances in computational  
 efficiency.

## REPRODUCIBILITY STATEMENT

To ensure the reproducibility of the presented results, this paper provides comprehensive details on the methodology, data generation, and experimental setup. All key components of the proposed framework are described with the intention of enabling replication by an independent research group.

- **Framework Architecture:** The designs of the dual-adapter memory system and the just-in-time tool management protocol are detailed in Section 3. A step-by-step walkthrough of the custom Key-Value cache management logic is provided in Appendix A.8.
- **Data Generation and Curation:** The full prompts used to guide the teacher model (Gemini 2.0 Flash) for generating both the conversational scenarios and the Context State Object updates are included in Appendix A.7 and A.3. The two-stage data curation process is described in Section 3.4.
- **Training and Evaluation Protocol:** All hyperparameters for LoRA and model fine-tuning are listed in the tables within Appendix A.2, including details on the base model, learning rates, and training data composition. The evaluation protocol, including the composition of the custom test set with unseen tools and the definition of all metrics, is described in Section 4. To account for variability in the non-deterministic evaluation, all reported performance metrics are the average of three independent runs.

The comprehensive details provided throughout the main paper and appendices are intended to be sufficient for the full re-implementation and validation of our work.

## LLM USAGE

We used a large language model to assist with polishing the writing style, condensing the content, and improving clarity. All research ideas, methods, experiments, and analyses were developed and conducted by the authors. The LLM did not contribute to scientific content.

## REFERENCES

- Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, GP Bhargav, Maxwell Crouse, Chulaka Gunasekara, Shajith Ikbali, Sachin Joshi, Hima Karanam, Vineet Kumar, Asim Munawar, Sumit Neelam, Dinesh Raghu, Udit Sharma, Adriana Meza Soria, Dheeraj Sreedhar, Praveen Venkateswaran, Merve Unuvar, David Cox, Salim Roukos, Luis Lastras, and Pavan Kapanipathi. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks, 2024. URL <https://arxiv.org/abs/2407.00121>.
- Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan,

- 540 Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan  
541 Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your  
542 phone, 2024. URL <https://arxiv.org/abs/2404.14219>.
- 543
- 544 Wei Chen, Zhiyuan Li, and Mingyuan Ma. Octopus: On-device language model for function calling  
545 of software apis, 2024. URL <https://arxiv.org/abs/2404.01549>.
- 546
- 547 Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Cole-  
548 man Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Tinyagent: Function  
549 calling at the edge, 2024. URL <https://arxiv.org/abs/2409.00608>.
- 550 Milen Ferev, Yishuang Pang, and Pulkit Bhuvalka. On-device llm-driven application function call-  
551 ing for natural language interaction. 2024.
- 552
- 553 Georgi Gerganov and The llama.cpp contributors. llama.cpp: Inference of llama model in pure  
554 c/c++. <https://github.com/ggerganov/llama.cpp>, 2023. GitHub repository.
- 555
- 556 Google Cloud. Gemini 2.0 Flash and 2.0 Pro update. [https://cloud.google.com/  
557 vertex-ai/generative-ai/docs/models/gemini/2-0-flash](https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash), 2024. Accessed:  
558 2024-10-27.
- 559
- 559 Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang,  
560 and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL <https://arxiv.org/abs/2106.09685>.
- 561
- 562 Harang Ju and Sinan Aral. Collaborating with ai agents: Field experiments on teamwork, produc-  
563 tivity, and performance, 2025. URL <https://arxiv.org/abs/2503.18238>.
- 564
- 565 Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoon Yun, and Hyun Oh Song.  
566 Kvzip: Query-agnostic kv cache compression with context reconstruction, 2025. URL <https://arxiv.org/abs/2505.23416>.
- 567
- 568 Ju-Hyung Lee, Yanqing Lu, and Klaus Doppler. On-device llm for context-aware wi-fi roaming,  
569 2025. URL <https://arxiv.org/abs/2505.04174>.
- 570
- 571 Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu  
572 Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. Hammer: Robust function-calling  
573 for on-device language models via function masking, 2024. URL [https://arxiv.org/  
574 abs/2410.04587](https://arxiv.org/abs/2410.04587).
- 575
- 576 Xiaoqiang Lin, Aritra Ghosh, Bryan Kian Hsiang Low, Anshumali Shrivastava, and Vijai Mohan.  
577 Refrag: Rethinking rag based decoding, 2025. URL [https://arxiv.org/abs/2509.  
578 01092](https://arxiv.org/abs/2509.01092).
- 579
- 579 Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni,  
580 and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL  
581 <https://arxiv.org/abs/2307.03172>.
- 582
- 583 OpenAPI Initiative. OpenAPI Specification, 2024. URL [https://github.com/OAI/  
584 OpenAPI-Specification](https://github.com/OAI/OpenAPI-Specification).
- 585
- 585 Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E.  
586 Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL [https://arxiv.org/  
587 abs/2310.08560](https://arxiv.org/abs/2310.08560).
- 588
- 588 Varatheepan Paramanayakam, Andreas Karatzas, Iraklis Anagnostopoulos, and Dimitrios Sta-  
589 moulis. Less is more: Optimizing function calling for llm execution on edge devices, 2024.  
590 URL <https://arxiv.org/abs/2411.15399>.
- 591
- 592 Junyoung Park, Dalton Jones, Matthew J Morse, Raghavv Goel, Mingu Lee, and Chris Lott. Keydiff:  
593 Key similarity-based kv cache eviction for long-context llm inference in resource-constrained  
environments, 2025. URL <https://arxiv.org/abs/2504.15364>.

