Rethinking Scale: Deployment Trade-offs of Small Language Models under Agent Paradigms

Xinlin Wang
Proximus Luxembourg S.A.
18 rue du Puits Romain
L-8070 Bertrange, Luxembourg
xinlin.wang.stats@gmail.com

Abstract

Despite the impressive capabilities of large language models, their substantial computational costs, latency, and privacy risks hinder their widespread deployment in real-world applications. Small Language Models (SLMs) with fewer than 10 billion parameters present a promising alternative; however, their inherent limitations in knowledge and reasoning curtail their effectiveness. Existing research primarily focuses on enhancing SLMs through scaling laws or fine-tuning strategies while overlooking the potential of using agent paradigms, such as tool use and multi-agent collaboration, to systematically compensate for the inherent weaknesses of small models. To address this gap, this paper presents the first large-scale, comprehensive study of <10B open-source models under three paradigms: (1) the base model, (2) a single agent equipped with tools, and (3) a routing-based multi-agent system with collaborative capabilities. Our results show that structured agent frameworks (combining step-by-step reasoning and tool use) substantially improve effectiveness over direct prompting, with single-agent systems achieving the best balance between performance and cost. In contrast, routing-based multi-agent setups introduce additional coordination overhead with limited gains under small-model constraints. Our findings highlight the importance of agent-centric design for efficient and trustworthy deployment in resource-constrained settings.

1 Introduction

In recent years, Large Language Models (LLMs) have demonstrated strong capabilities in reasoning, knowledge-intensive tasks, and complex decision-making (Shen et al., 2024), leading to growing adoption in financial applications such as compliance analysis, report summarization, sentiment monitoring, and trading signal extraction (European Parliament et al., 2025). However, financial services operate under strict privacy regulations, including GDPR and PCI DSS, which restrict the use of third-party cloud APIs for sensitive data. This creates a deployment dilemma: high-performing closed-source models cannot be freely used, while private large-scale infrastructure is prohibitively expensive for most institutions (Wang et al., 2025).

Although major financial institutions can build private LLM clusters, smaller organizations—such as regional banks, boutique funds, and individual investors—lack the resources to deploy large models locally. In practice, they are constrained to small language models (SLMs) with fewer than 10 billion parameters. While efficient, these models often struggle with multi-step numerical reasoning, domain-specific knowledge, and financial accuracy. Traditional improvement strategies such as large-scale pre-training or task-specific fine-tuning remain costly and impractical in resource-constrained settings (Haque et al., 2025).

Recent advances in agent-based systems suggest that tool use and structured collaboration may compensate for model limitations (Shen et al., 2024). However, it remains unclear whether such paradigms effectively enhance small models under strict hardware and privacy constraints. Moreover, existing financial benchmarks primarily evaluate task accuracy, overlooking deployment-oriented factors such as energy consumption, latency, and system robustness.

In this work, we shift the focus from model scaling to system design. We evaluate 27 open-source SLMs across three paradigms: base prompting, tool-augmented single-agent systems, and collaborative multi-agent systems. Experiments span 20 financial datasets across 8 task categories, measuring both effectiveness and resource efficiency. We make the following contributions:

- We present the first large-scale empirical study of $\leq$ 10B open-source SLMs in financial settings, systematically comparing base,

single-agent, and routing-based multi-agent paradigms across 27 models.

- We provide the first quantitative analysis of the trade-offs introduced by agent-based designs, showing that while tool augmentation improves effectiveness, multi-agent collaboration incurs significant coordination overhead and instability.
- We provide a practical guide for choosing the best system design based on the available model and specific financial tasks.

Importantly, our study focuses on system-level deployment configurations rather than isolating individual components, such as reasoning strategies or tool usage. In particular, agent paradigms in this work combine structured reasoning (ReAct-style prompting) and tool interaction as a unified framework. Therefore, observed improvements should be interpreted as the effect of the overall system design rather than any single factor in isolation.

2 Related Work and Positioning

Existing evaluations of large language models focus primarily on reasoning ability and output correctness. Representative benchmarks such as the Open LLM Leaderboard and HELM (Chiang et al., 2023; Liang et al., 2022) evaluate single models across diverse tasks but assume stable environments with sufficient computational resources. They emphasize final accuracy while largely overlooking deployment factors such as runtime stability, energy cost, and latency. Consequently, these benchmarks provide limited insight into practical usability under resource-constrained or long-running settings.

In finance, LLMs have been widely applied to text analysis and decision support, including FinGPT and BloombergGPT (Yang et al., 2023; Wu et al., 2023). These approaches typically rely on domain-specific fine-tuning or retrieval augmentation to improve performance. However, most depend on cloud APIs or centralized infrastructure, limiting applicability in localized or privacy-constrained environments.

Agent-based methods offer an alternative path. The Reasoning and Acting (ReAct) framework demonstrates that models can interleave reasoning with tool use to handle complex tasks (Yao et al., 2023), inspiring multi-agent systems such as AutoGen, LangChain, and CrewAI (Wu et al., 2024). Yet existing studies mainly highlight task completion capability rather than runtime stability or efficiency, and typically rely on large models. Systematic analysis of agent behavior under small-model and resource-constrained settings remains limited. While prior work has explored various multi-agent paradigms (e.g., Plan-and-Execute (Wang et al., 2023), Reflexion (Shinn et al., 2023), debate (Du et al., 2023)), these approaches often focus on capability improvements under large models. In contrast, our study evaluates a deployment-oriented, routing-based multi-agent configuration under small-model and resource-constrained settings.

