Small Language Models for Autonomous AI Agents: A Systematic Review of Efficient Reasoning, Tool Use, Planning and Reliability

Kishore[1], Shenbagavadivu[2], Mathiyarasi[3], Preethi[4], Vaishnavi Devi[5], and Veeraselvi[6]
[1,2,3,4,5,6]*Department of Information Technology, SRM Valliammai Engineering College, Tamil Nadu, India*

***Abstract*—Autonomous AI agents built on large language models (LLMs) have demonstrated strong reasoning, planning and tool-use capabilities, but their computational cost, latency, energy consumption and centralized deployment increasingly conflict with the operational requirements of real-world agentic systems. Small language models (SLMs) have re-emerged as a candidate substrate for such systems, offering order-of-magnitude reductions in inference cost while retaining much of the narrow, task-scoped competence that agentic sub-tasks actually require. This paper presents a systematic review of the literature on SLMs for autonomous agents, critically synthesizing work across four capability axes—reasoning, planning, tool use and memory-augmented retrieval—alongside the efficiency, latency, energy and privacy considerations that motivate their adoption. Existing surveys treat SLM capability, agent architecture, model routing and verification as largely separate literatures; this review's principal contribution is a Reliability-Focused Taxonomy that unifies them by explicitly linking SLM Capability to Agent Role, to a characteristic Failure Mode, to an applicable Verification mechanism, and to an Adaptive Escalation decision. Under the resulting framework, easy, low-risk sub-tasks are executed directly by an SLM; uncertain sub-tasks are executed with an accompanying verification pass; and complex or high-risk sub-tasks are escalated to a larger model, with the escalation decision and its outcome logged for continual recalibration. We compare this framework against existing cascade, routing and self-verification approaches, identify concrete research gaps in cross-task transferability of verifiers, standardized escalation benchmarks and lifecycle energy accounting, and outline directions for future work.**

***Index Terms*—adaptive escalation, agentic AI, edge inference, model cascading, reliability, small language models, tool use, verification.**

I. INTRODUCTION

The deployment of autonomous AI agents—systems that decompose a goal into sub-tasks, invoke external tools, and act with limited human supervision—has accelerated sharply since 2023, driven almost entirely by large language models (LLMs) acting as the agent's reasoning core [1], [2]. Frameworks surveyed by Masterman et al. [3] and Luo et al. [4] show a consistent architectural pattern: a generalist LLM interprets instructions, plans a sequence of actions, calls tools or application programming interfaces (APIs), and synthesizes results into a final response. This pattern has proved effective, but it silently assumes that every invocation in an agentic pipeline warrants the full generality of a frontier-scale model.

That assumption is increasingly being questioned. Belcak et al. [5], writing from NVIDIA Research, argue explicitly that small language models (SLMs)—broadly, models that fit on a single consumer-grade accelerator and respond with latency suitable for interactive use—are "sufficiently powerful, inherently more suitable, and necessarily more economical" for the majority of invocations inside an agentic system. Their position rests on an empirical observation echoed across several independent surveys [6]–[9]: agentic sub-tasks are typically narrow, repetitive and format-constrained (parsing an intent, selecting a tool, filling a function-call schema), and modern SLMs in the one-to-ten-billion-parameter range have closed much of the capability gap with earlier generations of far larger models on exactly this kind of task.

At the same time, the case for SLMs is not unconditional. Complex multi-hop reasoning, open-

domain dialogue and tasks requiring broad world knowledge still favor larger models, and naively replacing every LLM call in an agent with an SLM risks silent capability shortfalls that are difficult to detect without explicit verification. The practical question facing agent designers is therefore not "SLM or LLM" but "which model, verified how, and escalated when." Existing literature addresses fragments of this question—SLM capability benchmarking [6]–[9], model cascading and routing [28]–[31], and self-verification of LLM outputs [32], [34]—largely as separate research threads with limited cross-referencing.

This paper makes three contributions. First, it synthesizes the fragmented literature on SLM-based agentic reasoning, planning, tool use and memory into a single critical review, organized around the operational constraints (latency, computation, energy, privacy) that determine whether an SLM is deployable in a given agentic role. Second, it proposes a Reliability-Focused Taxonomy that connects SLM Capability, Agent Role, Failure Mode, Verification and Adaptive Escalation into a single decision framework, and positions this framework relative to existing cascading and routing approaches. Third, it identifies concrete, evidence-grounded research gaps—rather than generic calls for "more research"—in cross-domain verifier transfer, escalation benchmarking and full-lifecycle efficiency accounting.

The remainder of the paper is organized as follows. Section II describes the systematic review methodology and research questions. Section III surveys the SLM landscape and contrasts it with LLMs. Section IV reviews agentic reasoning, planning, tool use and memory in SLMs. Section V examines efficiency, latency, energy and privacy considerations. Section VI presents the proposed reliability taxonomy and adaptive escalation framework. Section VII analyzes failure modes and verification strategies. Section VIII compares the proposed framework with existing approaches. Section IX discusses research gaps and future directions, and Section X concludes.

II. SYSTEMATIC REVIEW METHODOLOGY

A. Research Questions

The review is structured around five research questions (RQs):

RQ1: What agentic capabilities—reasoning, planning, tool use and memory—have been demonstrated by SLMs, and how do they compare with LLM baselines on the same capability?

RQ2: What operational factors (latency, computational cost, energy consumption, privacy) distinguish SLM from LLM deployment in agentic settings?

RQ3: What failure modes are characteristic of SLM-driven agentic execution, and which verification mechanisms have been proposed to detect them?

RQ4: How do existing model-cascading, routing and self-verification approaches decide when to escalate from a small to a large model, and what are their limitations for agentic (as opposed to single-turn query-answering) settings?

RQ5: What unifying framework can connect SLM capability, agent role, failure mode, verification and escalation into a single, reusable reliability model for agent designers?

B. Search Strategy and Sources

Candidate literature was identified through structured queries against arXiv, IEEE Xplore, the ACM Digital Library and Google Scholar, using combinations of the terms "small language model," "SLM," "agent," "tool use," "function calling," "model cascade," "LLM routing," "self-verification" and "escalation," restricted primarily to work published between 2023 and 2026 to capture the post-ChatGPT agentic-AI literature while allowing inclusion of earlier foundational work (e.g., LoRA [23], chain-of-thought prompting [36]) that the field continues to build on. Reference lists of retrieved surveys [1]–[9] were backward-chained to surface additional primary studies, and forward citation search was used to identify more recent work citing key cascading and verification papers [28], [32], [33].

Fig. 2 summarizes the resulting identification-to-inclusion workflow. Records were first de-duplicated, then screened at the title/abstract level for relevance to at least one of the five capability or reliability axes above; full texts of the remaining records were assessed for methodological transparency (reported

benchmarks, ablations or architectural detail) before inclusion in the synthesis.

