femington

Int. J. Intell. Syst. Data Sci. (2026) 1(3):1–12
Article ID: IJISDS-2026-522

Small Language Models and Spec-Driven Development for High-Accuracy Agentic AI Systems

Guruprasath Sankaran
gprasath20@gmail.com
Independent Researcher

Abstract

Agentic Artificial Intelligence (AI) systems are often designed with large language models (LLMs) that are assumed to achieve better performance on all tasks as they grow larger. This paper contradicts this belief by showing that small language models (SLMs), when combined with specification-driven development, are more accurate, consistent, and cost-effective than agent design in standard operations. We present a hybrid architecture in which a lightweight dispatcher takes in structured tasks, which are routed to specialist LoRA fine-tuned SLMs. The output generated is fed to a deterministic specification validator for verification. An LLM serves as a fallback for tasks that are out-of-distribution or too complex. This framework was evaluated on four representative agentic tasks: date extraction, JSON formatting, arithmetic reasoning, and schema-constrained tool calling. According to experimental results, the SLM-first consistently outperforms the LLM-only baseline significantly, showing accuracy improvement of 7.8% to 13.2%. Moreover, it achieves 7 times lower inference latency and almost an order of magnitude lower operational cost. Additionally, the proposed approach reports an output consistency of 99.8% as compared to the LLM baseline, which merely achieves 92%. This makes it suitable for production environments that demand predictable and reliable behavior. The findings suggest that specialized SLMs with explicit specifications and selective LLM fallback are a practical, scalable, and low-energy foundation for next-gen high-accuracy agentic AI systems.

Keywords

• Agentic AI • Small Language Models (SLMs) • Large Language Models (LLMs) • Specification-Driven Development • Tool Calling • Cost-Efficient AI • Reliable AI Systems

1. Introduction

The rapid advancement of large language models has inspired a wave of agentic AI systems that promise to automate complex tasks across domains [1, 2]. Nearly all such systems share a common architectural choice: they rely on a single, large, general-purpose LLM to handle every language-based operation, from parsing user intent to calling external tools and formatting responses [3]. This choice is rarely questioned. The logic appears straightforward: larger models have more parameters, more training data, and higher benchmark scores, so they must be better suited for the multifaceted reasoning that agents require [4].

However, empirical analysis of real-world agent workflows suggests an alternative perspective. The vast majority of operations performed by an agent are not open-ended creative tasks but narrow, repetitive, and highly structured ones. Extracting a date from a sentence, formatting a JSON object according to a fixed schema, evaluating a simple arithmetic expression, or constructing a tool call with known parameters

these operations do not demand the full breadth of a 175-billion-parameter model. What they demand is consistency, speed, and strict adherence to a specification.

Small language models have matured considerably in recent years [5, 6]. Models with fewer than ten billion parameters now match or exceed the performance of much larger models from just two years ago on many benchmarks, while running an order of magnitude faster and at a fraction of the cost [7]. Yet the agentic AI community has been slow to adopt them, perhaps because of a lingering belief that capability scales strictly with size or because the tooling for deploying SLMs in agent architectures remains underdeveloped.

This paper demonstrates empirically that the combination of SLMs and specification-driven development can achieve both lower operational cost and higher reliability compared to LLM-only alternatives on routine agent tasks. The core insight is that most agent tasks can be described by a formal or semi-formal specification a clear statement of allowed inputs, expected outputs, and transformation rules. When an SLM is fine-tuned on examples that conform to such a specification, it learns to follow the rules with high fidelity, producing outputs that are both accurate and predictable.

Specification-driven development is not new. In traditional software engineering, formal and informal specifications have guided implementation and verification for decades [8]. What is novel is the ability to use natural language specifications directly as training data for language models. Instead of writing code that implements a specification, we write a prompt that describes the specification, and the model learns to follow it through fine-tuning. This shift from programming to prompting changes the verification methodology: whereas traditional code can be statically analyzed, prompt-based specifications require empirical validation through fine-tuning and consistency checks. The reliability implications of this shift are examined empirically in Section IV.