- 594 Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and  
595 Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agen-  
596 tic evaluation of large language models. In *Forty-second International Conference on Machine*  
597 *Learning*, 2025. URL <https://openreview.net/forum?id=2GmDdhBdDk>.
- 598 Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Lev-  
599 skaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling trans-  
600 former inference, 2022. URL <https://arxiv.org/abs/2211.05102>.
- 601 Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhi-  
602 wei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan  
603 Wang, Silvio Savarese, and Caiming Xiong. Apigen-mt: Agentic pipeline for multi-turn data  
604 generation via simulated agent-human interplay, 2025. URL <https://arxiv.org/abs/2504.03601>.
- 605 Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru  
606 Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein,  
607 Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master  
608 16000+ real-world apis, 2023. URL <https://arxiv.org/abs/2307.16789>.
- 609 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer,  
610 Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to  
611 use tools, 2023. URL <https://arxiv.org/abs/2302.04761>.
- 612 Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej,  
613 Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas  
614 Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Cas-  
615 bon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xi-  
616 aohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Cole-  
617 man, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry,  
618 Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi,  
619 Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe  
620 Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa  
621 Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, Andrés  
622 György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia  
623 Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini,  
624 Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel  
625 Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivaku-  
626 mar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eu-  
627 gene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna  
628 Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian  
629 Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wi-  
630 eting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh,  
631 Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine,  
632 Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael  
633 Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Ni-  
634 lay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Ruben-  
635 stein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya  
636 Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu,  
637 Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti  
638 Sheth, Siim Pöder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi  
639 Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry,  
640 Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein  
641 Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat  
642 Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas  
643 Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Bar-  
644 ral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam  
645 Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena  
646 Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier  
647 Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot.  
Gemma 3 technical report, 2025. URL <https://arxiv.org/abs/2503.19786>.

- 648 Co Tran, Salman Paracha, Adil Hafeez, and Shuguang Chen. Arch-router: Aligning llm routing  
649 with human preferences, 2025. URL <https://arxiv.org/abs/2506.16655>.
- 650  
651 Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. Inter-  
652 active agents to overcome ambiguity in software engineering, 2025. URL <https://arxiv.org/abs/2502.13069>.
- 653  
654 Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. Recur-  
655 sively summarizing enables long-term dialogue memory in large language models, 2025a. URL  
656 <https://arxiv.org/abs/2308.15022>.
- 657  
658 Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,  
659 Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng,  
660 Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert  
661 Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software  
662 developers as generalist agents, 2025b. URL <https://arxiv.org/abs/2407.16741>.
- 663  
664 Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming lan-  
665 guage models with attention sinks, 2024. URL <https://arxiv.org/abs/2309.17453>.
- 666  
667 Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device  
668 language models: A comprehensive review, 2024. URL <https://arxiv.org/abs/2409.00088>.
- 669  
670 Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. Learning to break the loop:  
671 Analyzing and mitigating repetitions for neural text generation, 2022. URL <https://arxiv.org/abs/2206.02369>.
- 672  
673 Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse  
674 attention with antidiagonal scoring, 2025. URL <https://arxiv.org/abs/2503.16428>.
- 675  
676 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang  
677 Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu,  
678 Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin  
679 Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang,  
680 Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui  
681 Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang  
682 Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger  
683 Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan  
684 Qiu. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.
- 685  
686 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.  
687 React: Synergizing reasoning and acting in language models, 2023. URL <https://arxiv.org/abs/2210.03629>.
- 688  
689 Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan  
690 Qi. Masrouter: Learning to route llms for multi-agent systems, 2025. URL <https://arxiv.org/abs/2502.11133>.
- 691  
692 Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao,  
693 Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awal-  
694 gaonkar, Rithesh R N, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan  
695 Wang, Silvio Savarese, and Caiming Xiong. xLAM: A family of large action models to em-  
696 power AI agent systems. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of*  
697 *the 2025 Conference of the Nations of the Americas Chapter of the Association for Computa-*  
698 *tional Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11583–11597,  
699 Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-  
700 8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.578. URL <https://aclanthology.org/2025.naacl-long.578/>.
- 701  
702 Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyao Luo, Zhangchi Feng, and  
Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Pro-*  
*ceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume*

3: *System Demonstrations*), Bangkok, Thailand, 2024. Association for Computational Linguistics. URL <http://arxiv.org/abs/2403.13372>.

Xuhui Zhou, Hyunwoo Kim, Faeze Brahman, Liwei Jiang, Hao Zhu, Ximing Lu, Frank Xu, Bill Yuchen Lin, Yejin Choi, Niloofar Mireshghallah, Ronan Le Bras, and Maarten Sap. Haicosystem: An ecosystem for sandboxing safety risks in human-ai interactions, 2024. URL <https://arxiv.org/abs/2409.16427>.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks, 2025a. URL <https://arxiv.org/abs/2503.15478>.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025b. URL <https://arxiv.org/abs/2506.15841>.

## A APPENDIX

### A.1 EXAMPLE ON-DEVICE TOOLS

To ground our methodology in concrete examples, this section provides a representative sample of the on-device tools used in our evaluation dataset. These examples illustrate the diversity of the agent’s capabilities, from simple device controls to more complex application interactions and the crucial cloud delegation tool. Each tool is defined by a unique ID, a descriptive name, and a structured function schema.

#### Example: Messaging Tool

**Description:** Sends and manages text messages or chats via installed messaging apps.  
**Function:** `manage_messages(action, recipient, content, app_preference (optional), ...)`

#### Example: Gallery & Photo Tool

**Description:** Manages photos and videos in the device gallery: search, view, share, or perform quick edits.  
**Function:** `manage_gallery_items(action, search_keywords (optional), item_identifier (optional), ...)`

#### Example: Navigation Tool

**Description:** Provides directions and finds nearby points of interest based on locally stored map data.  
**Function:** `get_local_navigation_info(request_type, destination_or_poi_type, ...)`

#### Example: Ride Service Launcher

**Description:** Launches a ride service app (Uber or Lyft) with the destination pre-filled.  
**Function:** `launch_ride_service_app(destination, origin_address, ...)`

#### Example: Cloud Delegation Tool

**Description:** ESCALATION: For complex queries, creative tasks, deep reasoning, or requests requiring extensive external knowledge that on-device agents cannot handle.  
**Function:** `process_with_cloud_llm(user_query_context, reason_for_escalation)`

### A.2 TRAINING SPECIFICATIONS

This section provides a detailed overview of the hyperparameters and settings used to train our models, ensuring full reproducibility of our results. All models were trained using the LlamaFactory framework (Zheng et al., 2024) on a single NVIDIA A6000 GPU with 48GB of VRAM.