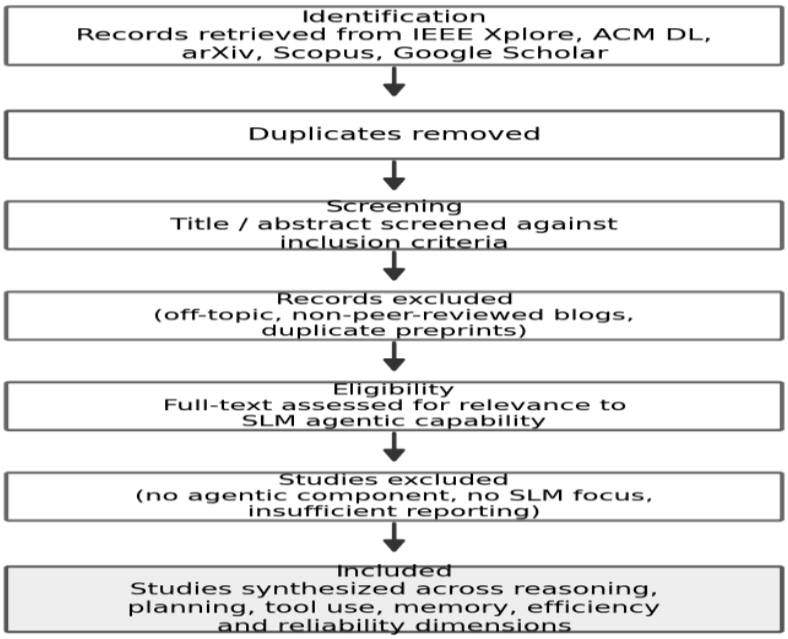

Fig. 2. Literature identification and selection workflow.

C. Inclusion and Exclusion Criteria

Studies were included if they (a) evaluated a language model with fewer than approximately ten billion parameters, or explicitly framed a technique as applicable to such models; (b) addressed at least one agentic capability (reasoning, planning, tool use, memory/retrieval) or a reliability mechanism (verification, cascading, routing, escalation) relevant to agent deployment; and (c) reported empirical results, an architectural contribution, or a position grounded in cited prior evidence. Studies were excluded if they addressed only pre-training or fine-tuning of SLMs with no agentic or tool-use framing, consisted solely of marketing material without technical content, or duplicated results already reported in an included survey.

D. Synthesis Approach

Rather than proceeding paper-by-paper, the synthesis in Sections IV–VIII is organized thematically: sources are grouped by the capability or mechanism they address, contrasted against each other on their assumptions and reported trade-offs, and used collectively to motivate and stress-test the reliability taxonomy proposed in Section VI. Where multiple studies report divergent efficiency figures for ostensibly similar comparisons, this is noted explicitly rather than reconciled by unweighted averaging, since inference cost is highly sensitive to hardware, batching and quantization choices that differ across studies [5], [8], [42], [43].

III. THE SLM LANDSCAPE AND ITS RELATION TO LLMS

A. Defining the Boundary

No single parameter threshold cleanly separates "small" from "large" language models, and several surveys deliberately avoid a fixed cut-off [5], [6]. Belcak et al. [5] instead define an SLM operationally as a model that can be executed on a single common consumer-grade accelerator with latency acceptable for one user's interactive agentic requests, and note that as of 2025 most models under roughly ten billion parameters satisfy this definition. Nguyen et al. [9] and Lu et al. [6] adopt a similarly functional framing, emphasizing deployability on resource-constrained hardware over a specific parameter count. This review

follows the same convention: "SLM" denotes models designed and evaluated for efficient, often on-device or single-accelerator, inference, while "LLM" denotes models whose intended deployment presumes multi-accelerator, data-center-scale serving.

B. SLM Families Relevant to Agentic Use

A cluster of recent model families is repeatedly cited in the agentic-SLM literature for combining competitive benchmark performance with sub-ten-billion-parameter footprints. Microsoft's Phi series demonstrates that careful data curation can substitute for scale on commonsense-reasoning and code-generation benchmarks: Phi-2 (2.7B) matches thirty-billion-parameter models on these axes while running roughly fifteen times faster [10], [11], and Phi-3-small (7B) reaches language-understanding and code-generation scores comparable to substantially larger models of the same generation [10]. NVIDIA's Nemotron-H family combines Mamba and Transformer blocks in 2, 4.8 and 9-billion-parameter variants to match thirty-billion-parameter dense models on instruction following and code generation at a fraction of the inference floating-point operations [13]. Hugging Face's SmolLM2 family, spanning 125 million to 1.7 billion parameters, is reported to reach the tool-calling and instruction-following performance of fourteen-billion-parameter contemporaries [12]. NVIDIA's Hymba-1.5B, a hybrid attention-Mamba architecture, is reported to outperform some thirteen-billion-parameter models on instruction accuracy while achieving substantially higher token throughput [14]. DeepSeek-R1-Distill models (1.5–8B), distilled from the DeepSeek-R1 reasoning model [15], are reported to exceed some larger proprietary models on reasoning-heavy tasks despite their reduced size. Salesforce's xLAM-2-8B, purpose-built for tool invocation, is reported to surpass several frontier general-purpose models on function-calling benchmarks despite its comparatively modest parameter count [16].

These figures come from the original authors' own benchmarking protocols and are not independently re-measured in this review; they are reported here, and in Table I, strictly as documented in the cited primary sources, and should be read as indicative of a broader trend—narrowing task-specific capability gaps—rather than as universally reproducible multipliers.

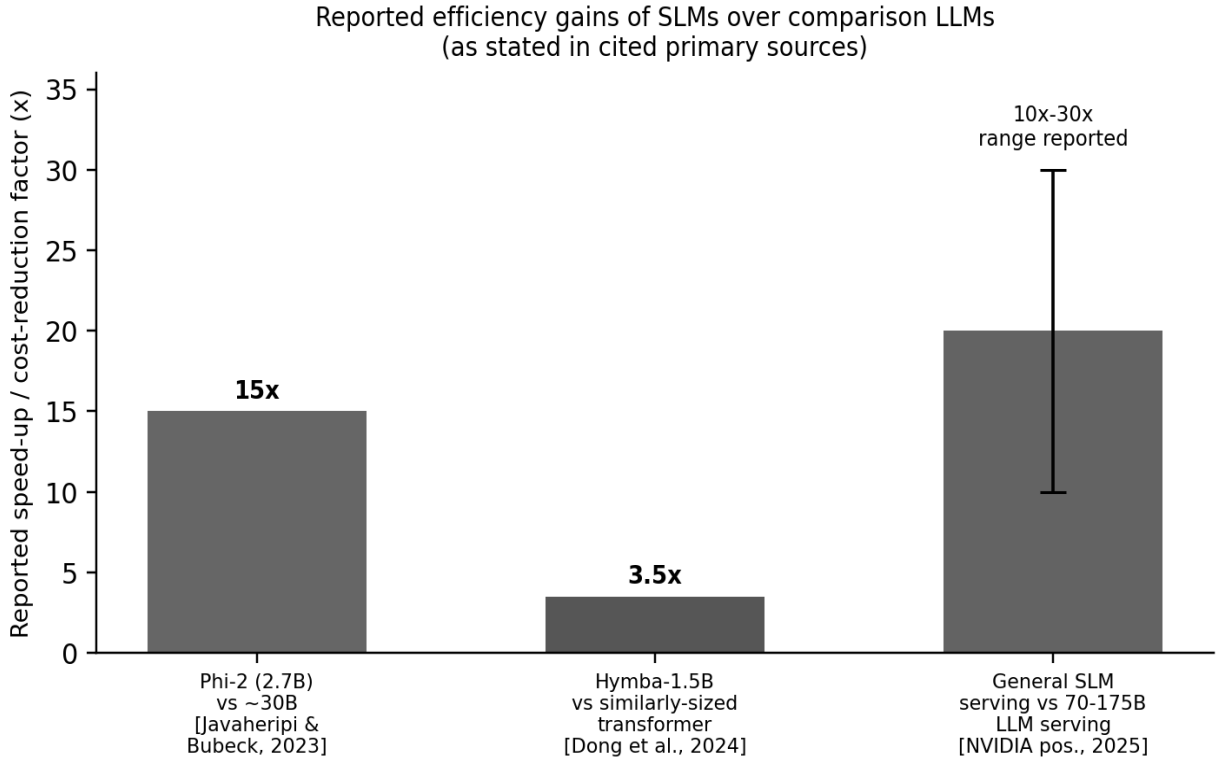

Fig. 5. Efficiency multiples are as reported by the cited original studies and are not re-measured in this review; figures are illustrative of literature-reported trends only.