Recent work has examined the efficiency and optimization of small language models, including latency and system-level improvements (Pham et al., 2025; Dettmers et al., 2023; Dao et al., 2022). However, these studies rarely connect deployment analysis with agent paradigms. As a result, we lack a clear understanding of how agent systems behave under realistic operational constraints.

This work addresses this gap by evaluating whether architectural design can compensate for limited model scale under privacy and hardware constraints, using the financial domain as a practical testbed.

3 Methodology and Experimental Design

Our study is designed to systematically evaluate the performance of three distinct paradigms: 1)Base SLMs; 2)Single-Agent Systems (SAS); 3) routing-based Multi-Agent Systems (MAS) across a diverse set of financial tasks. This section details our experimental setup, including the selected models, task formulations, and the architectural designs of the single-agent and multi-agent systems.

3.1 Architecture Designs of Three Paradigms

In this study, we consider three progressively complex paradigms, ranging from direct model use to collaborative agent systems. Fig. 1 shows their differences in control flow, tool access, and coordination structure.

It is important to note that the Base SLM setting uses direct zero-shot prompting without explicit reasoning scaffolding, whereas both SAS and routing-based MAS adopt structured ReAct-style framework (Yao et al., 2022) (see Appendix A), which introduces step-by-step reasoning alongside tool usage. As a result, performance differences

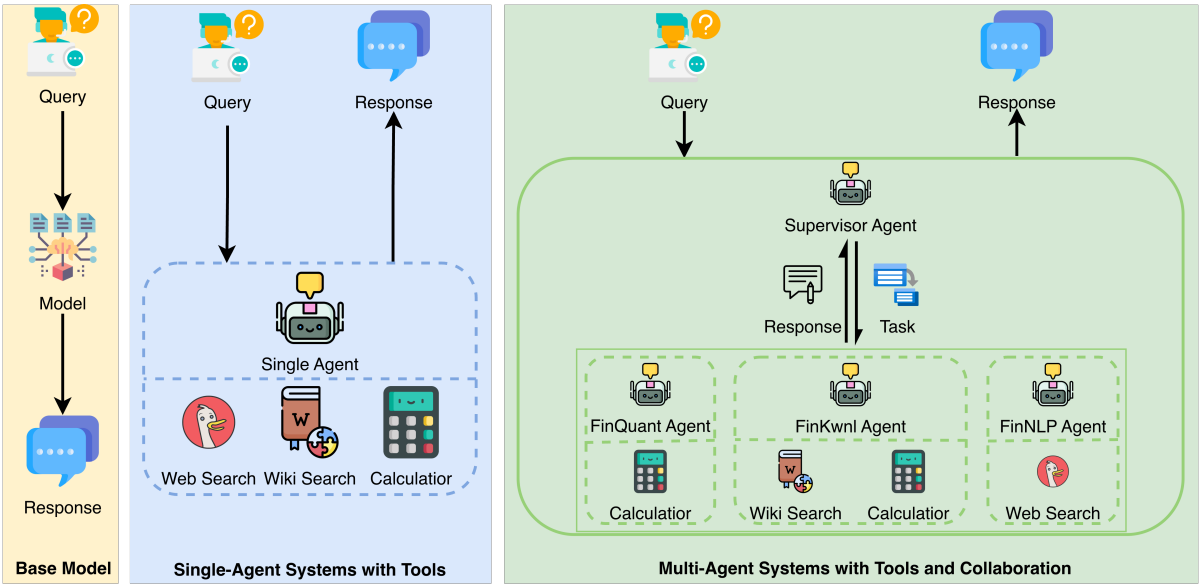

Figure 1: Architecture of three paradigms. The dashed rounded rectangle encloses the agent and the tools it can invoke. The supervisor agent, depicted within a solid rectangle, oversees and invokes the sub-agents contained in the inner solid rounded rectangle.

across paradigms may reflect the combined effect of structured reasoning and tool augmentation, which are not explicitly disentangled in this study. This structure supports step-by-step processing, which is meaningful for exploring financial tasks. Other agentic frameworks are not explored in this study.

3.1.1 Base SLM

Base SLM represents direct deployment without architectural changes. Each model receives the task input and generates the output directly. This setting reflects the raw model capability and serves as a reference for performance, stability, and resource usage.

3.1.2 Single-Agent System

SAS simulates the workflow of an all-around analyst and follows a think–act–observe cycle. After receiving a task, the agent decides whether to use external tools (calculator, wiki search, web search)[1] or answer directly. If tools are used, the agent processes the returned results and may repeat the cycle until producing a final answer. This setup represents a general-purpose financial analyst with tool support.

[1]Wiki search tool can be replaced by a local knowledge base, and web search tool can be replaced by on-premise retrieval systems in privacy-sensitive environments, without affecting the architectural comparison

3.1.3 Routing-based Multi-Agent System

Our MAS corresponds to a routing-based (supervisor–worker) architecture, which prioritizes simplicity and deployability in real-world systems. MAS consists of one supervisor and three specialized agents: FinancialKnowledgeAgent, FinancialNLPAgent, and FinancialQuantAgent. All agents follow the ReAct pattern.

