

Foundation First: Stabilizing SLM Agents via Pre-execution Logical Scaffolding

Juntae Lee¹ Jihwan Bang¹ Seunghan Yang¹ Sungha Choi^{2*†}

¹Qualcomm AI Research[‡] ²Kyung Hee University

{juntlee, jihwbang, seunghan}@qti.qualcomm.com sunghac@khu.ac.kr

Abstract

In on-device agentic systems, Small Language Models (SLMs) are essential for ensuring privacy-preserving personalization and low-latency execution. However, despite their capacity to handle isolated atomic tasks, during complex multi-step orchestration SLMs frequently suffer from stochastic drift—a phenomenon where minor logical deviations cascade into irrecoverable trajectory failures. In this work, we demonstrate the primacy of pre-execution planning, revealing that mitigating initial logical drift is vastly more effective than mid-trajectory interventions. Building on this insight, we propose **FORST** (FOundation fiRST) framework that implements sequential logical scaffolding in pre-execution planning, where predefined expert anchors are injected as fixed pre-computed KV-cache states. While the scaffolding ensures logical stability, KV-cache reuse efficiently realizes the structured anchor prefix without repeatedly prefilling static anchors. Then, we can maintain a monolithic reasoning chain within predefined logical planning boundaries and effectively eliminate the redundant computational overhead of structured anchors. Extensive evaluations on the MCP Eval benchmark demonstrate that establishing this rigid logical foundation via **FORST** achieves competitive or numerically higher performance than frontier models, while achieving an $8.7\times$ latency reduction on edge-device platforms.

1 Introduction

The rapid evolution of large language models (LLMs) (Achiam et al., 2023; Gemini Team, Google, 2025; Dubey et al., 2024; An Yang, 2023; Jiang et al., 2023) has catalyzed the development of autonomous agentic systems (OpenClaw, 2026;

Claude Code, 2026; Manus AI, 2026) capable of adaptive tool use and sequential decision-making. To deploy these agents for nuanced, personalized services involving sensitive user data (e.g., health records, personal schedules, financial data), computation must migrate from cloud infrastructure to edge devices. Consequently, Small Language Models (SLMs) have emerged as the central orchestrators for on-device workflows. Yet, managing multi-step agentic trajectories remains a critical bottleneck for SLMs. While they succeed at individual tool invocations, they frequently succumb to stochastic drift and hallucinatory loops when attempting to synthesize sequential observations to achieve a final goal (Belcak et al., 2025).

Moreover, unlike large-scale cloud models, on-device SLMs cannot rely on extensive task-specific fine-tuning: specialized small models undermines generalization, and operating multiple specialist SLMs is computationally infeasible in constrained environments. This makes efficient utilization of a single small model, without retraining, the central challenge for agentic orchestration.

To bridge the gap in agentic capabilities, reasoning-based test-time scaling (Snell et al., 2025) serves as a pivotal strategy to transcend the inherent constraints of SLMs. However, direct use of unconstrained open-ended autonomous reasoning (Chain of Thought (Wei et al., 2023), Chain of Note (Yu et al., 2024), Plan-and-Solve (Wang et al., 2023)) results in excessive computational overhead and increased logical fragility in SLM. Whereas, structured prompting enables constrained reasoning. However, existing works are cumbersome, necessitating repeated re-prefilling or leaning on the SLM’s unreliable self-correction during execution. Namely, they largely overlook the unique constraints of SLMs within complex agentic workflows, reflecting the limited attention given to reasoning in this context.

To design an efficiency-driven orchestration

*Work completed while employed at Qualcomm AI Research.

†Corresponding author.

‡Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

strategy, we first systematically analyze the optimal placement of reasoning within agentic workflows. Our empirical investigation reveals the primacy of pre-execution planning: the structural integrity of the initial plan is the single most significant determinant of task success for SLMs. Errors introduced at the onset act as poisoned context, making subsequent turn-wise reasoning or final response synthesis largely ineffective.

Motivated by this, we introduce the FOundation fiRST (**FORST**) framework, a novel approach that stabilizes SLM agents via Initial Logical Scaffolding. Rather than depending on a single macro-instruction to generate a plan, we employ expert-guided sequential prefilling. By sequentially injecting predefined logical priors (anchors) into the KV-cache and guiding the SLM to generate rationales conditioned on the preceding fixed anchor, we establish a robust logical foundation. This structural conditioning prevents the model from deviating into irrelevant logical paths, effectively bridging the execution-completion gap without the prohibitive overhead of long-thought verbosity or iterative tree-search.

The primary contributions of this paper:

- **The Primacy of Planning Insight:** We experimentally demonstrate that pre-execution structural grounding is significantly more effective than turn-wise or final-stage modifications for SLM agents, identifying the critical leverage point for preventing stochastic drift.
- **Logical Scaffolding via Sequential Prefilling:** We propose a training-free framework that guides SLM reasoning into a validated trajectory using expert-guided anchors, constraining the generation through structured logical scaffolding.
- **On-Device Efficiency Frontier-Level Performance:** By maintaining a monolithic chain and leveraging KV-cache injection for predefined anchors, our framework drastically reduces inference latency achieving initial planning up to 8.7× faster than the comparable 4B thinking model. Also, it achieves competitive accuracy with frontier proprietary models (e.g., GPT-4o, Gemini 2.5 Pro), thereby enabling SLMs to deliver reliable performance on complex agentic benchmarks within constrained on-device environments.

w/o CoT	(Naive CoT) Reasoning Placement				
	Init. Plan	Turn-wise (early half)	Turn-wise (last half)	Turn-wise	Final Synth.
62.9	67.3	64.9	64.3	64.2	64.4

Table 1: **Impact of Reasoning Placement in SLMs (Qwen3-4B-Inst).** Average completion (Comp.) accuracy (%) on the MCP Eval benchmark shows the primacy of initial planning over mid-trajectory interventions.

2 Method

We present **FORST** (FOundation fiRST), a framework designed to stabilize and efficiently scale the performance of pre-trained SLMs in on-device agentic workflows. Our systematic analysis demonstrates that among all possible stages for reasoning intervention, the initial planning phase constitutes the absolute bottleneck for multi-step agentic tasks. By constraining the model’s reasoning path with predefined structural logic at this formative stage, **FORST** establishes a consistent logical foundation that prevents stochastic drift before the execution loop even commences.