TABLE I. Representative SLM Families Cited in Agentic-Capability Literature

Model family	Representative size	Reported agentic-relevant result	Source
Phi-2 / Phi-3-small	2.7B / 7B	Commonsense reasoning and code generation on par with ~30B models; Phi-2 ~15x faster	[10], [11]
NVIDIA Nemotron-H	2 / 4.8 / 9B	Instruction-following and code-generation accuracy comparable to dense 30B models at an order-of-magnitude fewer FLOPs	[13]
SmolLM2	125M–1.7B	Tool-calling and instruction-following comparable to 14B contemporaries	[12]
Hymba-1.5B	1.5B	Outperforms some 13B models on instruction accuracy; higher token throughput than similarly sized transformers	[14]
DeepSeek-R1-Distill	1.5–8B	Strong commonsense/reasoning performance relative to size; distilled from DeepSeek-R1	[15]
xLAM-2	8B	State-of-the-art tool-calling performance relative to size on function-calling benchmarks	[16]

C. SLM vs. LLM: A Structural Comparison

Table II summarizes the qualitative trade-offs reported across the surveyed literature. The comparison is deliberately structural rather than numeric in most rows, because absolute performance gaps are benchmark- and task-dependent and shift with each model release; the direction and rationale of each trade-off, however, are stable across sources [5]–[9].

TABLE II. Structural Comparison of SLMs and LLMs in Agentic Deployment

Dimension	Small language models (SLMs)	Large language models (LLMs)
Typical deployment	Single accelerator or on-device; edge/mobile feasible [5], [12]	Multi-accelerator, data-center-scale serving [5]
Reasoning breadth	Strong on narrow, well-scoped sub-tasks; weaker on open-domain, multi-hop reasoning [6], [9]	Broad general reasoning across domains [1], [2]
Tool-call reliability	Competitive when fine-tuned for a fixed schema; degrades with novel/unseen APIs [16], [20]	Generalizes better to unseen tools with in-context prompting [19], [20]
Fine-tuning agility	Low-cost, hours-scale adaptation via LoRA/QLoRA [23], [24]	Costly; typically adapted via prompting rather than full fine-tuning
Latency / cost	Order-of-magnitude lower per-call cost and latency [5], [8]	Higher per-call latency and API cost
Energy / carbon	Substantially lower per-inference energy draw [42], [43]	Higher aggregate energy and water footprint at scale [41]–[43]
Privacy posture	Can execute fully on-device, avoiding data egress [44]–[46]	Typically requires transmission to a remote endpoint
Behavioral consistency	Easier to constrain to a single output format via targeted fine-tuning [5]	Must be prompt-engineered to avoid format drift across formats

IV. AGENTIC CAPABILITIES OF SLMS

Fig. 1 sketches the conceptual architecture shared, in broad outline, by the SLM-first agent designs reviewed in this section: an orchestrator decomposes an incoming task, dispatches sub-tasks to role-specialized SLMs for reasoning, planning and tool invocation, consults an episodic memory / retrieval store, and routes low-confidence outputs through a verifier and escalation path to a larger model. The remainder of this section reviews the evidence base for each labeled component in turn.

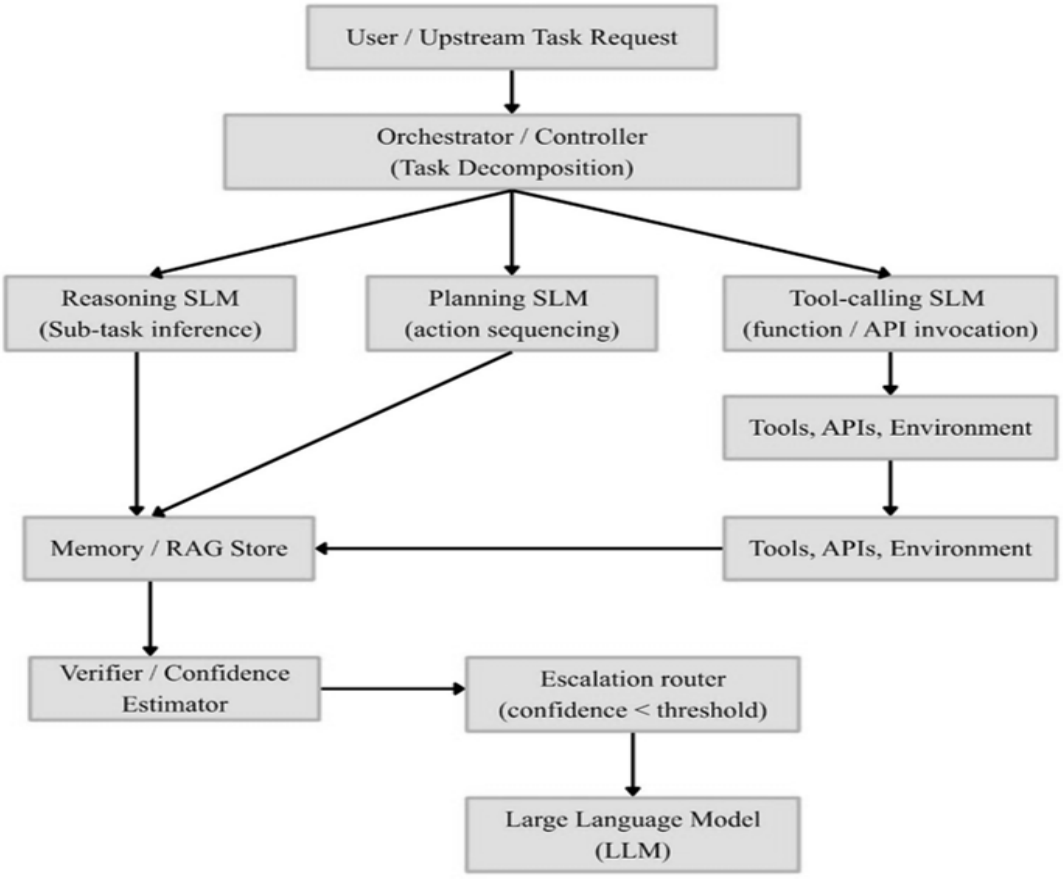

Fig. 1. Conceptual architecture of an SLM-first autonomous agent with a verification-gated escalation path to an LLM.

A. Reasoning

Reasoning ability in SLMs has advanced primarily through two complementary mechanisms: distillation from larger teacher models, and inference-time scaffolding. Distillation approaches such as "Distilling Step-by-Step" [25] and the reasoning-distillation method of Magister et al. [26] show that a smaller student model trained on chain-of-thought rationales generated by a larger teacher can outperform much larger models trained without such rationales, at a fraction of the training data. Fu et al. [27] similarly show that specializing a smaller model toward multi-step arithmetic and symbolic reasoning—rather than training it as a generalist—yields disproportionate gains on the targeted reasoning class. DeepSeek-R1-Distill models extend this idea to reasoning-oriented reinforcement-learning traces, distilling the reasoning behavior of a much larger reinforcement-learning-trained model into 1.5–8-billion-parameter students [15].