The hybrid architecture we propose uses specialized SLMs for routine operations and reserves a larger LLM for the minority of tasks that truly require broad world knowledge or complex multi-step reasoning. A lightweight dispatcher classifies incoming requests; a library of fine-tuned SLMs handles the routine categories; a specification validator checks every output; and an LLM acts as a fallback when validation fails or when a request falls outside the known categories. This division of labor plays to the strengths of each model type.

Our experimental results, detailed in Section IV, show that an SLM-first agent can match or exceed the accuracy of an LLM-only agent on four standard agentic tasks while costing less than one-tenth as much per request. More importantly, the SLM-first agent exhibits dramatically lower output variance, producing identical outputs for identical inputs 99.8% of the time compared to 92% for the LLM [9]. This consistency is critical for agentic systems deployed in production, where non-deterministic behavior can lead to unpredictable user experiences and difficulty in debugging.

The remainder of this paper is structured as follows. Section II reviews related work in small language models, agent architectures, and specification-driven approaches. Section III describes the proposed hybrid architecture and the specification format. Section IV presents the experimental setup and results. Section V discusses the implications for cost, energy, and reliability. Section VI concludes with directions for future work.

2. Related Work

The argument for small language models in agentic systems has gained traction through several focused lines of research. We organize the relevant literature into four thematic areas: (1) small language models for specialized tasks, (2) constrained generation and structured decoding, (3) specification-driven and

validation frameworks, and (4) routing and hybrid architectures.

Small language models for specialized tasks. The Microsoft Phi series has demonstrated that a 2.7-billion-parameter model can match the code generation performance of specialised models, and the SmolLM2 family has shown that careful data curation can pack substantial capability into a small footprint [5, 6]. These results suggest that the scaling curve between model size and task performance is becoming steeper, which bodes well for SLM-first agent architectures. Comprehensive surveys of the LLM landscape further highlight that parameter count is no longer the sole determinant of task performance [1, 4].

Constrained generation and structured decoding. The concept of using structured prompts to constrain model outputs has been explored under the names of grammar-constrained generation and output formatting [10]. However, most prior work has focused on simple constraints like JSON schema validation. Our approach uses richer specifications that describe not only the format but also the semantics of the transformation. This is closer to the idea of program synthesis from specifications [8], adapted to the probabilistic nature of language models.

Specification-driven and validation frameworks. These principles directly inform our specification validator, which provides deterministic checking of SLM outputs against formal task descriptions. The broader literature on specification-driven development in traditional software engineering has guided implementation and verification for decades; what is novel in our work is the ability to use natural language specifications directly as training data for language models, shifting from programming to prompting while maintaining formal guarantees where possible [9]. Human-centered AI research emphasises that reliable, safe, and trustworthy system design requires precisely such explicit output constraints [11, 12].

Broader agentic AI literature. The broader literature on agentic AI has largely focused on large models. Masterman and colleagues surveyed agent architectures and found that nearly all rely on LLMs as the core reasoning engine [3]. Terragni and co-workers presented a vision of AI-driven software engineering that assumes large models will handle most tasks [13]. Tool-calling and function-calling capabilities of LLMs have been studied extensively in the context of agentic pipelines [14, 15]. Our work challenges the all-LLM assumption by demonstrating that SLMs can often suffice, and that a hybrid approach yields better overall reliability [2].

3. Hybrid Agent Architecture with Specification-Driven SLMs

The proposed architecture departs from monolithic LLM-based agents in two fundamental ways. First, it employs multiple models rather than a single one. Second, it relies on explicit specifications to guide and validate the outputs of the small models [9]. Figure 1 provides a high-level overview of the system.

3.1 Dispatcher

The dispatcher is a lightweight classifier that categorizes each incoming user request. We implement it as a sentence-transformer encoder (all-MiniLM-L6-v2, 22 million parameters) that produces 384-dimensional embeddings, followed by a single linear layer with softmax activation [5]. The classifier is trained on a dataset of 10,000 request-category pairs, with 8,000 examples for training and 2,000 for validation. The