2.1 Single-Chain Structured Reasoning for SLM

We define the reasoning process as a trajectory $\mathcal{P}$, which serves as the logical precursor to any external action or final response. In conventional unconstrained autonomous reasoning, the model autoregressively generates this path $\mathcal{P} = \{r_1, r_2, \dots, r_T\}$ as a continuous sequence of rationales r . The probability of the reasoning path given an input context x is formulated as:

$$P(\mathcal{P} \mid x) = \prod_{t=1}^T P(r_t \mid x, r_{<t}). \quad (1)$$

In SLMs, this unconstrained autoregressive generation is highly susceptible to logical divergence. To mitigate this, while complex structured branching heuristics (e.g., tree-search) are often employed by exploring multiple reasoning paths, they significantly escalate computational overhead, making them impractical for resource-constrained SLMs.

Instead, we formulate structured reasoning as the sequential injection of predefined fixed planning cues within a single chain. As depicted in Fig. 1(b), **FORST** forces the reasoning path into an interleaved sequence of *structural anchors* (e_k) and model-generated rationales (g_k), where $k = 1, \dots, K$.

The resulting structurally forced reasoning path is represented as an ordered sequence: $\mathcal{P}_{\text{struct}} =$

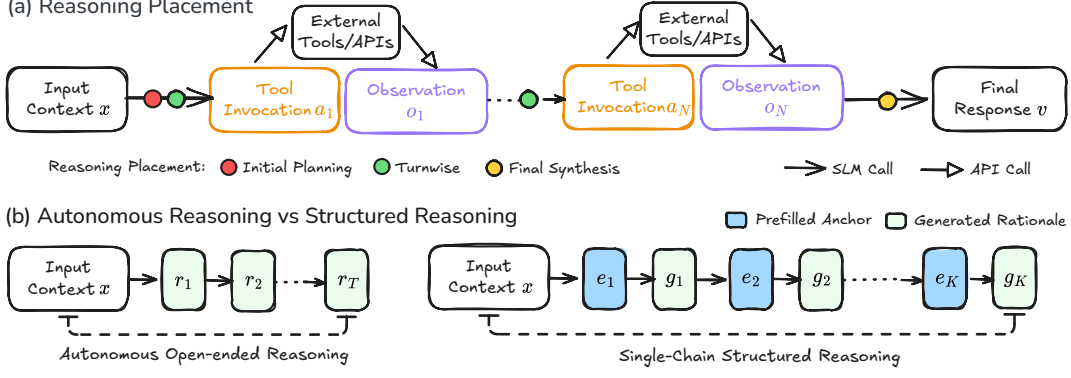


Figure 1: **Reasoning strategies in agentic workflows.** x is input task consisting of the task name and brief one-sentence goal. (a) Candidate placements for applying reasoning within an agentic trajectory. (b) r_t denotes autonomously generated reasoning segments. e_k denotes the injected expert-guided logical anchor (expert anchor), and the g_k represents the generated rationales in response to e_k .

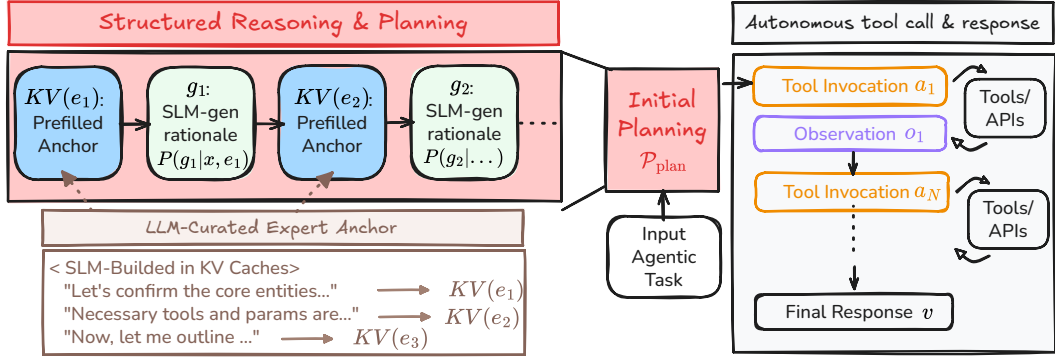


Figure 2: **Overview of FORST-based agentic orchestration.** In Phase 1 (left), predefined structural anchors (e_k) are sequentially injected into the KV-cache. The SLM is constrained to generate intermediate rationales (g_k) between these boundaries, establishing a robust logical foundation for the task (x). Building upon this monolithic planning trajectory, Phase 2 (right) executes the autonomous loop of agent actions (a) and environmental observations (o) until the final synthesis (v) is achieved.

($e_1, g_1, e_2, g_2, \dots, e_K, g_K$). The generation of each rationale segment g_k is conditioned on both the input and the preceding rigid boundaries:

$$P(g_k | \text{context}_k) = P(g_k | x, \{(e_i, g_i)\}_{i=1}^{k-1}, e_k). \quad (2)$$

Consequently, the joint probability of the **FORST**-structured reasoning path is represented as:

$$P(\mathcal{P}_{\text{struct}} | x) = \prod_{k=1}^K P(g_k | x, \{(e_i, g_i)\}_{i=1}^{k-1}, e_k). \quad (3)$$

Essentially, these anchors function as logical checkpoints that are prefilled into the context window. This design guides the reasoning generation toward empirically selected planning structure, allowing the SLM to focus its limited capacity on generating rationales within these structured boundaries.

2.2 Mitigating Stochastic Drift: Vulnerability of Late Interventions

An agentic task is defined by an execution trajectory $\mathcal{T} = (a_1, o_1, a_2, o_2, \dots, a_N, o_N)$, where the model sequentially invokes tools a_j and processes environmental observations o_j . We define the pairing of a_n and o_n as the n -th turn. In the $(N + 1)$ -th turn, the model synthesizes the final response v .

The Primacy of Initial Planning: As depicted in Fig. 1(a), we consider three reasoning strategies: initial planning, turnwise, final synthesis. Then, we compare naive CoT performance of (i) initial planning, (ii) early-half turnwise, (iii) past-half turnwise, (iv) every turnwise, and (v) final response. Table 1 demonstrates that (i) is the most effective stage, with the performance following a consistent hierarchy: (i) > (ii) > (iii) > (v) > (iv). Pointing to (i) as the most critical stage, the fact that (ii)

outperforms (iii) and (v) supports the importance of early stage. This is due to *error compounding*, where initial deviations act as poisoned context that cascades into stochastic drift in SLM. From a structural design perspective, enforcing a unified reasoning template across heterogeneous turns is practically unfeasible. Therefore, anchoring the reasoning process at the onset is the most stable and viable strategy for resource-constrained agents.