Inference-time scaffolding, most prominently chain-of-thought prompting [36] and self-consistency decoding [35], improves reasoning quality without any additional training by sampling multiple reasoning paths and aggregating over them. Belcak et al. [5] note that this class of technique is disproportionately valuable for SLMs precisely because test-time compute is comparatively cheap relative to the fixed cost of training or serving a larger model, making inference-time scaling an economically attractive lever specifically at the small end of the model-size spectrum.

The critical limitation, consistent across sources, is that SLM reasoning gains are narrowest exactly where agentic tasks are broadest: long-horizon, multi-hop reasoning that requires integrating disparate pieces of context still favors larger models with greater representational capacity [5], and the reasoning improvements reported for SLMs are concentrated on benchmarks resembling their fine-tuning or distillation

distribution rather than genuinely open-ended reasoning.

B. Planning

Planning—decomposing a goal into an ordered sequence of sub-tasks or actions—has been studied extensively for LLM-based agents [37], but far less for SLM-specific planners. The dominant pattern observed across agent frameworks [1], [3] delegates planning to the largest available model in the system (the "root agent" in Belcak et al.'s terminology [5]) while delegating individual sub-task execution to smaller models. This division of labor is architecturally convenient—planning requires broad contextual judgment while individual actions are often narrow and repetitive—but it also means that planning failures are disproportionately consequential: an incorrect decomposition propagates errors to every downstream SLM invocation regardless of how reliably each sub-task is executed. Memory-augmented planning approaches, reviewed by Wang et al. [37], mitigate this by retrieving previously successful task decompositions from an episodic memory store, which is of particular relevance to SLM-first agents because it substitutes retrieval (cheap) for reasoning depth (expensive at small scale).

C. Tool Use and Function Calling

Tool use is the agentic capability where SLMs have shown the clearest and most reproducible competitiveness with LLMs, precisely because function calling is a narrow, format-constrained generation task rather than an open-ended one. Toolformer [17] first demonstrated that a model with roughly 6.7 billion parameters could learn to invoke external APIs through self-supervised bootstrapping, outperforming a considerably larger non-tool-using baseline. ReAct [18] established the now-standard pattern of interleaving natural-language reasoning traces with discrete actions, and has since become the default scaffold for both LLM- and SLM-driven tool-calling agents. Benchmarks such as ToolLLM/ToolBench [19], the Berkeley Function-Calling Leaderboard [21] and τ-bench [22] provide the standardized evaluation surfaces against which subsequent fine-tuned models are compared.

Purpose-built small models have converged on the idea that general instruction-following capacity is not necessary for reliable tool calling; targeted fine-tuning on function-calling schemas is. Salesforce's xLAM-2-8B [16] and function-calling-focused open efforts fine-tuned on ToolBench-style data report tool-selection accuracy exceeding some substantially larger general-purpose models on curated evaluation sets. However, this competitiveness is conditional: small, general-purpose (i.e., not tool-fine-tuned) open models frequently attempt no tool call at all on scenarios requiring one, clustering at the bottom of Berkeley Function-Calling Leaderboard-style tiering [21], which indicates that the reported SLM tool-use successes are attributable to targeted fine-tuning rather than to an emergent property of small scale itself. This distinction matters directly for the reliability framework proposed in Section VI: an SLM's suitability for a tool-calling agent role is a property of its fine-tuning pipeline, not an intrinsic property of its parameter count, and verification should therefore be calibrated per fine-tuned checkpoint rather than assumed from model family alone.

D. Memory and Retrieval-Augmented Generation

Because SLMs typically operate with shorter effective context and lower parametric knowledge capacity than LLMs, memory and retrieval mechanisms are disproportionately important to their agentic viability. Retrieval-augmented generation (RAG) [39] supplies an SLM with task-relevant external context at inference time, substituting a lightweight retrieval step for costly parametric memorization. Agentic RAG extends this by embedding retrieval inside a reasoning-action loop rather than treating it as a single upstream step, allowing the agent to reformulate queries and iteratively refine what is retrieved [40]. Surveys of agent memory mechanisms [38] distinguish short-term working memory (the active context window), episodic memory (past task traces, often stored and retrieved via vector similarity, as in Generative Agents-style architectures), and semantic memory (structured or graph-based long-term knowledge). For SLM-first agents specifically, episodic memory of prior successful tool-call sequences and verification outcomes is particularly valuable because it allows an SLM to reuse a previously validated action pattern rather than re-deriving it from limited parametric reasoning capacity each time.

V. EFFICIENCY, LATENCY, ENERGY AND DEPLOYMENT CONSIDERATIONS

A. Computational and Latency Efficiency

The economic case for SLMs in agentic pipelines rests on a straightforward observation: agentic workloads issue many small, narrow model calls rather than few large conversational ones, so the marginal cost of each call dominates total system cost. Belcak et al. [5] report that serving a seven-billion-parameter SLM can be ten-to-thirty times cheaper, in latency, energy and floating-point operations, than serving a seventy-to-hundred-seventy-five-billion-parameter LLM, and that SLMs require little or no cross-accelerator parallelization, further reducing serving infrastructure overhead. Subramanian et al. [8] reach a similar qualitative conclusion from an independent survey of SLM efficiency literature, emphasizing that fine-tuning agility (parameter-efficient adaptation via LoRA [23] or QLoRA [24] in GPU-hours rather than GPU-days) compounds the cost advantage by allowing behavior fixes to be deployed quickly rather than accumulated into large, infrequent LLM prompt-engineering cycles.

B. Energy and Environmental Considerations

Inference, not training, increasingly dominates the aggregate energy footprint of deployed language models, because a single trained model is invoked repeatedly at scale for the remainder of its service life [42], [43]. Jegham et al. [43] benchmark energy, water and carbon footprint across thirty state-of-the-art models deployed in commercial data centers and find more than a sixty-five-fold difference in energy draw per long prompt between the most and least efficient systems studied, underscoring that model-size choice is a first-order lever for an agent's operational sustainability, not a secondary concern. Earlier lifecycle analyses of a single large model's training footprint [41] and infrastructure-level inference energy benchmarking [42] corroborate the broader pattern that per-call energy cost scales with model size and that this scaling compounds multiplicatively across the many narrow calls a typical agentic pipeline issues per user task.

C. Edge Deployment and Privacy

Because SLMs can execute on a single consumer-grade device, they enable an entire class of privacy-preserving agentic deployments that are structurally unavailable to data-center-hosted LLMs: sensitive user data can be processed locally and never transmitted off-device. Niu et al. [44] survey this pattern under the framing of on-device/cloud collaboration, in which the SLM either learns from a cloud LLM through distillation, supplies proxy signals that improve the cloud LLM, or co-evolves with it through privacy-preserving knowledge exchange. Apple's on-device foundation model deployment [45] is frequently cited as an industrial instantiation of this pattern, in which a small on-device model performs summarization, text refinement and other narrow tasks locally, escalating to a larger server-side model only for tasks that exceed on-device capability. Xu et al. [46] review the broader landscape of on-device language model deployment, noting that latency, battery consumption and thermal constraints—not privacy alone—are additional practical drivers of the shift toward SLM-first mobile and edge agent architectures.