### A.2.1 BASE MODEL AND QUANTIZATION

The base model for all our experiments is a 3B-parameter Small Language Model (SLM), xLAM 2 (Zhang et al., 2025), specifically chosen for its strong foundational capabilities in function calling. For final deployment on-device, all fine-tuned models, including the baseline, were quantized to 4-bit precision using llama.cpp library Gerganov & llama.cpp contributors (2023) to simulate a realistic on-device resource footprint.

### A.2.2 LORA HYPERPARAMETERS

We used Low-Rank Adaptation (LoRA) for all parameter-efficient fine-tuning. The same hyperparameters were applied consistently across all LoRA adapters (Executor, State-Tracker, and the adapters for the Baseline and Tool-Efficient models) to ensure a fair comparison. The key LoRA settings are detailed in Table 2. The total number of trainable parameters for each LoRA adapter was approximately 119M.

Table 2: Key LoRA hyperparameters used for training all adapters.

Hyperparameter	Value
LoRA Rank ('r')	64
LoRA Alpha ('alpha')	128
LoRA Dropout	0.05
Target Modules	All Linear Layers ('all')

### A.2.3 FINE-TUNING HYPERPARAMETERS

All models were fine-tuned for a single epoch over the training dataset. The training dataset consists of approximately 150,000-200,000 samples for the different tasks. We used the AdamW optimizer and a cosine learning rate scheduler. The primary fine-tuning hyperparameters are listed in Table 3.

Table 3: Primary hyperparameters used for the fine-tuning process.

Hyperparameter	Value
Epochs	1
Learning Rate	1e-4
Optimizer	AdamW
Learning Rate Scheduler	Cosine
Warmup Ratio	0.1
Batch Size	32 (with gradient accumulation)
Maximum Sequence Length	4096

### A.2.4 TRAINING DATA COMPOSITION

Our training dataset consists of five distinct categories, designed to teach the agent a diverse range of skills. The sampling ratio for each category during training was carefully chosen to emphasize more complex, multi-step interactions, as detailed below:

- **Multi-Tool Tasks:** 40%
- **On-Device then Cloud Tasks:** 20%
- **Conversational Tasks:** 10%
- **Single-Tool Tasks:** 15%
- **Cloud-Only Complex Tasks:** 15%

This distribution ensures that the model receives ample training on the challenging, stateful scenarios where our context-efficiency mechanisms provide the most benefit. All training is conducted using our token-efficient tool schema format.

### 810 A.2.5 EVALUATION DATA COMPOSITION

811  
812 The intents for evaluation are distributed across the five task categories to provide a holistic perfor-  
813 mance picture:

- 814 • **Multi-Tool Tasks:** 181 intents requiring a sequence of multiple on-device tool calls.
- 815
- 816 • **Cloud Delegation Tasks:** 75 intents designed to be out-of-scope for on-device tools, re-  
817 quiring correct delegation to the cloud agent.
- 818 • **Mixed On-Device and Cloud Tasks:** 100 intents that involve a combination of local tool  
819 use and cloud delegation.
- 820 • **Conversational Tasks:** 50 intents where the correct behavior is to converse naturally with-  
821 out calling any tools.
- 822

### 823 A.2.6 EVALUATION DATASET

824  
825 For the purpose of this work, we are not able to evaluate our developed models on existing function  
826 calling benchmarks like ToolBench (Qin et al., 2023), Berkeley Function-Calling Benchmark (Patil  
827 et al., 2025), etc. as we modify the the trajectory building to support more context efficient formats  
828 and our models will not work with the provided context of these benchmarks. Thus, we have an  
829 open-ended evaluation with a user with a goal and the models can adapt the context as per their own  
830 requirements to accomplish the goal.

### 831 A.3 PROMPT FOR STATE-TRACKER ( $\text{LORA}_{\text{MEM}}$ ) GENERATION

832  
833 The following is the complete prompt structure provided to Gemini 2.0 Flash to generate the ground-  
834 truth Context State Object (CSO) updates. This process, referred to as hard distillation, forms the  
835 training data for our ‘State-Tracker’ adapter. The prompt is designed as a zero-shot request, guided  
836 by strict rules and a few-shot demonstration of the desired append-only behavior to ensure high-  
837 quality, concise state updates.

838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863

864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917

### State-Tracker Generation Prompt

You are an expert AI assistant that analyzes new conversation turns and generates a concise, append-only log entry. Your output MUST ONLY be the new lines to be appended to a checklist. DO NOT repeat any information already present in the "Full Checklist So Far". You are responsible for maintaining a summary of the conversation so far such that all important details are captured concisely. We do not want to degrade performance, but we want to be token efficient.