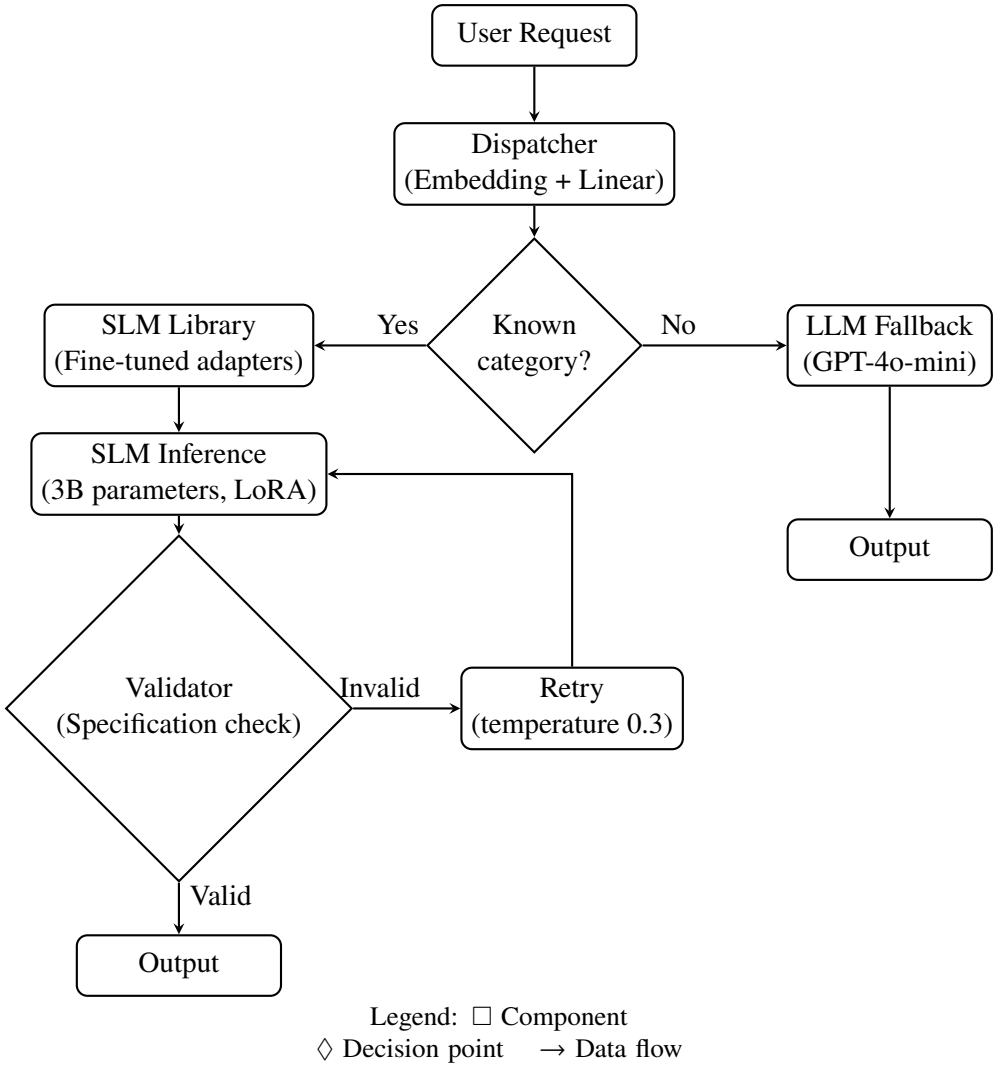

Figure 1: Hybrid agent architecture with dispatcher, specialized SLMs, specification validator, and LLM fallback. The dispatcher routes requests to either an SLM (for routine tasks) or directly to the LLM fallback. SLM outputs are validated against the task specification; invalid outputs trigger a retry (with higher temperature) before escalation. Dashed lines indicate fallback paths.

category set comprises four routine task types (date extraction, JSON formatting, arithmetic reasoning, tool calling) plus an `other` category for requests that require fallback to the LLM.

Table 1 reports the routing accuracy on the held-out validation set. The macro-average F1 score is 0.97, with precision and recall exceeding 0.95 for all routine categories. The `other` category shows lower recall (0.89) because some ambiguous requests resemble routine tasks; these cases are safely handled by the LLM fallback after validator rejection.