Fig. 7 sketches, in purely qualitative terms, the shift in research attention motivating this review: interest in embedding SLMs directly inside agentic pipelines, and in cascade/routing mechanisms that decide when to invoke a larger model, has grown visibly across the period covered by the included studies. No bibliometric database query was performed to produce this curve; it is included only as a schematic orientation for the reader and should not be read as a measured publication count.

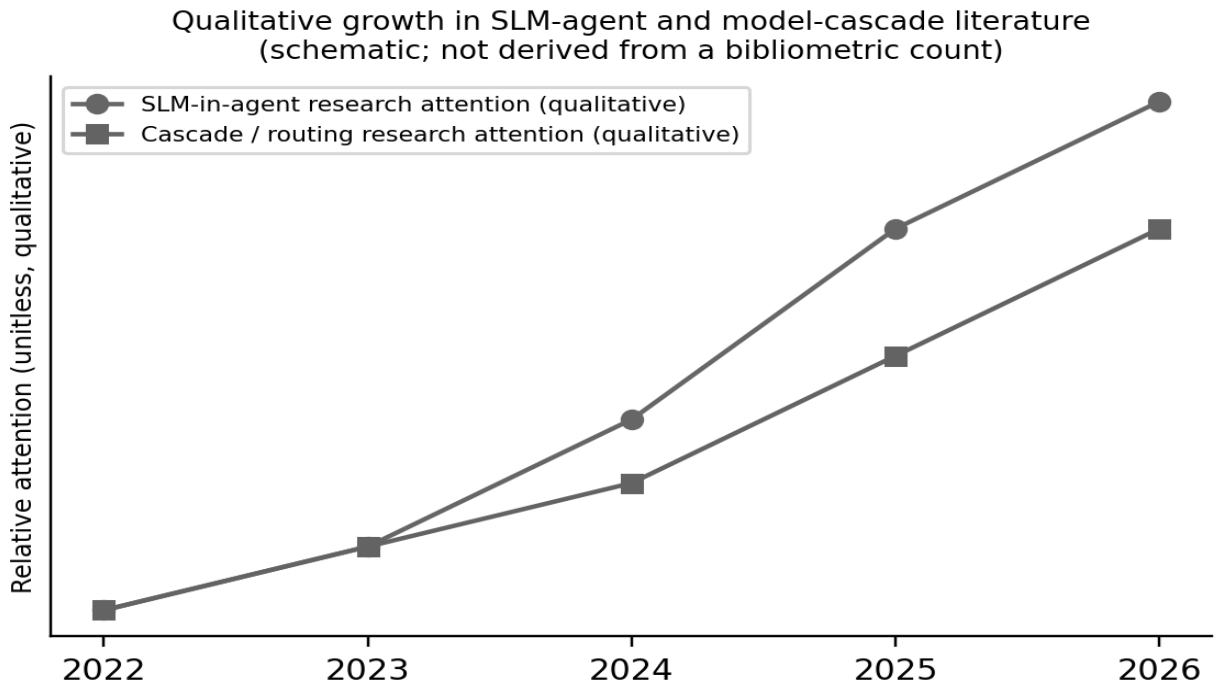

Fig. 7. Conceptual illustration only. No bibliometric database query was performed; the curve encodes the qualitative trend observed while conducting this review, not measured publication counts.

Taken together, Sections III–V establish the empirical case that SLMs are viable, efficient and increasingly capable components of agentic systems, but that this viability is capability- and task-dependent rather than universal. This motivates the central methodological question addressed by the remainder of the paper: how should an agent designer decide, systematically rather than ad hoc, which sub-tasks an SLM may handle unassisted, which require verification, and which must be escalated to a larger model.

VI. PROPOSED RELIABILITY-FOCUSED TAXONOMY AND ADAPTIVE ESCALATION FRAMEWORK

A. Motivation and Positioning

The literature reviewed in Sections IV and V establishes two facts that existing frameworks do not jointly exploit. First, SLM capability is highly heterogeneous across agentic roles: an SLM fine-tuned for function calling can rival much larger general-purpose models on that narrow task [16], [21] while remaining unreliable on multi-hop reasoning within the same agent pipeline [5], [6]. Second, the mechanisms that exist to manage this heterogeneity—model cascading [28]–[31], self-verification [32], [34]–[36] and multi-agent failure analysis [33]—have developed largely independently, each addressing one stage of the problem (respectively: which model to call, whether to trust its output, and what goes wrong when trust is misplaced) without a shared vocabulary connecting the three.

The proposed taxonomy closes this gap by defining a five-stage chain—SLM Capability → Agent Role → Failure Mode → Verification → Adaptive Escalation—in which each stage's output is the next stage's input, illustrated in Fig. 3. This is not merely a relabeling of cascading or routing; the key structural difference, elaborated in Section VIII, is that existing cascades route a query to a model tier before execution based on predicted difficulty [28]–[31], whereas the proposed framework couples the routing decision to an explicit, role-specific verification signal computed after the SLM has already attempted the sub-task, and treats the failure taxonomy itself (Fig. 6) as the shared reference against which both the verification mechanism and the escalation trigger are defined.

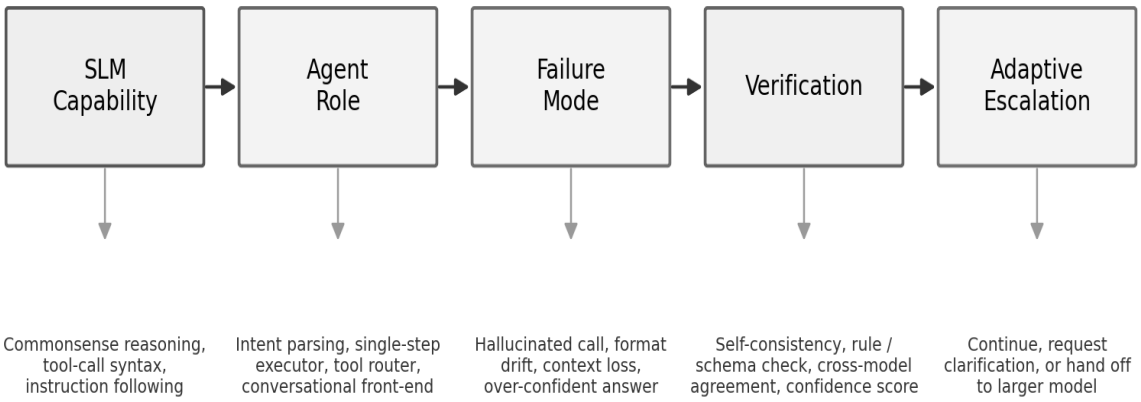

Fig. 3. Proposed reliability-focused taxonomy linking SLM capability to agent role, characteristic failure mode, verification mechanism and escalation outcome.

B. Stage 1–2: From SLM Capability to Agent Role

The first link in the chain maps a demonstrated SLM capability—commonsense reasoning, tool-call syntax adherence, instruction following, retrieval-conditioned generation—onto one of a small number of canonical agent roles: intent parser, single-step executor, tool router, planning assistant, or conversational front-end. This mapping is deliberately coarse-grained because, as Section IV-C shows, the reliability of a given capability is a property of the specific fine-tuned checkpoint and its training distribution, not of the base model family in the abstract [16], [21]. A role assignment therefore functions as a hypothesis to be continuously tested by the verification stage, not as a fixed classification made once at design time.