The Efficiency Paradox of Autonomous Reasoning: Furthermore, we observed that relying on autonomous test-time scaling (e.g., Qwen3-4B-Think) generates an average of 1.6k reasoning tokens for a single step a_n , yet yields only a marginal 2.1% increase in completion accuracy. Unconstrained "long-thought" frequently entraps SLMs in hallucinatory logical loops. Since complex workflows require multiple turns, open-ended reasoning is fundamentally misaligned with the latency and memory constraints of edge devices.

2.3 The FORST Framework: Logical Scaffolding

Motivated by the inherent vulnerability of SLMs to stochastic drift (Section 2.2), we construct the **FORST** framework as illustrated in Fig. 2.

Monolithic Scaffolding Formalization: By anchoring the reasoning at the initial planning stage, the agentic process is governed by the tight coupling of the structurally forced planning path ($\mathcal{P}_{\text{plan}}$) and the execution trajectory ($\mathcal{T}$). The joint probability is defined as:

$$P(\mathcal{T}, \mathcal{P}_{\text{plan}} \mid x) = \underbrace{\left[\prod_{k=1}^K P(g_k \mid x, \{e_i, g_i\}_{i=1}^{k-1}, e_k) \right]}_{\text{FORST-Scaffolded Foundation}} \times \underbrace{\prod_{j=1}^N P(a_j \mid x, \mathcal{P}_{\text{plan}}, a_{<j}, o_{<j})}_{\text{Autonomous Tool Execution}}. \quad (4)$$

Crucially, $\mathcal{T}$ and $\mathcal{P}$ maintain a *monolithic, single-chain trajectory*. Rather than relying on iterative re-prompting or tree-search paradigms that branch out the trajectory, **FORST** grounds the execution in a single, robust foundation, enabling SLMs to rival frontier models without the structural complexity of auxiliary reasoning modules.

Domain-Agnostic Structural Priors: To address concerns regarding domain overfitting, **FORST** utilizes a universal set of structural anchors rather than domain-specific heuristic rules. By eliciting

the topological planning structures from the frontier models across diverse validation domains, we derived a robust, domain-agnostic structured planning template. As detailed in Fig. 3, the derived expert anchors consist of three stages: entity and constraint analysis, identification of necessary tools and parameters, and concise execution planning, all of which are designed to be strictly domain-agnostic. This ensures that the injected anchors e_k provide a rigid logical skeleton that generalizes across various SLM architectures.

Latency Reduction via KV-Cache Injection: pre-defined anchors e_k are sequentially injected as pre-computed attention states into the KV-cache. As injecting KV states is computationally equivalent to text prompting (please see Sec. C), accuracy gains stem from our logical scaffolding. KV-cache reuse acts as a system-level optimization to bypass prefill overhead. We evaluate effect of this through on-device latency experiment. The overall system is detailed in Algorithm 1.

3 Experimental Results

3.1 Dataset

We evaluate our system using MCPEval (Liu et al., 2025), a comprehensive, Model Context Protocol (MCP)-based framework designed for the automated generation and deep evaluation of LLM agents. The benchmark encompasses diverse real-world domains: Healthcare, Airbnb, Sports, Book, CRM, National Parks, and Finance. By standardizing agent-environment communication through MCP, this dataset allows for the assessment of complex, multiturn agentic workflows that require sophisticated reasoning and precise tool execution.

To ensure high data quality and reliability, we followed their automated generation and verification pipeline by using two frontier LLMs. Specifically, we utilize GPT-4o as the Task LLM to synthesize the name, goal, description of the agentic task based on MCP tool call list in each domain. To validate each task, we employ Gemini2.5 Pro as a frontier agent to execute the generated tasks and collect execution trajectories. Tasks resulting in execution failure undergo an iterative refinement process, where the agent requests updated descriptions until a verifiable ground truth trajectory is established. Notice that, in testing, we only use the name and goal since the description provide too detailed instruction for ground-truth tool calling.

	Method	Venue	Trajectory (%)	Completion (%)
LLM	GPT-4o (OpenAI, 2024)	-	79.65	76.71
	Gemini2.5-Flash (Gemini Team, Google, 2025)	-	73.65	81.39
	Gemini2.5-Pro (Gemini Team, Google, 2025)	-	82.99	84.61
SLM	Naive	-	65.58	62.91
	Plan&Solve (<i>turnwise</i>) (Wang et al., 2023)	ACL 2023	62.14	64.31
	Plan&Solve (<i>planning</i>) (Wang et al., 2023)	ACL 2023	63.67	65.15
	Self-Refine (<i>turnwise</i>) (Madaan et al., 2023)	NeurIPS 2023	57.37	60.14
	Self-Refine (<i>planning</i>) (Madaan et al., 2023)	NeurIPS 2023	61.23	63.24
	Guide&Refine (<i>turnwise</i>) (Chen et al., 2025)	EMNLP 2025	73.31	72.14
	Guide&Refine (<i>planning</i>) (Chen et al., 2025)	EMNLP 2025	80.26	76.31
	InferAct (Fang et al., 2025)	EMNLP 2025	74.55	73.52
	FORST (Ours)		86.30	85.39

Table 2: **Comparative results of structured reasoning baselines and the proposed FORST for different models, including SLM (Qwen3-4B-Inst) and proprietary LLMs.** Average accuracy (%) of trajectory and completion are reported on the MCPEval dataset.

3.2 Evaluation Metrics

We strictly follow the MCPEval evaluation protocol, including the same LLM judge rubric without modification, to ensure comparability with prior work. Here, Gemini2.5-Pro is utilized.

1) *Trajectory Accuracy*: Measures procedural reliability, including the integrity of the planning, execution flow, and adaptive context awareness.

2) *Completion Accuracy*: Measures the quality of the final synthesis, focusing on factual precision, requirement coverage, and practical utility.

This dual-metric approach allows us to identify the *execution-completion gap*, where models may successfully invoke tools but fail to logically synthesize the resulting observations.

3.3 Implementation Details

From MCPEval benchmark, to obtain the planning anchors, we set the airbnb, CRM, and sports as the validation domains, and the remaining domains are used in testing. More details for implementation are in Appendix A.

3.4 Baselines

To show the effectiveness of our method, we compare the proposed **FORST** with four baselines addressing training-free structured reasoning:

-*Plan&Solve* (Wang et al., 2023) simply decouple chain-of-thought into two-stage structure: planning and answering.

-*Self-Refine* (Madaan et al., 2023) alternates between generating multi-faceted, actionable self-feedback and refining previous outputs with few-shot examples across multiple cycles.