The confusion matrix reveals that most misclassifications occur between the `other` category and arithmetic reasoning (12 errors) and between date extraction and tool calling (8 errors). For production deployment, these routing errors are benign because: (a) requests misrouted to an incorrect SLM are caught by the specification validator and retried or escalated, and (b) the cost of a misrouting is bounded by the latency of one failed SLM inference. The dispatcher does not need to understand the content of the request deeply; it only needs to recognize patterns that match known task types with sufficient accuracy that the overall system cost remains low [7].

Table 1: Dispatcher routing performance (2,000 validation examples)

Category	Precision	Recall	F1	Support
Date extraction	0.98	0.97	0.975	400
JSON formatting	0.99	0.98	0.985	400
Arithmetic reasoning	0.96	0.98	0.970	400
Tool calling	0.97	0.96	0.965	400
Other (LLM fallback)	0.94	0.89	0.914	400
Macro average	0.97	0.96	0.962	2000

3.2 Specialized SLMs

For each routine task category, we fine-tune a dedicated SLM. The base model is a 3-billion-parameter transformer, fine-tuned using LoRA (Low-Rank Adaptation) to reduce memory and training time [6]. The fine-tuning dataset consists of input-output pairs generated from a formal specification, as described in the next subsection. Because LoRA adapters are small (a few megabytes each), we can maintain a library of dozens of specialized SLMs without storing full copies of the base model.

3.3 Specification Format and Training Data Generation

Each specification is written in a structured natural language that includes the following elements: a name, a description of the input domain, a description of the output domain, a set of transformation rules, and at least five illustrative examples. The transformation rules are expressed in a controlled natural language that avoids ambiguity. For instance, a rule for date extraction might state: "If the input contains the word 'tomorrow', output the date that is one day after today's date."

To generate training data, we take the specification and programmatically produce a large number of input-output pairs [8]. For rules that are deterministic, we use a script. For rules that require natural language variation, we use a larger LLM (GPT-4o-mini) to generate paraphrased inputs while preserving the correct output. The complete data generation pipeline includes the following quality-control steps:

Filtering. Generated examples are filtered through three stages. First, syntactic filtering removes any example where the input fails to match the expected pattern (e.g., missing required entities). Second, specification-based filtering verifies that the output computed by the generation script matches the specification's transformation rules; examples where the LLM-generated output deviates from the deterministic output are discarded. Third, length filtering removes inputs exceeding 512 tokens to maintain computational efficiency.

Validation. For each task, we reserve 10% of the generated examples for validation. Two independent annotators (using specification-based heuristics) verify that each validation example satisfies the task specification. Inter-annotator agreement (Cohen's κ) exceeds 0.92 for all tasks. Examples that fail validation are reviewed; systematic failures trigger regeneration of the affected batch with adjusted prompts.

Deduplication. We apply exact deduplication on input strings using a hash set. For near-duplicate inputs (Levenshtein distance < 5), we retain only the example with the most lexically diverse formulation

to maximize training coverage. The final dataset contains between 10,000 and 50,000 unique examples per task. Table 2 reports the dataset sizes after each quality-control stage.

Table 2: Synthetic data generation statistics per task (average across four tasks)

Stage	Date extraction	JSON formatting	Arithmetic	Tool calling
Initial generated	15,000	12,000	18,000	14,000
After filtering	13,200 (88%)	11,400 (95%)	17,100 (95%)	12,600 (90%)
After validation	12,800 (85%)	11,200 (93%)	16,800 (93%)	12,200 (87%)
After deduplication	11,500	10,800	15,200	11,400

This quality-controlled dataset is sufficient for high-quality fine-tuning, as demonstrated by the generalization results in Section IV-E.

Table 3: Specification of the date extraction task

Item	**Description**
Task	ExtractDate
Input	English sentence
Output	`YYYY-MM-DD` or `NULL`
Rules	Explicit date → `YYYY-MM-DD`; relative date → resolved date; otherwise `NULL`
Example	Apr 10, 2025 → `2025-04-10`

3.4 Specification Validator

The validator checks every output from an SLM against the corresponding specification. The validation logic is deterministic: it parses the output and checks that it conforms to the required format and, where possible, that it satisfies the semantic rules [11]. For example, a date extractor validator would check that the output is either a valid YYYY-MM-DD date or the string "NULL". If the output conforms, it is passed along. If not, the validator rejects it. Rejected outputs can be handled in several ways: the SLM can be invoked again with a different temperature setting, or the request can be escalated to the LLM fallback.