C. Stage 3: Failure Modes

Each agent role carries a characteristic failure mode when staffed by an SLM rather than an LLM. Drawing on the empirically derived multi-agent system failure taxonomy of Cemri et al. [33]—which identifies fourteen failure modes across system-design, inter-agent-misalignment and task-verification categories from over sixteen hundred annotated execution traces—and adapting it to the single-agent, capability-constrained setting of an SLM executor, we group SLM-specific failures into three classes, shown in Fig. 6: capability-mismatch failures (the sub-task exceeds the SLM's demonstrated reasoning depth or requires long-tail knowledge it was not fine-tuned on), execution/tool-interface failures (malformed or hallucinated function calls, output-format drift, or loss of context under a constrained window), and verification-gap failures (a verifier accepts an over-confident wrong answer, no escalation trigger exists for a given task class, or the verification step itself costs more than simply escalating would have). This adaptation is presented as a conceptual extension grounded in [33]'s empirical categories rather than as a new independently validated taxonomy, since a dedicated empirical study of SLM-specific agent traces analogous to MAST's multi-agent trace analysis remains, to our knowledge, absent from the literature—a gap made explicit in Section IX.

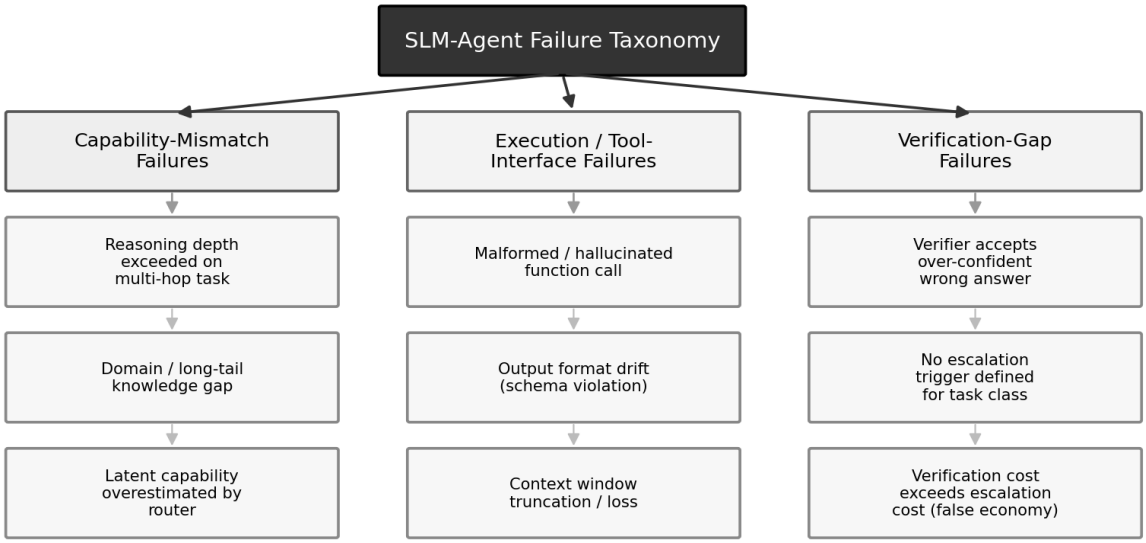

Fig. 6. Conceptual failure taxonomy for SLM-based agents, adapted from empirically grounded multi-agent failure categories (MAST [33]) and extended with capability- and verification-specific failure classes relevant to small models.

D. Stage 4: Verification Mechanisms

The verification stage estimates the reliability of an SLM's output before it is accepted or acted upon. The framework does not prescribe a single verification technique, because the appropriate mechanism depends on the failure mode most likely for the assigned role; instead, it organizes the applicable mechanisms surveyed in Sections IV–V into a role-indexed menu. For free-text or reasoning outputs, self-consistency across multiple sampled generations [35] and SelfCheckGPT-style consistency scoring [34] provide reference-free reliability estimates. For context-grounded answers, few-shot self-verification of the kind used in AutoMix [32]—in which the same or a comparably sized model is prompted to judge its own answer against the supplied context—offers a low-cost check without requiring a separately trained verifier. For tool calls, schema and type validation against the target API's specification provides a deterministic, zero-cost verification layer that catches format-drift failures before they reach an execution environment. Because self-verification signals are known to be noisy [32], the framework follows AutoMix's precedent of treating the raw verification score as an input to a second-stage meta-decision (Stage 5) rather than as a direct accept/reject gate.

E. Stage 5: Adaptive Escalation

The final stage converts a (task, verification-confidence) pair into one of three actions, shown in Fig. 4: an easy, low-risk task with no adverse verification signal is answered directly by the SLM; an uncertain, medium-risk task is answered by the SLM but only after an explicit verification pass, with the verification outcome determining whether the answer is returned or escalated; and a complex or high-risk task—identified either by the difficulty/risk classifier prior to execution, or by a verification failure after execution—is escalated to a larger model with full context handoff. Risk, in this framework, is deliberately treated as a first-class input alongside difficulty: a task can be simple in reasoning terms yet high-stakes (an irreversible financial transaction, a safety-relevant control action), in which case escalation or human-in-the-loop confirmation is warranted regardless of the SLM's measured confidence. This risk-sensitivity is the second

structural point of departure from cost-accuracy cascades such as FrugalGPT [28] and HybridLLM [29], which optimize purely for expected accuracy at a given cost and have no explicit notion of task stakes.

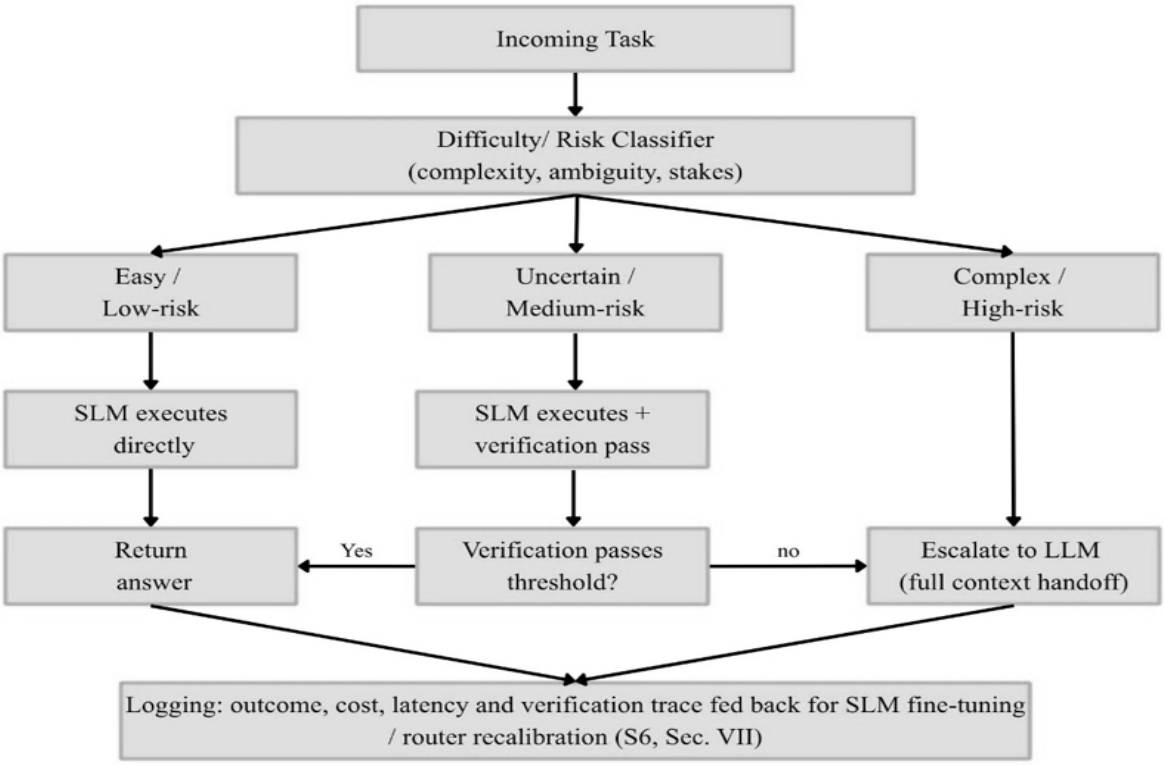

Fig. 4. Adaptive escalation decision framework: routing outcome depends jointly on task difficulty/risk and verification confidence.