The supervisor routes tasks to the most suitable expert agent but does not use tools. Each expert has limited tool access aligned with its role:

- FinancialNLPAgent: web search
- FinancialKnowledgeAgent: calculator and wiki search
- FinancialQuantAgent: calculator only

This design introduces role specialization and coordination among agents. All agent interactions rely on structured outputs (XML-style tags) and constrained tool invocation formats, which introduce additional execution constraints compared to the Base setting.

3.2 Models, Datasets and Tasks

3.2.1 Model Selection

We evaluate 27 open-source language models with fewer than 10 billion parameters. These models 1 span several widely used families, including Qwen, Gemma, LLaMA, Phi, DeepSeek, Mistral, and Solar, reflecting a diverse range of architectural and training choices commonly considered for local deployment.

Model	Version	Range
Qwen	2.5-0.5B, 2.5-1.5B, 2.5-3B, 2.5-7B, 3-8B	0.5B-8B
Llama	3.2-1B, 3.2-3B, 2-7B, 3.1-8B, 3-8B	1B-8B
Gemma	3-270M, 3-1B, 2-2B, 3-4B, 2-9B	0.27B-9B
Phi	3.5-mini, 4-mini, 3-small	3.8B-7B
DeepSeek-R1	Distill-Qwen-1.5B/7B, Distill-Llama-8B	1.5B-8B
Mistral	7B-v0.3, Ministral-8B	7B-8B
SOLAR	10.7B-v1.0	10.7B

Table 1: Model family, version, and parameter range of evaluated models.

3.2.2 Datasets, Tasks and Evaluation Metrics

Experiments are conducted on 20 publicly available financial datasets covering eight task categories: sentiment analysis, text classification, named entity recognition, question answering, stock movement prediction, credit scoring, summarization, and bankruptcy prediction. These tasks collectively reflect document-level reasoning, numerical understanding, and decision-making within financial contexts.

All datasets provide ground-truth labels for automatic evaluation. A complete dataset summary is provided in Appendix B. We prioritize consistency across datasets rather than dataset-level optimization; therefore, we sample 50 instances from each dataset.

3.3 Evaluation Metrics for Deployment Reality

To evaluate agent paradigms under realistic deployment constraints, we adopt a set of metrics that jointly capture effectiveness, efficiency, and robustness. Standard task-specific metrics are used where appropriate, but they are not redefined here due to space constraints. Below, we describe the metrics defined for this study.

Completion Rate. Completion Rate measures the robustness of a system in practical deployment settings. We define it as the proportion of samples for which the system returns a valid response without runtime errors, timeouts, or malformed outputs:

$$\text{Completion Rate} = \frac{\#\text{ of successful responses}}{\#\text{ of total samples}}.$$

This metric reflects the reliability of a paradigm beyond its nominal task performance.

Average Latency. Average Latency is defined as the mean end-to-end inference time required to process a single input sample, including all intermediate reasoning and agent interactions. This metric captures user-perceived responsiveness and is critical for latency-sensitive applications.

Normalized Response Quality (NRQ) To aggregate response quality across heterogeneous tasks and evaluation metrics, we define NRQ as a relative, architecture-level measure (see Appendix C). NRQ captures the overall effectiveness of an architecture while remaining invariant to the scale and type of task-specific evaluation metrics. For each dataset, response quality is expressed as the normalized improvement over the Base SLM, with the direction adjusted according to whether the underlying metric is higher-is-better or lower-is-better.

Composite Effectiveness Score. Due to the differences in evaluation metrics among datasets, to enable cross-dataset comparison, we compute a standardized composite score Z_c for each model-architecture combination.

$$Z_c = \frac{1}{N}\sum_{i=1}^{N}\frac{X_i - \mu_i}{\sigma_i}$$

where x denotes the raw metric value, and μ and σ denote the mean and standard deviation computed across datasets for the same model. The composite effectiveness Z-score is then obtained by averaging the Z-scores across all available dataset-specific metrics. A higher Z_c indicates better overall effectiveness relative to other configurations.

Leading Advantage. To quantify the decisiveness of performance differences, we define leading advantage as the relative gap between the best-performing and second-best-performing architectures :

$$\alpha = \frac{s_{\text{best}} - s_{\text{second}}}{|s_{\text{second}}| + \epsilon} \times 100\%, \quad \epsilon = 10^{-8}$$

where ϵ is a small constant used to avoid division by zero. This metric highlights whether observed improvements represent marginal gains or substantial performance advantages.

3.4 Implementation

All experiments were conducted in a unified environment to ensure comparability. We implement the experiments on NVIDIA H100 80GB GPU, CUDA 12.1, to ensure finish the experiments

quickly. Regarding the inference parameters setting, temperature was set to 0 to reduce randomness and enhance reproducibility, with top_p=0.9. The maximum interaction turns of agentic architectures were capped at 5 considering the time efficiency. We used vLLM for high-throughput, memory-efficient inference on HPC.

4 Results and Analysis

4.1 Overall System Trade-off

Table 2 summarizes architecture-level performance across effectiveness, efficiency, and stability, revealing a clear trade-off as system complexity increases.