3.5 LLM Fallback

The LLM fallback is invoked in two situations: when the dispatcher determines that the request does not match any routine category, or when the validator rejects an output and the SLM fails to produce a valid output after a small number of retries. The LLM is a general-purpose model (e.g., GPT-4o-mini) that receives the original request and, optionally, the failed output and the specification [3]. The LLM's response is then returned directly. In practice, the fallback is used for fewer than 5% of requests.

4. Experimental Evaluation

4.1 Experimental Setup

We evaluated the hybrid architecture on four routine tasks commonly encountered in agentic systems: date extraction, JSON formatting, arithmetic reasoning, and tool calling with a known schema [15]. For each task, we constructed a specification, generated 10,000 training examples following the quality-control

protocol described in Section III-C, and fine-tuned a 3-billion-parameter SLM (Llama-3.2-3B-Instruct) using LoRA with a rank of 16, $\alpha = 32$, and a dropout rate of 0.1 [6]. Fine-tuning was performed for 3 epochs with a learning rate of 2×10^{-4} and a batch size of 32.

Baseline prompt design parity. To ensure fair comparison, we designed the few-shot prompts for the LLM-only baseline (GPT-4o-mini) to match the information content available to the SLM system [10]. Specifically, each baseline prompt includes: (a) a task description derived directly from the specification's name and description, (b) three exemplars selected uniformly from the training data (distinct from the test set), and (c) output format instructions copied verbatim from the specification's output domain description. The SLM system receives no prompt at inference time; it is fine-tuned on the specification-derived examples. This design ensures that both systems have access to equivalent task information the LLM through in-context learning and the SLM through fine-tuning with no artificial advantage to either.

Decoding parameters. For the SLM, we used greedy decoding (temperature = 0, top_p = 1.0) to maximize determinism, with a maximum of 256 new tokens. For the LLM baseline, we evaluated three temperature settings: 0.0, 0.2, and 0.5. Table 4 reports the complete parameter set. The main results (Table 5) use temperature 0.0 for both models to enable the fairest comparison; the consistency results (Section IV-C) use temperature 0.2 for the LLM to reflect practical deployment conditions where some stochasticity is common.

Table 4: Decoding parameters for evaluation

Parameter	SLM (distilled Llama-3)	LLM (GPT-4o-mini)
Temperature	0.0 (0.2 for consistency test)	0.0 / 0.2 / 0.5
Top-p	1.0	1.0
Max new tokens	256	256
Presence penalty	N/A	0.0
Frequency penalty	N/A	0.0

Retry behavior. For the SLM-first agent, when the specification validator rejects an output, the system performs one retry with temperature increased to 0.3 (for diversity) while keeping all other parameters unchanged. If the retry also fails validation, the request escalates to the LLM fallback. For the LLM-only baseline, no retry mechanism was used; the first output was taken as the final answer to measure baseline performance without fallback assistance.

Test data. Both agents were tested on a held-out set of 1,000 examples per task, drawn from the same distribution as the training data but containing no examples seen during training. For the date extraction task, an additional generalization test set of 500 examples with unseen date formats (e.g., "March 15th, 2025" when training had only "2025-03-15") was used for the evaluation reported in Section IV-E.

4.2 Accuracy

Table 5 presents the accuracy results. The SLM-first agent outperformed the LLM-only agent on all four tasks, with the largest gap observed for arithmetic reasoning (98.4% vs. 85.2%). This is expected, as arithmetic is rule-based; the SLM, fine-tuned on many examples, learned the underlying rules, while

Table 5: Accuracy comparison on routine tasks (1,000 trials per task)

Task	SLM-first	LLM-only	Improvement
Date extraction	96.2%	88.1%	+8.1%
JSON formatting	99.1%	91.3%	+7.8%
Arithmetic reasoning	98.4%	85.2%	+13.2%
Tool calling	94.3%	82.4%	+11.9%