-*Guide&Refine* (Chen et al., 2025): In a step during structured reasoning, a frontier model detects

erroneous reasoning cases which is predefined. If detected, a few-shot is given to re-run the step.

-*InferAct* (Fang et al., 2025) employs two different frontier models to preemptively detect and report the misalignment to the original goals before executing every action. If detected, the expert feedback is given based on the report in every turn.

While prior methods such as Plan&Solve, Self-Refine, and Guide&Refine exemplify structured reasoning, they were not designed for long-horizon agentic workflows as considered in MCPEval, where standardized baselines remain limited. To enable a fair comparison, we adapt these methods under a unified setting (same model, tool environment) without modifying the benchmark protocol. Specifically, we construct two variants: (i) *planning*, applied only at the initial stage, and (ii) *turnwise*, applied at every step, allowing us to systematically evaluate reasoning placement across methods. We include both SLM-only and LLM-assisted baselines to cover the strongest available approaches.

3.5 Target SLMs

As SLM for edge-device, we mainly utilize Qwen3-4B-Inst (An Yang, 2023) in experimental analysis. Results in Qwen3-1.7B and 8B are in Appendix.

3.6 Comparative Results

Table 2 presents a comprehensive evaluation of **FORST** against the recent training-free reasoning methods and frontier proprietary models like GPT-4o and Gemini2.5.

Competitive Performance of FORST: Empirical results show that FORST achieves competitive performance compared to frontier models. FORST ob-

tains numerically higher trajectory and completion scores (86.30%, 85.39%) than GPT-4o (79.65% trajectory, 76.71% completion) and Gemini 2.5 Pro (82.99% trajectory, 84.61% completion).

While this does not imply a 4B SLM surpasses frontier models in open-ended reasoning, in strictly orchestrated procedural tasks, FORST’s structural forcing is more robust than the unconstrained generation of GPT-4o or Gemini 2.5 Pro. Results with the same anchors are also compared in Table 3.

Bottlenecks of SLMs in Unconstrained Reasoning: While Plan&Solve improves upon CoT by separating reasoning and answering, its open-ended reasoning fails to generalize to long-horizon agentic tasks, particularly with SLMs. The large score gap serves as evidence for the critical importance of the structured reasoning in initial planning.

Counterproductivity of Iterative Self-Correction in SLMs: Our empirical results reveal a notable performance decline when employing continuous self-correction. Specifically, while Self-Refine (*planning*) reaches 63.24%, its *turnwise* counterpart drops to 60.14%, falling below the naive SLM baseline (62.91%). This trend suggests that explicit, high-frequency intervention from the SLM often proves deleterious in complex agentic workflows. Rather than providing meaningful error correction, the SLM’s persistent verification tends to destabilize the underlying reasoning chain, effectively corrupting the inherent logic instead of refining it.

Diminishing Returns of Iterative LLM-Aided Feedback: Frameworks leveraging external LLM intelligence, such as Guide&Refine (76.31% in *planning*) and InferAct (73.52%), expectedly outperform SLM-centric baselines like Self-Refine. Nevertheless, they still exhibit a substantial performance gap to FORST. They rely on intricate feedback loops and few-shot expert corrections, which exacerbate the context-length bottleneck in SLMs. Due to the significant computational overhead and increased context requirements, the resulting performance gains remain largely incremental.

Notably, Guide&Refine performs better when restricted to the initial planning stage (76.31%) compared to its turnwise execution (72.14%). This reaffirms our hypothesis that constructing a consistent, predefined structure throughout a complex, dynamic trajectory is inherently difficult for SLMs. **Efficacy of FORST’s Structured Forcing:** In contrast, FORST accurately identifies and capitalizes on the fundamental importance of initial planning.

Model	Naive	Sequentially-prefilled ST Reasoning		
		Initial Planning (FORST)	Turnwise	Final Synthesis
GPT-4o	79.65	81.48	79.99	80.34
	76.71	79.43	77.03	78.43
Gemini2.5-Flash	73.65	75.58	73.81	74.22
	80.64	82.41	80.58	81.33
Qwen3-4B-Inst	65.58	86.30	66.29	66.41
	62.91	85.39	68.68	74.53

Table 3: **Importance of the initial planning in structured (ST) reasoning.** For each model, average accuracy (%) of trajectory (top) completion (bottom) (%) is reported, varying reasoning strategies.

By grounding the SLM with refined, expert-guided reasoning anchors at the very onset, we build a solid logical framework that effectively prevents stochastic drift.

3.7 Analysis

Here, we analyze the efficacy of FORST. Our investigation is framed by the central research question:

- To what extent does the use of sequentially-prefilled structured anchoring during the initial planning phase enhance the procedural reliability and task success of SLM agents?

RQ1: Why does initial planning represent the primary bottleneck for SLMs? To examine the effectiveness of structured reasoning across different model scales, we evaluate how our intervention affects both LLMs and SLMs. We omit frontier "thinking" models (e.g., Gemini 2.5 Pro) from this specific analysis, as their internal reasoning processes prevent the integration of externally imposed, predefined structures. Consequently, we adopt Gemini 2.5 Flash and GPT-4o as the LLM baselines for this comparison.

We evaluate three intervention points: *initial planning*, *turnwise*, and *final synthesis*. Regarding the *turnwise* intervention, as reasoning requirements evolve dynamically, we curate three predefined sets of sequential prefills from a validation set. To isolate the impact of the reasoning structure itself, we utilize Gemini 2.5 Pro to select the most appropriate set based on the current trajectory at each turn (Details are in Appendix F). As in Table 3, Gemini 2.5 Flash and GPT-4o exhibit marginal improvement across all intervention stages. This means that while LLMs can internalize instructions at any stage, the improvements remain incremental because these models already possess substantial inherent reasoning capacity.

Metric	Variants of Ours				FORST (Ours)
	SLI	Shuffling	Entity & Plan	Tool & Plan	
Traj. (%)	65.67	72.13	75.42	74.92	86.30
Comp. (%)	67.15	73.47	76.34	75.33	85.39

Table 4: **Ablation of our reasoning structures.** We compare our structured reasoning (e.g. entity&constraint analysis, necessary tools¶ms., final planning) with its variants (shuffling & partially skipped) and system-level instruction called SLI.

Conversely, SLMs demonstrate a pronounced sensitivity to the specific timing of intervention. Qwen3-4B-Inst exhibits a disproportionate performance surge particularly during the initial planning phase. The significant performance discrepancy between *initial planning* and *turnwise* interventions indicates that SLMs struggle to process complex, shifting instructions in the middle of a trajectory. Furthermore, the *turnwise* strategy remains inherently dependent on an LLM for template selection. As described in the Appendix, requiring the SLM to self-select the reasoning set leads to substantial performance degradation. Also, interventions at the *final synthesis* stage prove ineffective for SLMs, as early-stage logical errors typically result in a corrupted trajectory that lacks the evidence necessary for a valid conclusion. These findings confirm that initial planning serves as the most critical leverage point for structured reasoning in SLMs.