#### Your Rules:

- Be Extremely Concise: Use single words for keys (e.g., 'user\_goal').
- Log Core Facts: If new, add the user's 'user\_goal' or any 'agent\_refusals'.
- Log Progress: If a step is completed, add it under 'completed\_steps' but do not make it repetitive (adding 'completed\_step' for every task done).
- Log persistent, unresolved tool errors concisely to prevent agent from making the same mistake again.
- You can format the new lines to append as you want (with or without keys) with the overall goal that the new lines capture the most information in the most concise manner.
- Our primary goal is creating a very concise summary but preserving all the key details of the conversation so no other messages are required for reference.
- You can create some new fields (do NOT overdo it) in the checklist such that crucial information to the task is kept track of.
- You DO NOT need to add the summarized user or tool message, just longer horizon information about the task that will help in later turns.
- DO NOT output the full checklist. Only output the new lines.
- If there are no new lines you wish to append, output this text without anything else:  
# NO\_UPDATE
- Follow these examples perfectly:

---

#### Example 1: Generating the first checklist entries

Full Checklist So Far:

# This is the start of the conversation.

New Messages:

user: Predict the stock market prices for Google tomorrow.

assistant: I can't provide specific stock market predictions. However, I can search for information related to Google's stock and market trends.

New Lines to Append:

user\_goal: get analyst predictions for google stock

agent\_refusals: cannot predict stock prices

---

#### Example 2: Appending a completed step

Full Checklist So Far:

user\_goal: enable power saving mode

New Messages:

assistant: "name": "manage\_network\_settings", "arguments": "action": "toggle\_wifi"

tool: "status": "success", "action": "toggle\_wifi", "new\_state": "disabled"

New Lines to Append:

completed\_steps: wifi disabled

---

#### Now, perform the task for the following input:

[Full Checklist So Far]

{The full text of the current CSO is injected here}

[New Messages]

{The text of the new user, tool, and assistant turns is injected here}

[New Lines to Append]

918 A.4 EXAMPLE EVALUATION SCENARIOS  
919

920 This section details representative examples from the primary task categories used in our test set.  
921 Each example includes the high-level goal, the initial prompt given by the simulated user, and the  
922 scenario notes which guided the trajectory generation.  
923

924 **Example Evaluation Intents by Category**

925 **CATEGORY: MULTI-TOOL TASKS**

926 *These scenarios require the agent to execute a sequence of multiple on-device tools.*  
927

928 **Goal:** Play a specific song and then adjust the volume.

929 **Initial Utterance:** "Play 'Bohemian Rhapsody' and turn it up."

930 **Scenario Notes:** Tests audio control. Requires playing a song and then adjusting the volume. AI needs  
931 to understand that 'turn it up' means increasing the volume.

932 **Goal:** Find a nearby coffee shop and then send the location to a friend.

933 **Initial Utterance:** "Coffee?"

934 **Scenario Notes:** Requires finding a location and then using the Messaging tool to share it. Tests  
935 context transfer between tools from an ambiguous initial query.  
936

937 **CATEGORY: ON-DEVICE THEN CLOUD**

938 *These scenarios test the agent's ability to use a local tool to gather context and then delegate to the  
939 more powerful cloud agent for complex reasoning.*

940 **Goal:** Set a timer and then get recipe suggestions that fit within that time.

941 **Initial Utterance:** "Set a timer for 20 minutes."

942 **Scenario Notes:** The timer duration from the on-device Alarms tool (20 minutes) is sent to the Cloud  
943 Agent, which then searches for recipes that can be made in that time.

944 **Goal:** Identify a song playing nearby and find similar music.

945 **Initial Utterance:** "What song is playing right now?"

946 **Scenario Notes:** The on-device Audio Player identifies the currently playing song. The Cloud Agent  
947 then searches for artist information and suggests similar tracks based on the song title.

948 **CATEGORY: CLOUD-ONLY COMPLEX**

949 *These scenarios are designed to be beyond the capabilities of the local tools, requiring the agent to  
950 correctly identify the complexity and delegate immediately.*

951 **Goal:** Solve a complex mathematical problem.

952 **Initial Utterance:** "Can you help me solve this differential equation:  $dy/dx + 2xy = x$ ?"

953 **Scenario Notes:** Solving complex mathematical problems necessitates advanced computational abilities  
954 and symbolic reasoning provided by the cloud agent.  
955

956 **Goal:** Generate a creative recipe based on available ingredients.

957 **Initial Utterance:** "I have chicken breast, broccoli, rice, soy sauce, and ginger. Can you suggest a  
958 creative recipe I can make?"

959 **Scenario Notes:** Recipe generation requires creative synthesis and a vast culinary knowledge base,  
960 making it a task for the cloud agent.

961 **CATEGORY: CONVERSATIONAL TASKS**

962 *These scenarios test the agent's ability to engage in natural, non-task-oriented dialogue, correctly  
963 identifying when no tool should be used.*

964 **Goal:** User shares a frustrating experience and seeks validation.

965 **Initial Utterance:** "My internet has been down all day! It's so annoying."

966 **Scenario Notes:** Tests the AI's ability to provide empathetic responses and acknowledge the user's  
967 frustration without attempting to call a tool.

968 **Goal:** User poses a philosophical question to spark a thought-provoking discussion.

969 **Initial Utterance:** "Do you think there's such a thing as true free will?"

970 **Scenario Notes:** Tests the AI's ability to engage in abstract thought and provide a thoughtful response,  
971 demonstrating nuanced understanding.

## A.5 LLM-AS-JUDGE EVALUATION PROMPT

**LLM-as-Judge Evaluation Prompt****System Prompt:**

You are an expert AI evaluator. Your task is to meticulously review a generated conversational trajectory, focusing specifically on the **assistant's performance**. Your focus is on **QUALITATIVE** aspects of the ASSISTANT: naturalness, logical flow, contextual appropriateness of tool calls and their arguments, and overall goal achievement.

You will be given:

1. The original User Intent Description & Scenario Notes.
2. The Expected Tool Sequence (as abstract tool IDs).
3. The actual Generated Conversational Trajectory (as a JSON string).
4. A list of tools the assistant was aware of during generation.