In terms of effectiveness, SAS achieves the highest NRQ (4.85), showing clear improvement over the Base SLM. MAS provides only limited additional gains (NRQ = 0.36). However, reliability declines with complexity. The Base SLM maintains a near-perfect completion rate (99.67%), which drops to 79.92% for SAS and 72.01% for MAS. This suggests that more complex architectures introduce instability during execution.

Regarding efficiency, energy per token decreases from 1.83 (Base) to 1.03 (SAS) and 0.53 (MAS), a 71% reduction from Base to MAS. Tokens per second also increase (190 $\rightarrow$ 345 $\rightarrow$ 642). However, average latency nearly doubles for SAS and MAS (about 23 seconds) compared to the Base SLM (12.39 seconds). Token usage rises sharply from 2,356 tokens per sample (Base) to 8,040 (SAS) and 14,618 (MAS), reflecting coordination and multi-step reasoning overhead.

Tool usage behavior plays an important role in these results. We observe that smaller models (e.g., <3B parameters) often struggle to generate valid tool invocation formats, leading to malformed outputs and reduced completion rates in SAS and MAS. In contrast, larger models demonstrate more stable structured reasoning and tool interaction, which partially explains the improved effectiveness of SAS in higher-capacity regimes.

4.2 Efficiency–Effectiveness Trade-off

To evaluate practical utility, we analyze the trade-off between computational efficiency (Energy/T, log scale) and effectiveness (composite effectiveness score). As illustrated in Fig. 2, our empirical results reveal a clear divergence in the Pareto frontier across the Base, SAS, and MAS. We observe that many energy-efficient models achieve very low effectiveness, forming a trivial Pareto region that is practically irrelevant. We therefore focus on the high-effectiveness region of the Pareto frontier.

SAS (green squares) consistently occupies the upper even the upper-left quadrant of the trade-off space, achieving strong effectiveness (mostly $z > 0.4$) at relatively low energy levels. This indicates that single-agent design improves performance without large coordination cost.

In contrast, Base SLMs (blue circles) show an efficiency trap at larger scales. While smaller models perform moderately, 8B–10B variants move toward the lower-right region, combining high energy use with lower effectiveness ($z < -0.5$). This suggests that scaling parameter volume in isolation may yield diminishing returns. MAS (red triangles) shows higher variance. It can reach high effectiveness but often requires more energy due to coordination overhead, and smaller MAS setups sometimes perform worse than SAS.

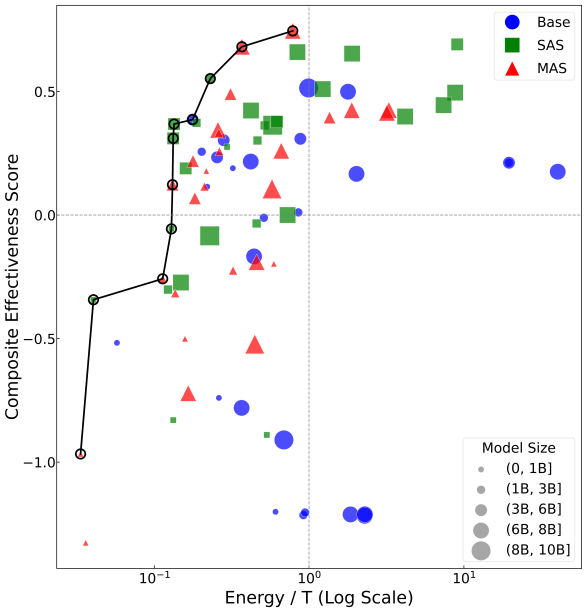

Figure 2: Efficiency-Effectiveness trade-off across model-architecture settings. Each point represents one model under one architecture. Different marker shapes indicate different architectures, and marker size reflects model parameter size. We additionally highlight the Pareto frontier to guide interpretation of optimal trade-offs.

4.3 Task–Architecture Adaptation

Beyond global metrics, we analyze how Base (B), SAS (S), and MAS (M) perform across different tasks (Fig. 3). Results show that no single paradigm dominates across all settings; the best choice depends on both the model family and task type.

MAS performs best on bankruptcy prediction

Architecture	Comp. Rate↑	NRQ↑	Energy/T(mJ)↓	Avg. Latency(s)↓	T/s↑	T/Sample↓
Base	**99.67%**	0.00	1.83	**12.39**	190.08	**2356.06**
SAS	79.92%	**4.85**	1.02	23.27	345.42	8039.75
MAS	72.01%	0.36	**0.52**	22.76	**642.34**	14618.22

Table 2: Overall architecture-level comparison across effectiveness, efficiency, and system stability. Arrows indicate optimization direction (↑ higher is better; ↓ lower is better). Comp. Rate: Completion Rate; NRQ: normalized response quality; Energy/T: energy per token; Avg. Latency: average latency; T/s: tokens per second; T/Sample: token per sample.

across most model scales, though often with a small margin. This suggests that multi-agent coordination can benefit high-risk financial tasks, despite its higher cost. In contrast, SAS works well for reasoning and generation tasks such as question answering, summarization, and credit risk prediction. For example, in Qwen2.5-0.5B and Llama-2-7b, SAS achieves over ($> 150\%$) improvement compared to other setups, showing that single-agent design can strengthen mid-sized models.