RQ2: Effectiveness of Logical Scaffolding and Structural Sequence: To isolate the impact of the prefill and generation mechanism in reasoning, we compare **FORST** against a system-level instruction (SLI), shuffling, and ablated structural variants (partial-skip). In the SLI, a fixed block of instructional text is injected in system prompt. Notice that, unlike our structured reasoning, there is no rationale generation in response to the instruction. Also, we randomly shuffled the logical order in the shuffling, and skipped one of the first two steps in the partial-skip. In Table 4, the gap between SLI (67.15%) and FORST (85.39%) suggests that merely providing global rules is insufficient for SLMs, as the lack of target-specific planning limits their ability to handle the nuances of complex queries. The competitive performance of FORST stems from its post-task, step-wise planning phase, where the model is compelled to synthesize intermediate logic (g_k) between fixed anchors (e_k) before any tool invocation occurs.

Crucially, the shuffling leads to a significant per-

Method	Performance		Avg. Tokens
	Traj. (%)	Comp. (%)	
Baseline (w/o CoT)	65.58	62.91	-
Naive CoT	66.23	67.27	0.5k
Autonomous Think	57.70	64.96	1.3k
FORST (Ours)	86.30	85.39	0.3k

Table 5: **Comparison with autonomous reasoning.** Qwen3-4B-Think and -Inst are used for autonomous think and others, respectively. **FORST** achieves higher procedural accuracy with much lower token overhead.

formance drop (85.39% $\rightarrow$ 73.47%), demonstrating that the effectiveness of **FORST** is not merely due to increased token count but depends on a specific, non-arbitrary logical sequence (Entity&Constraint $\rightarrow$ Tool&Params $\rightarrow$ Final Plan). Also, the partial-skip shows that omitting any stage of the scaffolding weakens the logical foundation, confirming that a complete structural prior is essential to prevent stochastic drift in SLMs.

RQ3: Can Structured Reasoning Surpass Autonomous Long-Thought? We benchmark **FORST** against autonomous reasoning strategies, including naive chain-of-thought in Qwen3-4B-Inst and the long-thought model, Qwen3-4B-Think. Through this comparison, we investigate whether the substantial test-time scaling characteristic of autonomous reasoning can result in enhanced agentic capabilities for SLMs.

Our findings, detailed in Table 5, underscore a distinct efficiency-quality paradox inherent in autonomous SLM reasoning. Although models such as Qwen3-4B-Think produce a substantially higher volume of reasoning tokens (averaging 1.3k), this verbosity fails to translate into a commensurate increase in accuracy. Rather, we identified recurring instances of reasoning circularity where the model succumbs to self-referential logical loops, and hallucinatory drift where the formulated plan diverges from the provided tool specifications.

Conversely, **FORST** attains superior performance while outputting significantly fewer tokens. By substituting open-ended generation with expert-guided sequential prefilling, **FORST** prunes the reasoning space and restricts the model from straying into irrelevant logical trajectories. Additionally, the implementation of KV-cache injection enables the delivery of this high-fidelity reasoning with remarkably low latency. This evidence suggests that for on-device agentic workflows, structured anchoring represents a more robust scaling strategy than

unconstrained test-time computation, offering the necessary logical depth for complex tasks without the excessive overhead of "long-thought" verbosity. **On-device Efficiency:** To evaluate practical utility, we compare **FORST** against Qwen3-4B-Think regarding latency measured on the Snapdragon 8 Elite Gen 5. While the thinking model’s reliance on heavy sequential decoding results in an average initial planning time of 48.05s, **FORST** significantly reduces this overhead. As shown in Table 5, **FORST** generates fewer tokens, and its prefilled expert anchors bypass the high costs of autoregressive decoding. Consequently, **FORST** achieves an initial planning latency of 5.48s, which is further decreased by utilizing pre-computed KV caches. Our proposed anchoring thus ensures responsive and deterministic performance, making it highly viable for edge-device deployment.

4 Related Works

4.1 LLM as Planner

Recent advancements utilize LLMs for task decomposition and sequential planning (LLM-Planner (Song et al., 2023), KnowAgent (Zhu et al., 2025)), with some incorporating iterative self-training to align with human intent (LLM-Personalize (Han et al., 2025)). However, these models’ reliance on high-level reasoning often leads to hallucinations or logical inconsistencies when scaled down to SLMs. While topological approaches (Wu et al., 2024; Chen et al., 2024b) like tree search or graph-based modules enhance robustness, they incur a large computational overhead and deviate from efficient single-chain inference. Even collaborative frameworks like COPE (Lee et al., 2026) still require LLM intervention, highlighting a lack of standalone robustness in smaller models.

FORST enables robust single-chain planning without the prohibitive overhead of topological search or the structural complexity of auxiliary modules.

4.2 Structured Reasoning

Current research in structured reasoning can be divided into learning-based optimization and inference-time prompting strategies. Learning-based methods integrate structural priors directly into the model’s parameters: SAND (Xia et al., 2025) finetunes agents through execution-guided critique of action deliberations, SEER (Chen et al., 2024a) optimizes hierarchical dependencies via reinforcement learning with structure-based returns, and Yubo Dong (2026) utilizes a graph-

based framework with Max-Flow rewards to improve reasoning efficiency. While these methods achieve high fidelity, they entail significant training complexity. Conversely, non-learning approaches utilize the inherent capabilities of LLMs without parameter updates: RCI (Kim et al., 2023) and Self-Refine (Madaan et al., 2023) employ recursive feedback loops for iterative improvement, while InferAct (Fang et al., 2025) leverages multiple additional models to preemptively detect misaligned actions. To stabilize multi-step inference, Guide&Refine (Chen et al., 2025) extracts structured guidelines from successful trajectories, and IRCOT (Trivedi et al., 2023) interleaves retrieval with reasoning steps to maintain factual consistency.

In spite of their efficacy in LLMs, they have not considered the SLMs’ properties. Learning-based frameworks require prohibitive computational resources for reinforcement learning, while prompting-based refinement introduces excessive latency and relies on the correction capabilities of frontier LLMs, which are notably deficient in smaller-scale models.