Every escalation decision, together with its cost, latency and verification trace, is logged and fed back into the system (bottom of Fig. 4). This closes the loop opened at Stage 1: recurring escalations for a given role indicate either that the SLM assigned to that role should be re-fine-tuned on the escalated cases, or that the role's difficulty/risk classifier is miscalibrated, echoing the iterative refinement step of the LLM-to-SLM conversion procedure described by Belcak et al. [5, S6].

VII. FAILURE MODES AND MITIGATION STRATEGIES

Table III consolidates the failure-mode and verification analysis of Section VI-C–D into a single reference table, cross-referencing each failure class with the mitigation mechanism(s) reported in the literature and their principal limitation.

TABLE III. Failure Modes and Corresponding Verification / Mitigation Strategies

Failure class	Representative failure	Verification / mitigation	Principal limitation
Capability mismatch	Reasoning depth exceeded on multi-hop task [5], [6]	Self-consistency [35]; escalation to LLM	Consistency checks add inference cost; do not fix the underlying gap
Capability mismatch	Domain / long-tail knowledge gap	Retrieval-augmented generation [39], [40]	Retrieval quality bounds answer quality; stale or sparse corpora limit recall

Execution / tool interface	Malformed or hallucinated function call [16], [21]	Schema / type validation before execution	Catches format errors, not semantically wrong but well-formed calls
Execution / tool interface	Output-format drift	Constrained decoding; single fine-tuned output format [5]	Requires fine-tuning investment per deployment
Execution / tool interface	Context-window truncation / loss	Episodic memory retrieval [37], [38]	Retrieval latency; relevance-ranking errors
Verification gap	Verifier accepts over-confident wrong answer	Meta-verification / POMDP-based routing [32]	Meta-verifier itself requires calibration data
Verification gap	No escalation trigger defined for task class	Role-indexed verification menu (Sec. VI-D)	Requires up-front role taxonomy; incomplete coverage for novel roles
Verification gap	Verification cost exceeds escalation cost	Cost-aware calibration (cf. cascade routing [28]–[31])	Requires reliable cost model, which drifts as pricing/hardware change

VIII. EXISTING APPROACHES VS. THE PROPOSED FRAMEWORK

Model cascading and routing methods share the proposed framework's goal of using a smaller model wherever possible while preserving accuracy, but differ from it in scope and decision structure. FrugalGPT [28] learns a scoring function over a sequence of increasingly capable APIs and stops at the first model whose output clears a learned threshold; it optimizes single-turn query answering and does not model multi-step agentic execution or task risk. HybridLLM [29] and RouteLLM [30] learn a binary or preference-based router that selects a model before generation, based on predicted query difficulty, and report substantial reductions in large-model calls (up to forty percent for HybridLLM [29]) without measurable quality loss; both, however, commit to a model choice before seeing the candidate's actual output, so they cannot catch failures that only manifest after generation. RouterBench [31] standardizes evaluation across such routers but likewise evaluates single-turn selection rather than post-hoc verification. AutoMix [32] is the closest existing analogue to the verification stage of the proposed framework, using a small model's self-verification of its own answer to decide whether to escalate, refined through a partially observable Markov decision process to counteract verification noise; AutoMix, however, is designed for single-turn, context-grounded question answering and does not address tool-calling agent roles, multi-step task decomposition, or an explicit risk dimension separate from predicted accuracy.

Table IV summarizes these distinctions. The proposed framework's contribution is not a new routing algorithm to compete with FrugalGPT, HybridLLM or AutoMix on a benchmark—each remains a valid implementation choice for the framework's Stage-4/5 mechanics—but rather the taxonomy that makes explicit which agent role, which failure mode and which risk level a given verification-and-escalation policy is actually addressing, which none of the individual cascading, routing or verification papers surveyed provide in isolation.

TABLE IV. Comparison of Existing Cascading / Verification Approaches with the Proposed Framework

Approach	Unit of decision	Timing of decision	Risk-aware?	Agentic role granularity
FrugalGPT [28]	Single query	Sequential, post hoc per stage	No	None (query-level only)
HybridLLM [29]	Single query	Pre-generation (binary router)	No	None
RouteLLM [30]	Single query	Pre-generation (preference-based)	No	None
RouterBench [31]	Evaluation harness	N/A (benchmark, not a policy)	No	None

AutoMix [32]	Single query	Post-generation self-verification	Partial (via POMDP router)	None
MAST [33]	Multi-agent trace	Post hoc failure analysis	N/A (diagnostic, not a policy)	Cross-agent, not per-role
Proposed framework	Agent sub-task / role	Post-generation, role-indexed verification	Yes (explicit difficulty x risk)	Explicit per-role taxonomy with feedback loop

IX. RESEARCH GAPS AND FUTURE DIRECTIONS

Several concrete gaps follow directly from the synthesis above rather than from a generic call for further study.

First, no empirical failure-trace study analogous to MAST [33] currently exists specifically for single-agent, SLM-staffed pipelines; the failure taxonomy proposed in Fig. 6 is adapted from multi-agent LLM traces and would benefit from validation against annotated SLM-agent execution logs at comparable scale.

Second, self-verification mechanisms such as AutoMix's [32] and SelfCheckGPT's [34] have been validated primarily on context-grounded question answering; their transferability to tool-calling and multi-step planning roles—where a "correct" output is a valid action rather than a factual statement—is largely untested, and schema validation alone (Section VII) cannot catch semantically incorrect but well-formed tool calls.

Third, existing cascade and routing benchmarks [28]–[31] evaluate cost-accuracy trade-offs on single-turn queries; there is no standardized benchmark that evaluates an escalation policy across a full multi-step agentic trajectory, where an early wrong escalation decision compounds through every subsequent step. Constructing such a benchmark, ideally using the role taxonomy proposed in Section VI-B as its organizing structure, would allow direct comparison of escalation policies in the setting they are actually deployed in.

Fourth, energy and carbon accounting for agentic systems has so far been studied at the level of individual model calls [42], [43] rather than end-to-end agentic sessions, which may issue dozens of SLM calls, a handful of verification passes, and occasional LLM escalations per user task; a lifecycle accounting methodology that attributes energy cost to the escalation policy itself, not only to the underlying model sizes, would let designers optimize sustainability directly rather than as a side effect of cost optimization.

Fifth, privacy-preserving escalation—handing off from an on-device SLM to a cloud LLM without transmitting the sensitive context that triggered escalation in the first place—remains largely unaddressed; the on-device/cloud collaboration surveys reviewed in Section V-C [44]–[46] describe distillation and proxy-signal patterns for training-time collaboration but say comparatively little about privacy-preserving context handoff at inference time, which is precisely the moment the proposed framework's escalation step occurs.