Interestingly, Base SLMs remain competitive for classification and token-level tasks, including named entity recognition, sentiment analysis, and stock prediction, especially in the Gemma and Phi families. For gemma-3-270m, the Base SLM outperforms agent paradigms by a large margin on most tasks. This is because the agent paradigms of this model fail to run on most tasks.

4.4 Failure Mode Analysis

To compare direct inference and agentic orchestration, we analyze failure modes in Base, SAS, and MAS. Fig. 4 shows that agentic systems (SAS and MAS) have more errors than the Base SLM. This is because the Base SLM either produces an output or times out, while SAS and MAS use multi-turn reasoning and tools, which increases system-level failures.

ReAct in SAS and MAS shifts failures toward structural and budget issues. In SAS, *Context Length Exceeded* and *No Delegation dominate*, as long prompts and iterative reasoning exhaust the context window. In MAS, inter-agent dynamics cause *Delegation Failure* and *No Delegation*. While MAS avoids *Timeout* and *Output Protocol Violations*, agents can get stuck in failed hand-offs or fail to start delegation, offsetting the benefits of the multi-agent setup.

A substantial portion of failures in SAS and MAS can be attributed to invalid or unsuccessful tool invocation, including malformed structured outputs and incorrect tool selection. This highlights a key challenge for small models when operating under structured agent frameworks.

5 Discussion and Practical Implications

5.1 Discussion

Our results challenge the prevailing "scaling law" hypothesis (Kaplan et al., 2020), which posits that performance is a monotonic function of parameter count and computation. The observed "U-turn" in Base SLMs (Fig. 2) reveals an optimization bottleneck where increased model capacity without an agent paradigm yields diminishing, and eventually negative, returns. However, the transition to agentic architectures introduces a fundamental coordination tax (Zhang et al., 2024; Rizvi-Martel et al., 2025). While Single-Agent Systems offer a Pareto-optimal balance by concentrating reasoning capabilities with moderate energy costs, routing-based multi-agent systems suffer from the energy expenditure of inter-agent communication, and iterative ReAct prompting in this setting does not consistently translate into proportional gains in correctness.

This decoupling of efficiency and effectiveness highlights a critical form of architectural fragility. By wrapping models in complex agent paradigms, we trade the unpredictability of raw model outputs for the complexity of managing agent states. The high incidence of *Delegation Failure* and *Context Length Exceeded* in our failure analysis suggests that the bottleneck for modern AI is no longer raw generative capability but rather context management and instruction adherence. Consequently, increasing agent counts often introduces new vectors for systemic failure, such as infinite delegation loops, rather than enhancing collective intelligence.

In this study, we do not explicitly disentangle the effect of structured reasoning from tool usage. Since SAS and MAS employ ReAct-style prompting with explicit reasoning steps, part of the observed performance gain may stem from improved reasoning rather than tool interaction itself.

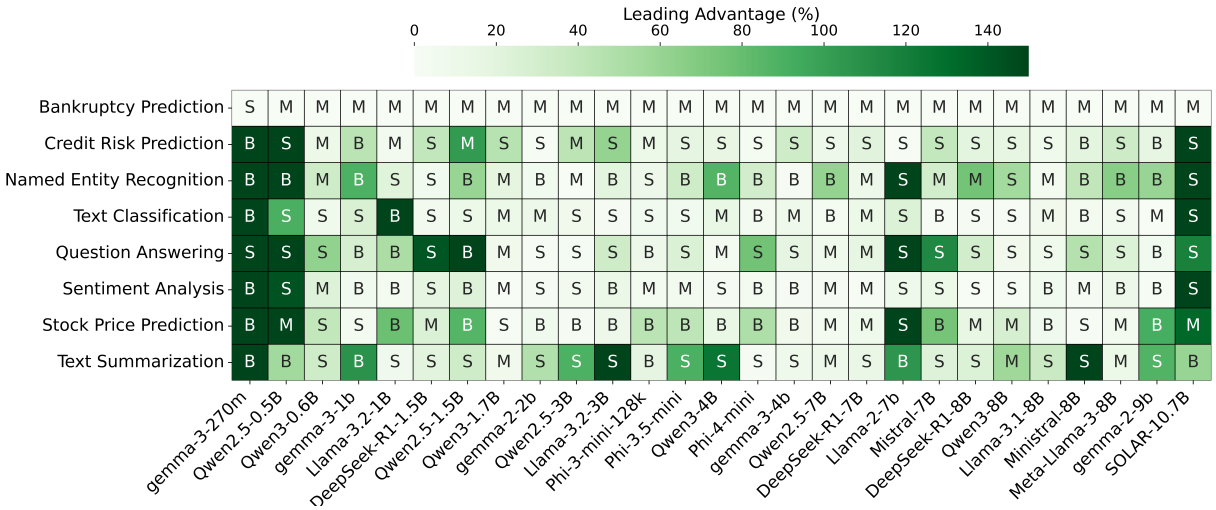

Figure 3: Heatmap of task-architecture adaption. B denotes Base SLM; S denotes SAS; M denotes MAS. The letter in each cell means that setting performs best. The background color of each cell shows how large the advantage is. Darker colors indicate a larger performance gap between the best and second-best options.