5 Conclusions

In this work, we introduced **FORST**, a framework designed to unlock the reliability of SLM-based agents through the primacy of initial planning. Our investigation reveals that the failure of SLMs in complex orchestration is not due to a lack of tool-use capability, but rather the absence of a robust logical foundation at the onset of a task. By employing structured logical forcing via expert-guided sequential prefilling, **FORST** ensures reasoning consistency and prevents the stochastic drift typical of autonomous SLMs. Furthermore, our approach can maximize on-device efficiency by leveraging system-level KV-cache re-use, achieving performance levels comparable to frontier models like GPT-4o without the overhead of massive test-time scaling. These results demonstrate that strategic, structured intervention at the planning stage is a more effective and scalable path for deploying sophisticated agentic systems on edge devices than increasing model parameter count or purely autonomous reasoning.

6 Limitations

While **FORST** ensures high reliability, we identify three strategic areas for extension. First, while our domain-agnostic anchors enable broad deployment

across diverse tasks, highly specialized domains may require further structural refinement to fully capture unique, niche-specific logic. Second, our focus on single-agent orchestration is a strategic design necessitated by current edge hardware constraints. As on-device performance scales, adapting this framework to multi-agent protocols will be essential for managing higher-order complexity. Finally, the one-time prefilling cost is a negligible trade-off for the 8.7x latency reduction, with ongoing system-level optimizations expected to further minimize this initial overhead.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *ArXiv:2303.08774*.
- Baosong Yang Beichen Zhang Binyuan Hui Bo Zheng Bowen Yu Chang Gao Chengen Huang Chenxu Lv Chujie Zheng Dayiheng Liu Fan Zhou Fei Huang et al. An Yang, Anfeng Li. 2023. Qwen3 technical report. *Arxiv:2505.09388*.
- Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. <https://arxiv.org/abs/2506.02153>.
- Guoxin Chen, Kexin Tang, Chao Yang, Fuying Ye, Yu Qiao, and Yiming Qian. 2024a. SEER: Facilitating structured reasoning and explanation via reinforcement learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.
- Jiaxiang Chen, Zhuo Wang, Mingxi Zou, Zhucong Li, Zhijian Zhou, Song Wang, and Zenglin Xu. 2025. From implicit exploration to structured reasoning: Leveraging guideline and refinement for llms. In *Findings of the 2025 Conference on Empirical Methods in Natural Language Processing*.
- Ziru Chen, Michael White, Ray Mooney, Ali Payani, Yu Su, and Huan Sun. 2024b. When is tree search useful for LLM planning? it depends on the discriminator. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.
- Claude Code. 2026. Claude code. <https://claude.com/product/claude-code>.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *ArXiv:2407.21783*.
- Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. 2025. Preemptive detection and correction of misaligned actions in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.
- Gemini Team, Google. 2025. Gemini 2.5 pro: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. <https://arxiv.org/abs/2507.06261>. Technical Report.
- Dongge Han, Trevor McInroe, Adam Jelley, Stefano V. Albrecht, Peter Bell, and Amos Storkey. 2025. LLM-personalize: Aligning LLM planners with human preferences via reinforced self-training for house-keeping robots. In *Proceedings of the 31st International Conference on Computational Linguistics*.
- HuggingFace. 2026. Huggingface. <https://huggingface.co/>.
- Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.
- Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. In *Conference on Neural Information Processing Systems*.
- Byeongchan Lee, Jonghoon Lee, Dongyoung Kim, Jaehyung Kim, Kyungjoon Park, Dongjun Lee, and Jinwoo Shin. 2026. Efficient llm collaboration via planning. <https://arxiv.org/abs/2506.11578>.
- Zhiwei Liu, Jielin Qiu, Shiyu Wang, Jianguo Zhang, Zuxin Liu, Roshan Ram, Haolin Chen, Weiran Yao, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. 2025. MCP Eval: Automatic MCP-based deep evaluation for AI agent models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Conference on Neural Information Processing Systems*.
- Manus AI. 2026. Manus ai. <https://manus.im/>.
- OpenAI. 2024. Gpt-4o. <https://openai.com/index/hello-gpt-4o/>. Accessed: 2024-05-13.
- OpenClaw. 2026. Openclaw. <https://openclaw.ai/>.

- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling llm test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representation*.
- Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *International Conference on Computer Vision*.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.
- Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. <https://arxiv.org/abs/2201.11903>.
- Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and Dongsheng Li. 2024. Can graph learning improve planning in llm-based agents? In *Conference on Neural Information Processing Systems*.
- Yu Xia, Yiran Shen, Junda Wu, Tong Yu, Sungchul Kim, Ryan A. Rossi, Lina Yao, and Julian McAuley. 2025. Sand: Boosting llm agents with self-taught action deliberation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.
- Wenhao Yu, Hongming Zhang, Xiaoman Pan, Peixin Cao, Kaixin Ma, Jian Li, Hongwei Wang, and Dong Yu. 2024. Chain-of-note: Enhancing robustness in retrieval-augmented language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.
- Linchao Zhu Yi Yang Yubo Dong, Hehe Fan. 2026. Structured reasoning for llms: A unified framework for efficiency and explainability. In *International Conference on Learning Representation*.
- Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Hua-jun Chen, and Ningyu Zhang. 2025. KnowAgent: Knowledge-augmented planning for LLM-based agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*.

- Appendix -

Foundation First: Stabilizing SLM Agents via Pre-execution Logical Scaffolding

A Implementation Details

All methods were implemented using the PyTorch framework and the Hugging Face (HuggingFace, 2026) Transformers library. Experiments were conducted on NVIDIA A100 (80GB) GPUs. For distributed processing and memory optimization, we utilized the accelerate library where applicable. To ensure the reproducibility of our results and minimize stochastic variance, we employed greedy decoding by setting the temperature to $T = 0$ for all generation tasks. Other decoding parameters (e.g., top_p, top_k) were disabled to maintain deterministic outputs, allowing for a rigorous comparison of model performance across different benchmarks.

Dataset Size: For the benchmark, we generated 100 tasks per topic to ensure a balanced and statistically significant distribution across diverse domains. The specific prompt templates and task-generation pipelines strictly adhere to the standards described in (Liu et al., 2025)

Implementation of Structured Reasoning: We evaluated two distinct prompting strategies: providing explicit instructions for reasoning within the user-side prompt versus prefilling the assistant’s response with an initial reasoning header (e.g., a “thought” block). Experimentally, the former approach proved more effective. This is likely because modern instruction-tuned models are primarily optimized to satisfy constraints and stylistic requirements defined within the user context; by contrast, forcing a specific starting sequence in the assistant’s turn can occasionally lead to a distribution shift that disrupts the model’s natural generation flow or weakens its adherence to the requested logical structure. We utilized user-side prompt templates as the textual source to generate the reference KV-cache states. Once converted into attention blocks, these function as fixed cached prefix states that are reused during inference, bypassing the distribution shift issues often found in assistant-side prefills while maintaining absolute structural integrity.