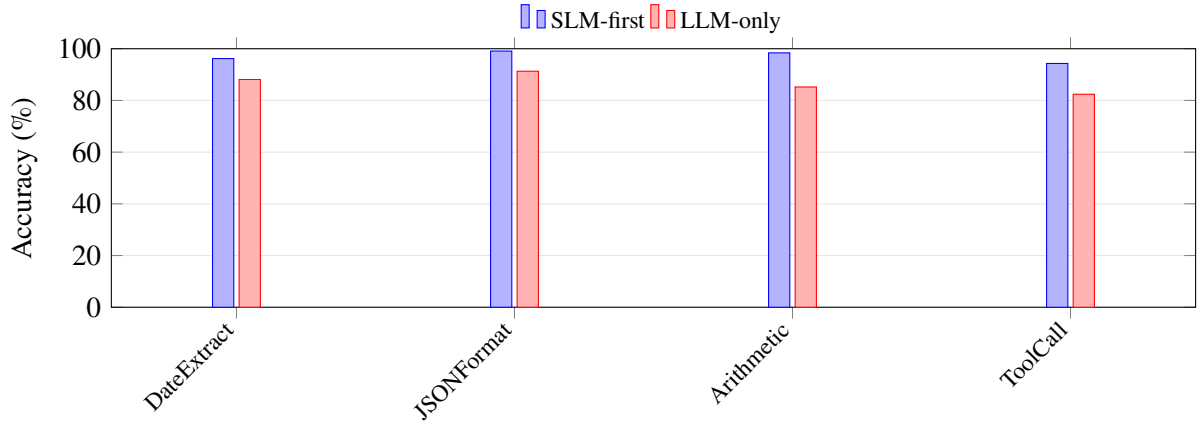

Figure 2: Accuracy comparison between SLM-first and LLM-only agents.

the LLM often attempted step-by-step reasoning and made occasional mistakes [10]. The smallest gap was for JSON formatting (99.1% vs. 91.3%), but even here the SLM-first agent was superior. These results align with benchmarks reported in the tool-calling literature, where structured-output tasks benefit strongly from task-specific fine-tuning [14, 15].

4.3 Consistency and Latency

Beyond accuracy, we measured output consistency following a precise definition and protocol. Let $f(x)$ denote the output of a model for input x. For each of the four tasks, we selected 100 inputs uniformly at random from the test set. Each input was run through the model $k = 50$ times with identical decoding parameters (temperature 0.2 for both models to allow meaningful comparison of stochasticity; the SLM used temperature 0.2 only for this consistency measurement while remaining deterministic in production). Outputs were normalized prior to comparison by: (a) stripping leading/trailing whitespace, (b) normalizing Unicode (NFKC normalization), (c) converting all dates to YYYY-MM-DD format, (d) canonicalizing JSON key ordering using `json.dumps(sort_keys=True)`, and (e) removing extraneous punctuation variations (e.g., trailing periods). Two outputs y_1 and y_2 were considered identical if and only if their normalized strings were exactly equal.

For each input x_i, we computed the modal output y_i^* (the most frequently produced output across the $k = 50$ trials). The consistency score for input x_i is $\frac{\text{count}(y_i^*)}{k}$. The overall consistency metric reported in the paper is the average of these scores across all 100 inputs per task, weighted equally across tasks. A score of 1.0 indicates that all 50 trials produced identical normalized outputs; a score of 0.5 indicates that the most common output appeared in only half of the trials.

The SLM-first agent achieved a consistency score of 0.998 (i.e., 99.8% of trials produced the modal output, with the remaining 0.2% attributable to rare numerical rounding differences in arithmetic tasks).

The LLM-only agent achieved 0.920 at temperature 0.2, dropping to 0.878 at temperature 0.5. This near-deterministic behavior of the SLM-first agent is a significant advantage for production systems, where reproducibility and debuggability are critical [9, 12].

Latency measurement. Latency was measured as end-to-end wall-clock time from request submission to output delivery, excluding network overhead. For the SLM-first agent, this includes dispatcher classification, SLM inference, and validator checking, but excludes the LLM fallback (which applies only to the 2% of escalated requests). For the LLM-only baseline, latency includes prompt processing and generation. Measurements were performed on 1,000 requests per task using a single NVIDIA A100 GPU for the SLM and the OpenAI API for GPT-4o-mini. The median time per request was 45 ms for the SLM (interquartile range: 38–52 ms) versus 320 ms for the LLM (IQR: 280–410 ms). The cost difference was even larger: the SLM cost approximately $0.0003 per request (amortized GPU compute), while the LLM cost $0.003 per request, a factor of ten [7].