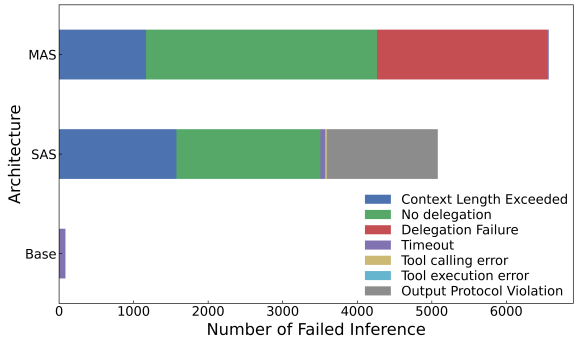

Figure 4: Distribution of failure modes for Base, SAS and MAS

5.2 Implications for Industry Deployment

From a practical view, these findings show that we should stop thinking that "bigger is always better." Instead, we should choose the design that fits the specific task.

First, we need to choose the paradigm based on what it does best. SAS is the best choice for complex creative tasks because it works very well and saves energy. While the high variance of routing-based MAS suggests it may be more suitable for high-entropy domains, such as financial forecasting, where the statistical redundancy of multiple perspectives outweighs the coordination costs. For simpler extraction tasks like named entity recognition, the Base remains superior, as agentic overhead tends to dilute precise token-level mappings.

Second, these systems break easily, so we need to build them carefully. Builders should make backup plans. If the system gets stuck or runs out of space, it should switch back to the basic model to ensure it still provides an answer.

5.3 Limitations and Future Work

This study covers a fixed set of small language models, tasks, and agent designs. Findings may not generalize to other domains with different reasoning or interaction patterns. The agent systems use simple coordination strategies, however, the stronger control mechanisms may improve stability. Adaptive agents that change behavior over time are not included. Future work could study dynamic control, better delegation strategies, and system behavior under real user traffic.

Acknowledgments

This research was funded by the Luxembourg National Research Fund (FNR), grant reference NCER22/IS/16570468/NCER-FT. The author would like to acknowledge the support of Proximus Luxembourg S.A.. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

References

Wei-Lin Chiang and 1 others. 2023. Open llm leaderboard. In *NeurIPS Datasets and Benchmarks Track*.

Tri Dao and 1 others. 2022. Flashattention: Fast and memory-efficient exact attention. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Tim Dettmers and 1 others. 2023. Efficient fine-tuning of small language models. In *International Conference on Machine Learning (ICML)*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Committee on Economic European Parliament, Monetary Affairs, and Rapporteur Arba Kokalari. 2025. Report on the impact of artificial intelligence on the financial sector (a10-0225/2025). Adopted 5.11.2025 by the Committee on Economic and Monetary Affairs.

Mohd Ariful Haque, Fahad Rahman, Kishor Datta Gupta, Khalil Shujaee, and Roy George. 2025. Tinyllm: Evaluation and optimization of small language models for agentic tasks on edge devices. *Preprint*, arXiv:2511.22138.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Percy Liang, Rishi Bommasani, Tony Lee, and 1 others. 2022. Holistic evaluation of language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Nghiem Thanh Pham, Tung Kieu, Duc Manh Nguyen, Son Ha Xuan, Nghia Duong-Trung, and Danh Le-Phuoc. 2025. SLM-bench: A comprehensive benchmark of small language models on environmental impacts. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 21369–21392, Suzhou, China. Association for Computational Linguistics.

Michael Rizvi-Martel, Satwik Bhattamishra, Neil Rathi, Guillaume Rabusseau, and Michael Hahn. 2025. Benefits and limitations of communication in multi-agent reasoning. *Preprint*, arXiv:2510.13903.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024. Small LLMs are weak tool learners: A multi-LLM agent. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16658–16680, Miami, Florida, USA. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv.org/abs/2303.11366*, 1.

Bo Wang, Weiyi He, Shenglai Zeng, Zhen Xiang, Yue Xing, Jiliang Tang, and Pengfei He. 2025. Unveiling privacy risks in LLM agent memory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25241–25260, Vienna, Austria. 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. *arXiv preprint arXiv:2305.04091*.

Qingyun Wu and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *International Conference on Learning Representations (ICLR)*.

Shijie Wu, Ozan Irsoy, and 1 others. 2023. Bloomberggpt: A large language model for finance. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Qianqian Xie, Weiguang Han, Zhengyu Chen, Ruoyu Xiang, Xiao Zhang, Yueru He, Mengxi Xiao, Dong Li, Yongfu Dai, Duanyu Feng, Yijing Xu, Haoqiang Kang, Ziyan Kuang, Chenhan Yuan, Kailai Yang, Zheheng Luo, Tianlin Zhang, Zhiwei Liu, Guojun Xiong, and 15 others. 2024. The finben: An holistic financial benchmark for large language models. *Preprint*, arXiv:2402.12659.

Qianqian Xie, Weiguang Han, Xiao Zhang, Yanzhao Lai, Min Peng, Alejandro Lopez-Lira, and Jimin Huang. 2023. Pixiu: A large language model, instruction data and evaluation benchmark for finance. *Preprint*, arXiv:2306.05443.

Shuo Yang, Yilun Liu, and 1 others. 2023. Fingpt: Open-source financial large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP), Industry Track*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Shunyu Yao, Jeffrey Zhao, and 1 others. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2024. Cut the crap: An economical communication pipeline

for llm-based multi-agent systems. *Preprint*, arXiv:2410.02506.