X. CONCLUSION

Small language models have matured to the point where, for a substantial share of agentic sub-tasks, they offer competitive capability at a fraction of the latency, computational and energy cost of large language models. This review has synthesized evidence for that claim across reasoning, planning, tool use and memory, and has shown that the operational case for SLMs—lower latency, lower cost, on-device privacy—is well documented but conditional on task type and fine-tuning quality rather than universal. The paper's central contribution, a Reliability-Focused Taxonomy connecting SLM Capability, Agent Role, Failure Mode, Verification and Adaptive Escalation, is offered as a unifying structure for reasoning about that conditionality explicitly, rather than leaving it implicit in ad hoc engineering choices. The framework does not replace existing cascading, routing or self-verification techniques; it situates them as interchangeable implementations of its Stage-4 and Stage-5 mechanics, and adds the missing connective layer—an explicit failure taxonomy and risk-aware escalation trigger—that ties model choice to agent role rather than to query difficulty alone. The research gaps identified in Section IX, particularly the absence of a multi-step

agentic escalation benchmark and of SLM-specific failure-trace studies at MAST-like scale, define a concrete agenda for the next stage of work in this area.

REFERENCES

[1] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z.-Y. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J.-R. Wen, "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, no. 6, 2024.

[2] Z. Xi et al., "The rise and potential of large language model based agents: A survey," arXiv:2309.07864, 2023.

[3] T. Masterman, S. Besen, M. Sawtell, and A. Chao, "The landscape of emerging AI agent architectures for reasoning, planning, and tool calling: A survey," arXiv:2404.11584, 2024.

[4] J. Luo et al., "Large language model agent: A survey on methodology, applications and challenges," arXiv:2503.21460, 2025.

[5] P. Belcak, G. Heinrich, S. Diao, Y. Fu, X. Dong, S. Muralidharan, Y. C. Lin, and P. Molchanov, "Small language models are the future of agentic AI," arXiv:2506.02153, 2025.

[6] Z. Lu, X. Li, D. Cai, R. Yi, F. Liu, X. Zhang, N. D. Lane, and M. Xu, "Small language models: Survey, measurements, and insights," arXiv:2409.15790, 2024.

[7] F. Wang et al., "A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with LLMs, and trustworthiness," arXiv:2411.03350, 2024.

[8] S. Subramanian, V. Elango, and M. Gungor, "Small language models (SLMs) can still pack a punch: A survey," arXiv:2501.05465, 2025.

[9] C. V. Nguyen et al., "A survey of small language models," arXiv:2410.20011, 2024.

[10] M. Abdin et al., "Phi-3 technical report: A highly capable language model locally on your phone," arXiv:2404.14219, 2024.

[11] M. Javaheripi and S. Bubeck, "Phi-2: The surprising power of small language models," Microsoft Research Blog, Dec. 2023.

[12] L. B. Allal et al., "SmolLM2: When Smol goes big -- data-centric training of a small language model," Hugging Face technical report, 2025.

[13] A. Blakeman et al., "Nemotron-H: A family of accurate and efficient hybrid Mamba-Transformer models," arXiv:2504.03624, 2025.

[14] X. Dong et al., "Hymba: A hybrid-head architecture for small language models," arXiv:2411.13676, 2024.

[15] DeepSeek-AI, "DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning," arXiv:2501.12948, 2025.

[16] J. Zhang et al., "xLAM: A family of large action models to empower AI agent systems," arXiv:2409.03215, 2024.

[17] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in Proc. NeurIPS, 2023.

[18] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in Proc. ICLR, 2023.

[19] Y. Qin et al., "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," in Proc. ICLR, 2024.

[20] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive APIs," arXiv:2305.15334, 2023.

[21] F. Yan, H. Mao, C. C.-J. Ji, T. Zhang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "Berkeley function calling leaderboard," Univ. of California, Berkeley, 2024. [Online].

[22] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, "tau-bench: A benchmark for tool-agent-user interaction in real-world domains," arXiv:2406.12045, 2024.

[23] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in Proc. ICLR, 2022.

[24] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in Proc. NeurIPS, 2023.

[25] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, and T. Pfister, "Distilling step-by-step! Outperforming larger language models with less training data and smaller model sizes," arXiv:2305.02301, 2023.

[26] L. C. Magister, J. Mallinson, J. Adamek, E. Malmi, and A. Severyn, "Teaching small language models to reason," in Proc. 61st Annu. Meeting Assoc. Comput. Linguistics, 2023, pp. 1773-1781.

[27] Y. Fu, H. Peng, L. Ou, A. Sabharwal, and T. Khot, "Specializing smaller language models towards multi-step reasoning," in Proc. ICML, 2023.

[28] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to use large language models while reducing cost and improving performance," arXiv:2305.05176, 2023.

[29] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Ruhle, L. V. S. Lakshmanan, and A. H. Awadallah, "Hybrid LLM: Cost-efficient and quality-aware query routing," in Proc. ICLR, 2024.

[30] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica, "RouteLLM: Learning to route LLMs from preference data," arXiv:2406.18665, 2024.

[31] Q. J. Hu, J. Bieker, X. Li, N. Jiang, B. Keigwin, G. Ranganath, K. Keutzer, and S. K. Upadhyay, "RouterBench: A benchmark for multi-LLM routing system," arXiv:2403.12031, 2024.

[32] P. Aggarwal, A. Madaan, A. Anand, S. P. Potharaju, S. Mishra, P. Zhou, A. Gupta, D. Rajagopal, K. Kappaganthu, Y. Yang, S. Upadhyay, Mausam, and M. Faruqui, "AutoMix: Automatically mixing language models," arXiv:2310.12963, 2023.

[33] M. Cemri et al., "Why do multi-agent LLM systems fail?," arXiv:2503.13657, 2025.

[34] P. Manakul, A. Liusie, and M. J. F. Gales, "SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models," in Proc. EMNLP, 2023.

[35] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in Proc. ICLR, 2023.

[36] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Proc. NeurIPS, 2022.

[37] L. Wang et al., "Understanding the planning of LLM agents: A survey," arXiv:2402.02716, 2024.

[38] Z. Zhang et al., "A survey on the memory mechanism of large language model based agents," arXiv:2404.13501, 2024.

[39] Y. Gao et al., "Retrieval-augmented generation for large language models: A survey," arXiv:2312.10997, 2024.

[40] "Agentic retrieval-augmented generation: A survey on agentic RAG," arXiv:2501.09136, 2025.

[41] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of BLOOM, a 176B parameter language model," J. Mach. Learn. Res., vol. 24, no. 253, 2023.

[42] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally, "From words to watts: Benchmarking the energy costs of large language model inference," in Proc. IEEE High Performance Extreme Computing Conf. (HPEC), 2023.

[43] N. Jegham, M. Abdelatti, L. Elmoubarki, and A. Hendawi, "How hungry is AI? Benchmarking energy, water, and carbon footprint of LLM inference," arXiv:2505.09598, 2025.

[44] C. Niu et al., "Collaborative learning of on-device small model and cloud-based large model: Advances and future directions," arXiv:2504.15300, 2025.

[45] T. Gunter et al., "Apple intelligence foundation language models," arXiv:2407.21075, 2024.

[46] J. Xu, Z. Li, W. Chen, Q. Wang, X. Gao, Q. Cai, and Z. Ling, "On-device language models: A comprehensive review," arXiv:2409.00088, 2024.