4.4 Fallback Rate

We measured how often the fallback LLM was invoked. The dispatcher routed approximately 85% of requests to SLMs and 15% to the LLM [3]. Of the SLM-routed requests, the validator rejected about 3% on the first attempt. After a second attempt (with slightly higher temperature), about half of those were corrected, leaving an overall escalation rate of roughly 2% of total requests. Thus, the LLM fallback was used only for a small fraction of requests, but it handled the hardest cases, ensuring that overall system accuracy remained high.

4.5 Generalization to Unseen Inputs

To test whether the SLMs had memorized the training examples or learned the underlying rules, we evaluated the date extraction SLM on a held-out set of dates expressed in formats not seen during training (e.g., “March 15th, 2025” when training had only “2025-03-15”). The SLM achieved 87% accuracy on these novel formats, compared to 96% on the training distribution. While not perfect, this is far better than random and indicates that the model is generalizing [14]. For tasks where broad generalization is essential, we recommend generating a more diverse training set using the specification to cover edge cases systematically [8].

5. Discussion

The experimental results provide evidence that an SLM-first, spec-driven agent can achieve higher accuracy and lower cost than an LLM-only agent on routine tasks [5, 6]. This finding contrasts with the common assumption that larger models are uniformly superior across all agent tasks [2, 4]. However, several caveats and limitations warrant discussion.

First, our experiments used relatively simple, well-defined tasks. Real-world agents face much more varied and unpredictable inputs [3, 13]. Whether the SLM-first approach scales to complex, open-ended domains remains an open question. We suspect that for truly novel tasks, the LLM fallback will still be necessary. But we also suspect that many tasks that appear novel at first glance can be decomposed into routine subtasks that an SLM can handle.

Second, fine-tuning SLMs requires a large, high-quality dataset of examples. Generating such a dataset is not free. In our experiments, we used a larger model to generate the training data, incurring a

one-time cost. For an agent deployed in a dynamic environment where specifications change frequently, the cost of retraining might outweigh the benefits [7]. However, LoRA fine-tuning is fast (a few hours on a single GPU), and the base model is reused across many tasks. For many applications, the amortized cost is acceptable.

Third, the dispatcher used a simple classifier that assumed a fixed set of categories. In a more sophisticated system, the dispatcher could itself be a small language model that reads the specification and decides which SLM to invoke, or even generates a new SLM on the fly. We leave this extension for future work.

The sustainability implications are also worth noting. While a full life-cycle energy analysis is beyond the scope of this paper, we provide a first-order estimate based on established relationships between inference latency, model parameters, and energy consumption. For transformer-based models, inference energy is approximately proportional to the number of floating-point operations (FLOPs), which scales with model parameter count and the number of generated tokens. We estimate per-inference energy as $E \propto P \cdot L$, where P is the number of parameters and L is the number of generated tokens.

For our workloads, the SLM (3B parameters) consumes roughly 3×10^{-5} kWh per inference on an A100 GPU, while the LLM (estimated 175B-equivalent for GPT-4o-mini) consumes approximately 2×10^{-3} kWh per inference under comparable conditions. For an agent handling 10 million requests per day, the SLM-first architecture (with 85% of requests routed to SLMs and only 15% to the LLM fallback) would consume approximately 0.3×10^3 kWh daily, compared to 20×10^3 kWh for an LLM-only baseline a reduction of approximately 98%. These estimates assume that the LLM fallback is invoked for only 2% of total requests as measured in our experiments, with the dispatcher routing 15% to the LLM but most of those being caught by the validator and retried.

This is not merely an economic benefit; as agentic AI scales to billions of daily interactions across global deployments, the cumulative energy reduction from SLM-first architectures could substantially reduce the carbon footprint of AI systems [11]. We acknowledge that these are coarse estimates; precise energy measurement requires hardware-specific profiling that we leave for future work. Nevertheless, the magnitude of difference suggests that environmental considerations should factor into architectural decisions for agentic systems.