A Details of Reasoning and Acting Implementation

As illustrated by the think-act-observe cycle 5, upon receiving a task, the agent first enters the thinking phase, autonomously determining whether to invoke tools (calculator, wiki search, or web search) and their invocation sequence. The agent may provide answers directly without tool assistance. Should it identify a tool requirement, it proactively transitions to the action phase to invoke external tools, then proceeds to the observation phase: ingesting tool outputs, validating them against intermediate reasoning, and updating its internal state. This cycle (think-act-observe) may iterate multiple times until the model accumulates sufficient empirical information to synthesize the final answer.

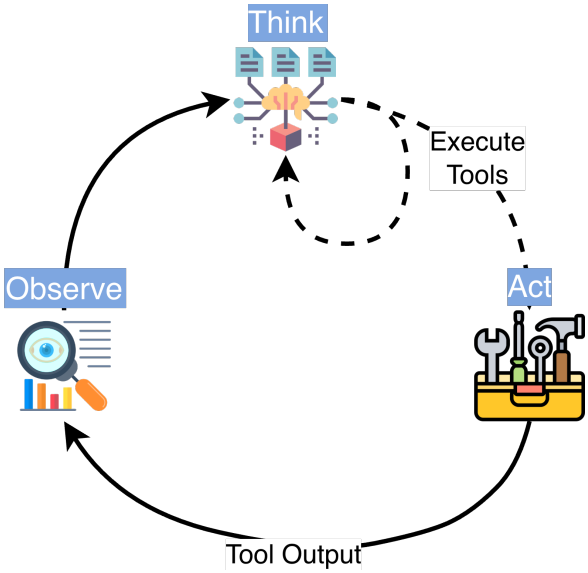

Figure 5: Structure of Reasoning and act

B Summary of Dataset

Our evaluation spans eight representative financial NLP tasks, selected to cover a broad range of linguistic, analytical, and decision-making challenges encountered in real-world financial applications. These tasks include sentiment analysis, text classification, named entity recognition, question answering, stock movement prediction, credit scoring, and two financial summarization tasks. Together, they reflect both document-level and sentence-level reasoning, factual grounding, numerical understanding, and predictive modeling.

To ensure a comprehensive assessment, we evaluate performance across 20 publicly available datasets as shown in Table 3, each widely used in the financial NLP community. The datasets vary in size—from small expert-annotated corpora to large-scale benchmark collections—and target distinct aspects of financial language understanding, such as investor sentiment (FiQA SA, Financial PhraseBank), numerical table reasoning (FinQA, TATQA), risk classification (German, Australian, LendingClub), and multi-document summarization (ECTSum, EDTSum).

C Formal Definition of Normalized Response Quality(NRQ)

This section provides the formal definition of the *Normalized Response Quality* metric used in the main paper. NRQ is designed to aggregate response quality across heterogeneous tasks and datasets with different evaluation metrics while isolating the effect of architectural design choices. There are mainly three steps to achieve the final overall NRQ as shown in this section.

C.1 Dataset-level Relative Response Quality Gain

Let d denote a dataset and $a \in \mathcal{A}$ denote an architecture. We use $s_{a,d}$ to represent the task-specific evaluation score obtained by architecture a on dataset d. Each dataset is associated with a flag indicating whether its evaluation metric is *higher-is-better* (e.g., Accuracy, F1, ROUGE, Exact Match) or *lower-is-better* (e.g., error rate, mean absolute error).

We designate the base architecture as an anchor and compute the relative response quality gain of architecture a on dataset d as:

$$g_{a,d} = \begin{cases} \dfrac{s_{a,d} - s_{\text{Base},d}}{|s_{\text{Base},d}| + \epsilon}, & \text{if higher-is-better,} \\ \dfrac{s_{\text{Base},d} - s_{a,d}}{|s_{\text{Base},d}| + \epsilon}, & \text{if lower-is-better,} \end{cases}$$

where $\epsilon = 10^{-8}$ is a small constant used to avoid division by zero.

By construction, $g_{a,d} > 0$ indicates that architecture a improves response quality relative to the base architecture on dataset d, while $g_{a,d} < 0$ indicates degradation.

Example. Suppose the Base model achieves an accuracy of 0.60 on a dataset, while SAS achieves 0.75. Since accuracy is higher-is-better, the relative response quality gain is computed as: $g = (0.75 - 0.60)/0.60 = 0.25$ indicating a 25% improvement over the base model.

Tasks	Dataset	# of sample	Evaluation Metrics
Sentiment analysis	textbfFiQA SA	235	F1
	Financial PhraseBank	970	F1
	TFNS	2390	F1
	NWGI	4.05k	F1
Text classification	Headline	20.5k	F1
Named entity recognition	NER	3.5k	F1
Question answering	CFA_QA	1032	EM Accuracy
	FinQA	1147	EM Accuracy
	ConvFinQA	1490	EM Accuracy
	TATQA	1668	EM Accuracy
Stock movement prediction	BigData22	1470	Accuracy
	ACL18	3720	Accuracy
	CIKM18	1140	Accuracy
Credit Scoring	German	200	F1
	Australian	139	F1
	LendingClub	2691	F1
Summarization	ECTSum	495	ROUGE-L
	EDTSum	2000	ROUGE-L
Bankruptcy prediction	Polish	235	AUC
	Taiwan	452	AUC

Table 3: Summary of Dataset

C.2 Task-level Aggregation NRQ

To prevent tasks with more datasets from disproportionately influencing the aggregate score, we first aggregate relative response quality gains at the task level. Let $\mathcal{D}_t$ denote the set of datasets associated with task t. The task-level response quality score for architecture a is defined as:

$$G_{a,t} = \frac{1}{|\mathcal{D}_t|} \sum_{d \in \mathcal{D}_t} g_{a,d}.$$

This aggregation captures the average architectural effect within a coherent task category, independent of the number of datasets or the scale of their underlying evaluation metrics.