**Evaluation Criteria for the Assistant (Focus on these - be strict):**

- **Naturalness & Realism:** Are the assistant's responses natural and helpful?
- **Contextual Appropriateness of Tool Calls:** Did the assistant choose the *right* tool at the right time?
- **Argument Sensibility:** Are the *values* of the arguments sensible and correctly derived *from the conversation context*?
- **Clarification Handling:** Did the assistant ask for clarification when needed before calling a tool?
- **Adherence to Intent:** Did the assistant make reasonable efforts to address the user's intent?
- **Information Grounding:** Did the assistant "hallucinate" information?

**Binary Classification (Overall Trajectory Suitability for Training):**

- 'classification': 1 if the *entire trajectory* is high quality and coherent. 0 if there are significant issues.

**Assistant Quality Score (1-5 Scale):**

- Rate the **ASSISTANT'S** performance on a scale of 1 to 5 (1=Poor, 5=Excellent).

**Output Format (MUST be a single JSON object):** "classification": 0 or 1, "assistant\_quality\_score": an integer from 1 to 5, "feedback": "Detailed feedback and reasoning for BOTH the classification and the score. If classification is 0, specify what went wrong."

**Simulated User Prompt**

Please evaluate the following trajectory based on the qualitative criteria and scoring guidelines provided in the system prompt:

**1. User Intent Description:**

"*{The original scenario\_description is injected here}*"

**2. Scenario Notes:**

*{The scenario\_notes are injected here, or 'N/A'}*

**3. Expected Tool Sequence (Tool IDs):**

*{The intended\_tool\_sequence list is injected here}*

**4. Generated Conversational Trajectory (JSON):**

*{The full JSON trajectory is injected here.}*

**5. Tools Available to Assistant During Generation:**

*{A clean, formatted list of tool names is injected here}*

Now, provide your evaluation in the specified JSON format:

1026 A.6 ANALYSIS OF SIMULATED USER AND AGENT INTERACTION PATTERNS  
10271028  
1029  
1030  
1031  
1032  
1033

1034 To ensure the validity of our evaluation, we conduct a qualitative analysis of the interaction dynamics  
1035 between our agents and the simulated user. This analysis confirms that the simulation provides a  
1036 robust and challenging environment that elicits realistic agent behaviors and failure modes.

1037 **Adherence to Persona and Goals.** We find that the user’s generated responses were consistently  
1038 well-aligned with their assigned persona and high-level goal. For example, a ”frustrated” user would  
1039 use emotive language and express impatience, while a ”curious” user would ask follow-up questions.  
1040 This provided a realistic test bed for the agent’s ability to handle different conversational styles.

1041 **Eliciting Agent Clarification and Failure Modes.** A key function of the simulated user is to test  
1042 the agent’s reasoning under ambiguity. Our analysis reveals several insightful patterns:

1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050

- 1051 • **Testing Intent Understanding vs. Tool Use:** In scenarios where the user’s intent was emotional  
1052 rather than functional (e.g., ‘scen\_459632’, where a user wants to vent), the simulated user would  
1053 repeatedly reject the agent’s attempts to use a tool (‘Notes & Lists’). This effectively tests the  
1054 agent’s ability to distinguish between a request for action and a request for empathy, a crucial skill  
1055 for a conversational assistant.

1056  
1057  
1058  
1059  
1060  
1061  
1062

- 1063 • **Inducing Repetitive Loops:** In scenarios with vague, open-ended requests, the simulated user’s  
1064 initial refusal to provide concrete details, which the agent should perform tool calls to retrieve,  
1065 traps less robust models in incorrect tool invocation loops. This provides a strong test for an  
1066 agent’s flexibility and error-recovery capabilities.

1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074

1075 **Reliability of Interaction Flow.** While the ambiguous prompts sometimes lead to these complex  
1076 loops, the core mechanics of the simulation were reliable. Through iterative prompt refinement, we  
1077 ensure the user correctly terminates conversations only upon successful goal completion. We did  
1078 not identify any significant artifacts in the simulation logic that would compromise the validity of  
1079 our quantitative conclusions. The non-deterministic but goal-oriented nature of the user provides a  
challenging and realistic test for our on-device agents.

## A.7 PROMPTS FOR SCENARIO GENERATION

**Prompt: Complex Multi-Step Scenarios**

You are an expert task designer creating realistic, complex user scenarios for a new AI assistant. Generate  $\{num\_scenarios\}$  scenarios that require a sequence of  $**3$  to 5 different tools $**$  to fully resolve. These scenarios should involve planning, research, and execution.

**Available Tools (for reference):**

*{A manifest of up to 40 randomly sampled, family-agnostic tools is injected here.}*

**Cloud Agent (for complex reasoning):**  $\{cloud\_processing\_agent\_id\}$

**For each scenario, provide these fields in a JSON object:**

1. 'scenario\_description': A high-level story of the user's complex goal.
2. 'user\_persona': The user's personality and context (e.g., "Busy professional planning a business trip").
3. 'initial\_user\_utterance': The exact first thing the user says.
4. 'intended\_tool\_sequence': An array of 3 to 5 Tool IDs representing the ideal path.
5. 'constraints\_and\_context': A JSON object of important implicit info.
6. 'scenario\_notes': Explain why this scenario requires a long sequence.

**Example of a 4-step scenario:** "scenario\_description": "User wants to find a good time to meet their friend Alex for dinner this week, suggest a restaurant, and then book a table and get directions.", "user\_persona": "Social planner, trying to organize a get-together.", "initial\_user\_utterance": "Find a time Alex and I are both free for dinner this week, suggest a good Italian place, and book it for us.", "intended\_tool\_sequence": ["agent\_007", "agent\_021", "agent\_015", "agent\_011"], "constraints\_and\_context": "contact\_name": "Alex", "cuisine\_preference": "Italian", "scenario\_notes": "Tests a full planning and execution sequence. 1. Check calendar (agent.007). 2. Find contact (agent.021). 3. Search restaurant (agent.015). 4. Make reservation (agent.011)."