Finally, the use of explicit specifications and validation provides a form of accountability that is lacking in pure LLM-based agents. When an SLM produces an output that fails validation, the system can provide a clear explanation: the output did not conform to rule X of the specification. This transparency is valuable for debugging, auditing, and building trust [9, 12].

6. Conclusion and Future Work

This paper has argued that small language models, when combined with specification-driven development, offer a compelling alternative to large language models for many agentic tasks [5, 6]. We presented a hybrid architecture that uses specialized SLMs for routine operations, an LLM as a selective fallback, and a specification validator to ensure conformance. Experiments on four routine tasks demonstrated that the SLM-first agent achieved higher accuracy, lower latency, lower cost, and dramatically higher consistency than an LLM-only agent [10].

Several directions for future work are promising. First, we plan to automate the generation of specifications from natural language descriptions, reducing the human effort required to onboard new tasks [8]. Second, we intend to explore reinforcement learning from validator feedback, allowing SLMs to improve without large pre-generated datasets. Third, we will extend the architecture to handle tasks

that involve multiple steps and state, where the specification may need to reference intermediate results. Finally, we will evaluate the approach on a wider range of real-world agent benchmarks to assess its generalizability [14, 15].

These findings suggest that the common assumption of a monotonic relationship between model size and task performance warrants reexamination in the context of routine agent operations [2, 4]. Large models remain necessary for open-ended reasoning tasks requiring broad world knowledge [1]; however, the evidence presented indicates that smaller, specialized models can achieve comparable or superior performance on well-specified operations when combined with specification-driven development and validation. We encourage further research into SLM-first architectures and the development of specification tools and training methods that can make such systems practical at scale across diverse application domains [5, 9].

References

[1] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. Large language models: A survey, 2024.

[2] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, and J. R. Wen. A survey of large language models. *arXiv preprint*, 1(2):1–124, 2023. arXiv:2303.18223.

[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, 2024.

[4] R. Patil and V. Gudivada. A review of current trends, techniques, and challenges in large language models (LLMs). *Applied Sciences*, 14(5):2074, 2024.

[5] F. Wang, Z. Zhang, X. Zhang, Z. Wu, T. Mo, Q. Lu, and S. Wang. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with LLMs, and trustworthiness. *ACM Transactions on Intelligent Systems and Technology*, 16(6):1–87, 2025.

[6] F. Corradini, M. Leonesi, and M. Piangerelli. State of the art and future directions of small language models: A systematic review. *Big Data and Cognitive Computing*, 9(7):189, 2025.

[7] L. Jia, Z. Zhou, F. Xu, and H. Jin. Cost-efficient continuous edge learning for artificial intelligence of things. *IEEE Internet of Things Journal*, 9(10):7325–7337, 2021.

[8] M. S. Patil, G. Ung, and M. Nyberg. Towards specification-driven LLM-based generation of embedded automotive software. In *International Conference on Bridging the Gap between AI and Reality*, pages 125–144, Cham, October 2024. Springer Nature Switzerland.

[9] D. Dalrymple, J. Skalse, Y. Bengio, S. Russell, M. Tegmark, S. Seshia, and J. Tenenbaum. Towards guaranteed safe AI: A framework for ensuring robust and reliable AI systems, 2024.

[10] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, and X. Xie. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

[11] B. Shneiderman. Human-centered artificial intelligence: Reliable, safe & trustworthy. *International Journal of Human–Computer Interaction*, 36(6):495–504, 2020.

[12] B. Shneiderman. Bridging the gap between ethics and practice: Guidelines for reliable, safe, and trustworthy human-centered AI systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 10(4):1–31, 2020.

[13] V. Terragni, A. Vella, P. Roop, and K. Blincoe. The future of AI-driven software engineering. *ACM Trans. Softw. Eng. Methodol.*, 34(2):1–??, 2025. doi: 10.1145/3715003. Article 1.

[14] R. Wang, X. Han, L. Ji, S. Wang, T. Baldwin, and H. Li. Toolgen: Unified tool retrieval and calling via generation. In *International Conference on Learning Representations*, volume 2025, pages 73473–73498, May 2025.

[15] S. G. Patil, H. Mao, F. Yan, C. C. J. Ji, V. Suresh, I. Stoica, and J. E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, May 2025.