Clarification on KV-Cache Source: The ‘user-directed instructions’ mentioned above refer to the textual format used to generate the reference KV-cache states. We found that using user-side

prompts as the source for pre-computation (rather than assistant-side headers) better aligns with the SLM’s instruction-tuning distribution. Once converted into KV states, the anchor templates are reused as fixed prefix states during inference. This does not introduce a separate constrained-decoding mechanism; rather, it implements the same structured anchor context more efficiently by avoiding repeated prefill computation. The implementation utilized [vLLM/HuggingFace Accelerate] to perform direct memory-pointer manipulation for cache injection on the Snapdragon 8 Elite platform."

B Results on More SLMs

To evaluate the generalizability of both our proposed framework and the structured reasoning baselines, we extended our experiments to a broader range of models, specifically Qwen3-1.7B and Qwen3-8B. As illustrated in Table 6, models without instruction tuning consistently exhibit inferior performance in agentic tasks compared to Qwen3-4B-Inst (86.30% in trajectory, 85.39% in completion). Notably, even the larger Qwen3-8B variant fails to surpass the 4B instructed model when alignment is absent—a trend that persists across all evaluated baselines. This performance gap highlights the inherent difficulty non-tuned models face in processing complex reasoning instructions. In this challenging setting, baseline methods suffer from severe performance degradation, whereas our proposed method, **FORST**, demonstrates superior robustness by leveraging a streamlined single reasoning chain. Furthermore, across all models, the comparative analysis of reasoning strategies (naive, Autonomous thinking, turnwise, and final synthesis) shows similar trends observed with Qwen3-4B-Inst. These consistent empirical results support our central contribution: the critical importance of expert intervention during the initial planning stage for successful task execution.

C Computational Equivalence of KV-Cache Injection and Text Prompting

We clarify that injecting pre-computed KV-cache states for the structured anchors e_k is computa-

Method	Traj. (%)	Comp. (%)
Naive (Qwen3-1.7B)	34.94	26.66
Autonomous Think (thinking mode)	22.45	13.18
Self-Refine (<i>turnwise</i>) (Madaan et al., 2023)	19.23	13.51
Self-Refine (<i>planning</i>) (Madaan et al., 2023)	20.12	17.22
Guide&Refine (<i>turnwise</i>) (Chen et al., 2025)	28.82	22.21
Guide&Refine (<i>planning</i>) (Chen et al., 2025)	27.35	24.32
InferAct (Fang et al., 2025)	31.42	23.89
turnwise	24.47	20.58
final synthesis	32.64	15.84
FORST (Ours)	42.14	29.31

Method	Traj. (%)	Comp. (%)
Naive (Qwen3-8B)	52.60	60.62
Autonomous Think (thinking mode)	58.48	63.21
Self-Refine (<i>turnwise</i>) (Madaan et al., 2023)	50.71	58.34
Self-Refine (<i>planning</i>) (Madaan et al., 2023)	52.04	61.22
Guide&Refine (<i>turnwise</i>) (Chen et al., 2025)	58.08	71.22
Guide&Refine (<i>planning</i>) (Chen et al., 2025)	62.02	73.14
InferAct (Fang et al., 2025)	59.43	70.49
turnwise	47.45	68.32
final synthesis	50.29	64.13
FORST (Ours)	62.31	78.86

Table 6: **Comparative results of structured reasoning baselines and variants of FORST, the proposed FORST for two SLM models (Qwen3-1.7B (top) and Qwen3-8B (bottom)).** Average accuracy (%) of trajectory and completion are reported on the MCP Eval dataset.

tionally equivalent to providing the same anchors as standard text context. This equivalence holds due to the causal masking property and the deterministic positional anchoring in autoregressive Transformers utilizing Rotary Position Embedding (RoPE) (e.g., Qwen). Let the token sequence of a pre-defined structural anchor be $X_{\text{anchor}} = [x_1, x_2, \dots, x_m]$ and the dynamically generated rationale be $X_{\text{dynamic}} = [x_{m+1}, \dots, x_T]$.

Because of the strict lower-triangular causal mask in decoder-only SLMs, the representations for any token in the anchor depend exclusively on the preceding tokens. Crucially, the conditional independence holds at the pre-RoPE representation level. The pre-RoPE Keys and Values for the anchor can be computed in strict isolation:

$$K_{\leq m}^{\text{pre}}, V_{\leq m} = \text{Forward_pre_RoPE}(X_{\text{anchor}})$$

In RoPE, positional information is injected by rotating the queries and keys based on their absolute positions. Because the structural anchor in FORST is permanently fixed at the absolute beginning of the context window (positions 0 to $m - 1$), the rotational transformation applied to $K_{\leq m}^{\text{pre}}$ is entirely deterministic and invariant to the content or length of the subsequent X_{dynamic} . Therefore, when FORST injects these states and processes

Algorithm 1: FORST: Foundation **f**IRST via **D**eterministic Logical Scaffolding

Input : User query x , Predefined structural anchors $\mathcal{E} = \{e_1, \dots, e_K\}$, SLM θ

Output : Final synthesized response v , Execution trajectory $\mathcal{T}$

// Phase 1: Establishing Logical Foundation

```

1 Initialize planning path  $\mathcal{P}_{\text{plan}} \leftarrow \emptyset$ , KV Cache  $\mathcal{C}$  with context  $x$ 
2 for  $k \leftarrow 1$  to  $K$  do
3   Inject predefined anchor  $e_k$  into KV Cache  $\mathcal{C}$ 
4   Force generation of rationale
      $g_k \sim P(g_k \mid x, \{(e_i, g_i)\}_{i=1}^{k-1}, e_k)$  using  $\theta$ 
5   Update planning path
      $\mathcal{P}_{\text{plan}} \leftarrow \mathcal{P}_{\text{plan}} \cup \{e_k, g_k\}$ 
6   Update  $\mathcal{C}$  with attention states of  $g_k$ 

```

// Phase 2: Autonomous Execution Loop

```

7 Initialize trajectory  $\mathcal{T} \leftarrow \emptyset$ 
8 for  $j \leftarrow 1$  to  $N$  do
9   Generate tool action
      $a_j \sim P(a_j \mid x, \mathcal{P}_{\text{plan}}, a_{<j}, o_{<j})$ 
10  if  $a_j$  is Final Answer then
11     $v \leftarrow a_j$ 
12    break
13  Execute  $a_j$  and receive environmental observation  $o_j$ 
14  Update trajectory  $\mathcal{T} \leftarrow \mathcal{T} \cup \{a_j, o_j\}$ 
15 return  $v, \mathcal{T}$ 

```

X_{dynamic} , the relative distance calculations in the self-attention dot product:

$$\text{Attn}(Q_t, K_{\leq t}, V_{\leq t}) = \text{softmax} \left(\frac{Q_t K_{\leq t}^T}{\sqrt{d_k}} \right) V_{\leq t}$$

Aside from negligible hardware-level floating-point variations (e.g., non-deterministic atomic additions in FlashAttention), the resulting probability distribution $P(g_k \mid x, e_k)$ remains identical to that of standard full-sequence prefilling. Thus, KV-cache reuse acts as a latency-reduction optimization without altering the reasoning trajectory.