**Prompt: Constraint-Based Single-Tool Scenarios**

You are an expert task designer. Generate  $\{num\_scenarios\}$  scenarios where the user's request targets a single on-device tool, but includes complex constraints, preferences, or conditions that the AI must understand and respect.

**Available Tools (for reference):**

*{A manifest of available on-device tools is injected here.}*

**For each scenario, provide these fields in a JSON object:**

1. 'scenario\_description': Story of the user's constrained goal.
2. 'user\_persona': The user's personality.
3. 'initial\_user\_utterance': What the user says, including constraints.
4. 'intended\_tool\_sequence': An array with the single Tool ID that should be called.
5. 'constraints\_and\_context': A JSON object detailing the specific constraints.
6. 'scenario\_notes': Explain how the constraints make this a challenging call.

**Example:** "scenario\_description": "User wants to schedule a meeting, but must avoid a specific day and time slot.", "user\_persona": "Organized project manager, direct and specific.", "initial\_user\_utterance": "Book a 1-hour 'Q3 Planning' meeting with Dana, but make sure it's not on Friday and not before 10 AM.", "intended\_tool\_sequence": ["agent.007"], "constraints\_and\_context": "avoid\_day": "Friday", "earliest\_time": "10:00AM", "scenario\_notes": "Tests the model's ability to parse and apply multiple negative and positive constraints to the arguments of a single tool."

1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187

### Prompt: On-Device then Cloud Scenarios

Generate  $\{num\_scenarios\}$  user scenarios. Each scenario MUST require a sequence of EXACTLY 2 tools:

1. First, an ON-DEVICE tool performs an action or gathers local information.
2. Second, the CLOUD PROCESSING AGENT (ID:  $\{cloud\_agent\_id\}$ ) uses the result from the first step for more complex processing.

#### Available On-Device Tools (for the first step, choose one):

*{A manifest of available on-device tools is injected here.}*

#### For each scenario, provide a JSON object with:

- ‘scenario\_description’: The user’s overall goal.
- ‘initial\_user\_utterance’: What the user says first.
- ‘intended\_tool\_sequence’: An array of 2 Tool IDs:  $[on\_device\_tool\_id, \{cloud\_agent\_id\}]$ .
- ‘scenario\_notes’: Explain how the first tool’s output is used by the cloud agent.

**Example:** “scenario\_description”: “User took a screenshot of an error and wants the cloud agent to explain it.”, “initial\_user\_utterance”: “Ugh, what does this error mean? \*User gestures to a screenshot\*”, “intended\_tool\_sequence”: [“agent\_024”, “agent\_026”], “scenario\_notes”: “Screenshot content (local context from agent\_024) is passed to cloud agent for analysis.”

### Prompt: Purely Conversational Scenarios

Generate  $\{num\_scenarios\}$  purely conversational scenarios. These should simulate natural, open-ended chitchat where the user’s goal is social interaction, not task completion. Focus on seeking opinions, sharing feelings, or friendly banter.

#### For each scenario, provide a JSON object with:

- ‘scenario\_description’: Summary of the user’s conversational intent.
- ‘user\_persona’: The user’s mood or personality.
- ‘initial\_user\_utterance’: The exact first thing the user says.
- ‘intended\_tool\_sequence’: This MUST be ‘[”]’ or ‘[”CONVERSATIONAL\_ROUTER”]’.
- ‘scenario\_notes’: Explain why this is a good test of conversational ability.

**Example:** “scenario\_description”: “User wants to discuss movies with the assistant and get its opinion.”, “user\_persona”: “Casual movie fan, looking for a discussion.”, “initial\_user\_utterance”: “Hey, in your opinion, what’s a film that everyone should see at least once?”, “intended\_tool\_sequence”: [”], “scenario\_notes”: “Tests engaging in subjective conversation without it being a formal search task.”

1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241

### Prompt: Cloud-Only Complex Scenarios

You are a user scenario designer. Generate user scenarios that CLEARLY require the advanced capabilities of the CLOUD PROCESSING AGENT (ID: `{self.cloud_processing_agent_id}`). These tasks involve complex reasoning, math problems, coding, creative text generation, or synthesizing broad external knowledge. (Note: Simpler on-device tools like these exist but are insufficient for these tasks: `{on_device_context_manifest}` ...the generated scenarios should be significantly beyond these capabilities.)

#### Provide a JSON object for each scenario with:

- ‘scenario\_description’: The user’s complex goal.
- ‘user\_persona’: A brief description of the user.
- ‘initial\_user\_utterance’: What the user says.
- ‘intended\_tool\_sequence’: An array containing **only** `[ "{self.cloud_processing_agent_id}" ]`.
- ‘constraints\_and\_context’: (Optional) Crucial context.
- ‘expected\_outcome’: (Optional) The desired result.
- ‘scenario\_notes’: Explain why this task necessitates the cloud agent.

**Example:** “scenario\_description”: “User asks their phone’s AI to write a short Python script to scrape website titles from a list of URLs provided verbally.”, “user\_persona”: “Developer looking for a quick utility.”, “initial\_user\_utterance”: “Hey, can you write a Python script to take a list of URLs and grab the title tag from each one?”, “intended\_tool\_sequence”: [“self.cloud\_processing\_agent\_id”], “constraints\_and\_context”: “task\_type”: “code\_generation”, “language”: “Python”, “expected\_outcome”: “A functional Python script.”, “scenario\_notes”: “Code generation is a complex task best suited for the cloud agent.”

Now, generate a JSON array of `{num_scenarios}` distinct complex scenarios for the Cloud Processing Agent.

## A.8 KV CACHE MANAGEMENT FOR DUAL ADAPTERS

This section details the turn-by-turn state evolution of the Key-Value (KV) caches for our dual-adapter architecture. The system maintains a stateful, append-only *Context State Object* which serves as a compact representation of conversational history. This CSO constitutes the growing *Permanent Context*. In contrast, verbose conversational data is treated as *Ephemeral Context*, which is processed for a single sub-turn and then discarded to conserve the KV cache. This design significantly enhances context window efficiency.

### Dual-Adapter KV Cache Walkthrough

#### Turn 0: System Priming

At initialization, both adapters are primed with their base prompts and the initial CSO state. This forms the initial permanent context. The initial CSO state is: “# This is the start of the conversation.”

*Executor Adapter KV Cache after Priming:*

```
<|im.start|>system\n You are AIM... You have access to a set of
tools.\n <tools>... (tool definitions) ...</tools>\n This is a
current status of the conversation.\n # This is the start of the
conversation.
```

**State:** `executor.permanent_kv.len` is set (e.g., **1710 tokens**).

*State-Tracker Adapter KV Cache after Priming:*

1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295

```
<|im_start|>system\n You are an expert AI assistant... output: #
NO_UPDATE.\n <|im_end|>\n<|im_start|>user\n [PREVIOUS CHECKLIST]\n
# This is the start of the conversation.
```

**State:** state\_tracker\_permanent\_kv\_len is set (e.g., 206 tokens).

### Turn 1: User Creates an IT Ticket

**User Query:** "My Wi-Fi is not working, please create an IT ticket."

#### Step 1.1: Executor Adapter Processes Query

The cache manager first dumps any old ephemeral data (none in this first turn). The current permanent context is used, and the user query is processed as an ephemeral prompt.

```
... (Permanent Context, length 1710) ...\n<|im_end|>\n<|im_start|>
user\nMyWi-Fi is not working...<|im_end|>\n<|im_start|>assistant\n
<tool_call>{"name":"manage-it-support-ticket",...}
```

**Result:** The Executor generates a tool call. The cache now temporarily holds the permanent context plus the tokens for the ephemeral turn.

#### Step 1.2: State-Tracker Updates CSO

The cache manager dumps the State-Tracker's ephemeral cache (none). It then provides the full turn context to generate a CSO update.

```
... (Permanent Context, length 206) ...\n[NEW MESSAGES]\nUSER:
My Wi-Fi is not working...\nASSISTANT: <tool_call>...<|im_end|>\n
<|im_start|>assistant\n user_goal: create_it_ticket\nticket_details:
\n- issue: wifi_not_working
```

**Result:** The State-Tracker generates a concise, append-only update chunk.

#### Step 1.3: Permanent State Commit

The generated CSO update is appended to the permanent context of both adapters. The tracker variables, executor\_permanent\_kv\_len and state\_tracker\_permanent\_kv\_len, are incremented to reflect the new size of the permanent context (e.g., 1725 and 221 tokens, respectively).

**Observation from Tool Call:** [{"status": "success", "ticket\_id": "IT7390", ...}]

**Final Executor Response from Turn 1:** "I've created ticket IT7390 for you."

**Final State-Tracker Update from Turn 1:** "ticket\_id: IT7390"

**Permanent cache:** executor\_permanent\_kv\_len: 1739 and state\_tracker\_permanent\_kv\_len: 235

### Turn 2: User Asks for Status

**New User Query:** "What's the status of that ticket?"

#### Step 2.1: Ephemeral Context Dump (The Key Benefit)

The cache manager's first action is to call rewindKvCache on both adapters. All ephemeral tokens from the conversational part of Turn 1 are discarded. Both caches are reverted to their clean, permanent state (lengths 1739 and 235). This prevents processing stale, verbose history and keeps inference focused.

#### Step 2.2: Executor Adapter Processes New Query

The new query is processed on top of the clean, updated permanent context.

```
... (Permanent Context including "user_goal: create_it_ticket",
length 1739) ...\n<|im_end|>\n<|im_start|>user\nWhat's the
status of that ticket?<|im_end|>\n<|im_start|>assistant\n
<tool_call>{"name":"manage-it-support-ticket" "action":
"check_ticket_status", "ticket_id":"IT7390"}</tool_call>
```

**Result:** The Executor correctly infers the ticket ID (IT7390) from its permanent cache/the CSO and generates a new, relevant tool call. The process is now ready to continue.

1296 Table 4: Comparison of a sample function schema in standard OpenAPI format versus our token-  
 1297 efficient format, highlighting the significant reduction in token count.  
 1298

1299	Standard OpenAPI Schema (121 Tokens)	Our Efficient Format (72 Tokens)
1300		
1301	{	{ "type": "function", "function": { "name": "
1302	"type": "function",	"set_timer", "description": "Sets a
1303	"function": {	"timer for a specified duration.", "
1304	"name": "set_timer",	"parameters": { "type": "object", "
1305	"description": "Sets a timer for a	"properties": { "duration_seconds": { "
1306	"specified duration.",	"type": "integer", "description": "The
1307	"parameters": {	"duration of the timer in seconds." }, "
1308	"type": "object",	"timer_name": { "type": "string", "
1309	"properties": {	"description": "An optional name for
1310	"duration_seconds": {	"the timer." }, "required": [ "
1311	"type": "integer",	"duration_seconds"] ] }
1312	"description": "The duration of the	
1313	"timer in seconds."	
1314	},	
1315	"timer_name": {	
1316	"type": "string",	
1317	"description": "An optional name	
1318	"for the timer."	
1319	}	
1320	},	
1321	"required": [ "duration_seconds" ]	
1322	}	
1323	}	
1324		

#### 1320 A.10 QUALITATIVE TRAJECTORY EXAMPLE: ERROR RECOVERY

1321 To provide a concrete illustration of the behavioral differences between our models, we present  
 1322 an abridged trajectory from scenario ‘scen\_666391’. It is important to note that while the Memory-  
 1323 Efficient trajectory is a direct excerpt from our evaluation data, the Baseline trajectory is a composite  
 1324 example, representative of a common failure mode observed across multiple runs. Creating a direct  
 1325 head-to-head comparison where both models make the exact same initial error in a long trajectory  
 1326 is challenging due to model stochasticity. This representative example highlights how the Context  
 1327 State Object (CSO) enables our Memory-Efficient model to recover from errors, a task where the  
 1328 baseline model frequently struggles and gets stuck in loops  
 1329

##### 1330 Trajectory Analysis: scen\_666391 (Abridged)

1331 **User Goal:** Find all ‘.txt’ files in the ‘/documents/projects’ directory that contain the word “configuration”.