D Anchor Source for FORST

Prompt template for FORST
You are currently in Phase 1: Initial Planning . Do NOT call any tools yet. Your goal is to analyze the request and create a solid plan through 3 Steps. Follow the structured steps provided in the prompt.\n\n In this turn, ONLY answer the Step1. Do Not proceed to the Steps2 and 3 in this turn. Step 1: Entity & Constraint Analysis \n Identify the core entities and specific constraints from the goal and title of the given task.\n\n- Core Entities (Who/What/Where):
In this turn, ONLY answer the step2. Do Not proceed to the next Steps in this turn. Namely, Only identify the necessary tools and pre-calculate any specific parameters (Dates, IDs) required for the first steps. If logic (like adding days) is needed, show the calculation.
Now Step3. Based on this reasoning, ONLY outline the concise execution steps.

Figure 3: Three-step prompt templates used as the source for pre-computing static KV cache prefixes in **FORST**. Unlike standard prompting, these templates are converted into reusable KV cache attention states and injected into the SLM’s context window at runtime, preventing the model from altering the structural anchors during the reasoning process.

E Prompt Template for System-level Instructions Variant

Prompt template for static guidance
Before generating the next tool call or final response, you MUST verify the following: 1. Goal Alignment : Does my next action directly contribute to the user’s original goal? 2. Contextual Consistency : Am I using the correct identifiers (IDs, keys, names) from the PREVIOUS steps? Do not confuse information from different entities. 3. Tool Schema Adherence : If calling a tool, does the tool name and all its parameters EXACTLY match the provided schema?
Are all required parameters present and correctly formatted? 4. Feasibility Check : Is the task still possible with the available information? If a previous tool call returned empty or insufficient data, I must report this limitation instead of guessing or calling another tool with invalid inputs.

Figure 4: Prompt template for the system-level instruction in Table 4.

F Prompt Template for *turnwise* Variant in FORST

Type1: Prompt template for plan

To tackle this task, first clarify the user's ultimate goal in one sentence.

Good. Now, list the specific pieces of information you need to gather to achieve this goal, less than 300 tokens.

Based on that, what is the immediate first sub-task you need to perform? Describe it briefly.

Which tool is best suited for this sub-task and why? Describe it briefly.

Type2: Prompt template for checking tool argument

Which tool have you decided to call? Describe it briefly.

Check the tool definition. What are the specific required parameters for this tool? Describe it briefly.

Based on the conversation history and context, determine the exact values for these arguments. Describe it briefly.

Type3: Prompt template for observation success

Analyze the tool output. What key data or facts have you successfully gathered? Describe it briefly.

Verify this data against the user's request. Does it confirm any specific facts? Describe it briefly.

With this information in hand, what is the next logical step to move forward? Describe it briefly.

Type4: Prompt template for observation error

The tool execution failed or returned no results. Analyze the potential root cause (e.g., typo, invalid ID, filter too strict), briefly.

Do not give up. Propose a concrete alternative strategy or fallback plan to get the information, less than 300 tokens.

Specify the new search query or parameter adjustment you will use for this attempt.

Type5: Prompt template for repeated same observation error

This tool execution failed or returned no results, repeatedly. Check the potential root cause is tool itself and consider using other tools. What is the next logical step to move forward?

Type6: Prompt template for filtering
Review the retrieved list. What specific condition or criteria must the selected item meet according to the user? Describe it briefly.
Iterate through the candidate items. Analyze the first few items to see if they match the criteria, briefly.
Identify the final selected item that best matches the criteria.
Type7: Prompt template for final response synthesis
The task is nearly complete. Summarize the key findings you have gathered so far, point by point. Do this less than 500 tokens.
Synthesize these findings into a final conclusion. Do this less than 500 tokens.
Draft the final response to the user. What key details or caveats must be included? Do this less than 500 tokens.
Prompt template for type selection via LLM
<pre> # Context - **User Goal**: {task.goal} - **Available Tools**: {tools_list_str} # Conversation History (Last few turns) {history_str} # Candidate Templates 1. **TEMPLATE_PLAN**: Use when starting a new task or replanning. (User input -> Plan) 2. **TEMPLATE_TOOL_ARGS**: Use when a specific tool needs to be called, to reason about parameters. 3. **TEMPLATE_OBSERVATION_SUCCESS**: Use when a tool returned valid data. Analyze and move to next step. 4. **TEMPLATE_OBSERVATION_ERROR**: Use when a tool returned an error or empty result. Plan a fallback. 5. **TEMPLATE_OBSERVATION_REPEATED_SAME_ERROR**: Use when a tool returned repeated (more than 2 times) error or empty result. Plan to use another tool. 6. **TEMPLATE_FILTERING**: Use when a list of items was returned and needs filtering by condition. 7. **TEMPLATE_SYNTHESIS**: Use when all info is gathered and a final answer is needed. # Decision Logic - If last message is **User** -> **TEMPLATE_PLAN**. - If last message is **Tool Output**: - Error/Empty -> **TEMPLATE_OBSERVATION_ERROR** - Repeated Error/Empty in a specific tool: **TEMPLATE_OBSERVATION_REPEATED_SAME_ERROR** - List needing selection -> **TEMPLATE_FILTERING** - Good data -> **TEMPLATE_OBSERVATION_SUCCESS** or **TEMPLATE_SYNTHESIS** (if done). - If Agent intended to use a tool but hasn't called it -> **TEMPLATE_TOOL_ARGS**. # Output Format Return ONLY the JSON object: {"selected_template_id": "TEMPLATE_NAME"} </pre>

Figure 5: Prompt template for *turnwise* in Table 3.