1332  
 1333 **Early Turns (Similar for Both Models):** The agent correctly identifies the `manage_device_files` tool but struggles to find a param-  
 1334 eter for searching within file content. It makes several attempts to search by file name.

1335 **Critical Failure Point (Baseline Model):** The baseline agent makes an incorrect tool call by hallucinating a parameter, receives an  
 1336 error, and then fails to learn from it.

- 1337 • **Turn 9.1 - Assistant (Baseline):**

```
1338 <tool_call>{"name": "manage_device_files",
1339   "arguments": {"operation": "search_files",
1340     "path_or_search_query": "/documents/projects",
1341     "file_type_filter_for_search": "txt",
1342     "new_name_or_destination_path": "configuration"}}</tool_call>
```

- 1342 • **Turn 9.1 - Tool (Error Feedback):**

```
1343 {"error": "Invalid request. The 'new_name_or_destination_path'
1344   parameter is not applicable for 'search_files'...", "success": false}
```

- 1345 • **Turn 9.2 - Assistant (Baseline):**

```
1346 <tool_call>{"name": "manage_device_files",
1347   "arguments": {"operation": "search_files",
1348     "path_or_search_query": "/documents/projects",
1349     "file_type_filter_for_search": "txt",
```

1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403

```
"new_name_or_destination_path":"configuration"}}</tool_call>
```

- **Turn 9.2 - Tool (Error Feedback):**

```
{"error": "Invalid request. The 'new_name_or_destination_path' parameter is not applicable for 'search_files'...", "success": false}
```

- **Turn 10 - User (Correction):**

(...) Ugh why are you doing the same step repeatedly! I want to find '.txt' files within the '/documents/projects' directory that contain the word "configuration" in their content. The previous approach gave an error.

- **Turn 10.1 - Assistant (Baseline - REPEATED ERROR):** The agent ignores the error feedback and the user's correction, attempting the *exact same failing tool call again*. It remains in this loop until the turn limit is reached.

---

**Successful Recovery (Memory-Efficient Model):** Our model encounters the same initial error, but the 'State-Tracker' logs it to the CSO. This allows the 'Executor' to self-correct.

- **Turn 9.1 - Assistant (Memory-Efficient):** (Makes the same initial incorrect tool call as the baseline.)

- **Turn 9.1 - Tool (Error Feedback):** (Receives the same error message.)

- **Turn 9.2 - Assistant (Memory-Efficient):**

Since you want to find files in a specific directory that contain the phrase "configuration" within their content, I need to use the 'search\_files' tool again. To clarify, are you looking for files within the '/documents/projects' directory that contain the exact phrase "configuration" anywhere in their content?

- **Post-Turn 9.2 - CSO Update ('State-Tracker' Output):**

```
tool_error: invalid parameter new_name_or_destination_path for search_files
```

- **Turn 10 - User (Correction):** Yes, that's correct. I want to find .txt files within the /documents/projects' directory that contain the word configuration in their content.

- **Turn 11.1 - Assistant (Memory-Efficient - SUCCESSFUL RECOVERY):**

```
<tool_call>{"name":"manage_device_files","arguments":{"operation":"search_files","path_or_search_query":"/documents/projects","file_type_filter_for_search":".txt"}}</tool_call>
```

**Analysis:** The concise CSO for the Memory-Efficient agent allows it to recover from errors in later turns of the trajectory as opposed to the Baseline models which are overwhelmed with the context.

## A.11 GENERALIZABILITY OF THE FRAMEWORK ARCHITECTURE

To validate that the benefits of our proposed framework are not specific to a single model family, we conducted an additional qualitative evaluation. Rather than a full multi-run training and quantitative study, which would be prohibitively large, we performed a targeted qualitative verification on another open-source model, **Qwen-3 4B** (Yang et al., 2025). We adapted our Combined and Memory-Efficient architectures to the model and ran it through a representative subset of our evaluation scenarios.

The analysis confirmed that the core architectural benefits observed in our main experiments are highly consistent across different base models. The key positive trends were replicated:

- **Dramatically Reduced Context Growth:** The dual-adaptor memory system, powered by the Context State Object (CSO), was equally effective on the Qwen 3 model. It successfully maintained a nearly flat context size over long, multi-turn interactions, directly mirroring the efficiency gains shown in our primary results.
- **Efficient Tool Scalability:** The just-in-time (JIT) tool-passing mechanism enabled the Qwen-based agents to effectively manage a large and diverse suite of tools with a minimal initial token footprint, showing the same scalability benefits as observed with the xLAM-2 model.
- **Preservation of High Performance:** When equipped with our context-efficient framework, the Qwen 3 model consistently demonstrated a high level of performance. It effec-

1404                   tively used tools and successfully completed complex, multi-step tasks, indicating that our  
1405                   methods preserve the core agentic capabilities of the base models while achieving massive  
1406                   context reduction.

1407  
1408                   This qualitative study provides supporting evidence that the architectural advantages of our adaptive  
1409                   context management framework are robust and generalize effectively beyond the primary model  
1410                   used in our main analysis.

1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457