C.3 Architecture-level Aggregation NRQ

Finally, we compute the overall normalized response quality of architecture a by averaging task-level scores across all tasks. Let $\mathcal{T}$ denote the set of evaluated tasks. The architecture-level NRQ score is defined as:

$$\text{NRQ}(a) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} G_{a,t}.$$

This equal-weighted aggregation ensures that all tasks contribute uniformly to the final score, yielding a single architecture-level measure of response quality that is comparable across heterogeneous tasks, datasets, and evaluation metrics.

D Evaluation Metrics

Energy per Token. Energy per Token quantifies the computational efficiency of a model by measuring the average energy consumption normalized by the number of generated tokens. It captures the marginal energy cost of text generation and is particularly relevant for large-scale or resource-constrained deployments.

Tokens per Sample. Tokens per Sample measures the average number of tokens generated for each input sample. This metric reflects verbosity and computational load, and serves as a proxy for inference cost in token-based billing or throughput-limited systems.

E Prompt Design for Base SLM, Single-Agent and routing-based Multi-Agent Systems

E.1 Base SLM

There is no advance prompt engineering regarding the Base SLMs. The prompts for each query are from the datasets in The FinAI community (Xie et al., 2023, 2024).

E.2 Design Principles for Agent Paradigms

The prompt design enforces deterministic structured outputs, constrained tool invocation, and explicit reasoning control. The SAS evaluates unified reasoning capacity, while the MAS evaluates decomposed coordination under identical operational constraints. All prompts are designed under three principles to ensure stable deployment and fair architectural comparison:

1. **Strict structural control.** All responses must follow predefined XML-style tags (e.g., `<think>`, `<action>`, `<observation>`, `<final_answer>`) to prevent format drift.
2. **Explicit tool governance.** Tool usage is constrained by hard formatting rules to avoid hallucinated tools and malformed JSON calls.
3. **Task-grounded reasoning.** The model must first identify the required answer format (e.g., "yes/no", "A/B/C") before reasoning.

The Single-Agent System (SAS) and Multi-Agent System (MAS) differ only in architectural decomposition, while maintaining identical tools and structural constraints.

E.3 Single-Agent System (SAS)

The SAS defines a unified financial intelligence agent specialized in analyzing financial data and answering questions including financial NLP tasks (sentiment, NER, classification, summarization), conceptual finance questions, numerical reasoning, credit scoring and bankruptcy prediction, stock movement prediction.

E.3.1 Structured Workflow

The SAS follows a ReAct-style protocol. The agent iteratively performs reasoning and tool usage until a final answer is produced.

- `<question>`: contains the input task
- `<think>`: model reasoning and decision process
- `<action>`: optional tool invocation
- `<observation>`: tool execution results
- `<final_answer>`: final output

This structure separates reasoning from final output, enforces explicit tool invocation, and ensures that tool results are system-provided rather than hallucinated.

E.3.2 Tool Usage

The agent has access to a fixed set of tools, including a calculator, a wiki search tool, and a web search tool. Tool usage is governed by strict rules:

- Mathematical calculations → calculator
- Concept definitions → wikisearcher
- Latest information → websearcher
- Pure reasoning → no tool

Tool invocation must follow a constrained JSON format within the `<action>` tag, which prevents malformed outputs and hallucinated tools.

E.3.3 Structured Constraints.

To ensure stable deployment, the prompt enforces:

- Strict XML-style output format
- Explicit separation between reasoning and final answers
- Deterministic tool invocation syntax

These constraints are critical for reducing format drift and improving completion reliability in small models.

E.3.4 Minimal SAS Example

A simplified interaction is shown below:

```
<question>Calculate working capital:
   assets=500, liabilities=300</question>
<think>Working capital =
    assets - liabilities</think>
<action>{"name": "calculator",
    "expression": "500 - 300"}</action>
<observation>200</observation>
<final_answer>200</final_answer>
```

This example illustrates how reasoning, tool usage, and final answer generation are explicitly separated.

E.4 Routing-based Multi-Agent System (MAS)

The MAS decomposes reasoning into **Supervisor**, **FinancialKnowledgeAgent**, **FinancialNLPAgent** , and **FinancialQuantAgent**.

E.4.1 Supervisor Protocol

The supervisor is a routing-only controller and never performs task solving.

It follows a strict two-phase ReAct-style protocol:

Phase 1 (No Observation Present).

- Select exactly one specialist agent.
- Delegate using <action>.
- Must not output <final_answer>.

Phase 2 (Observation Present).

- Evaluate the agent result.
- Output the final answer.
- No agent simulation or modification of observation.

Minimal Supervisor Example:

A fully instantiated example is shown below.

```
<question>
Predict whether the company is likely
to face bankruptcy given:
working capital/total assets = 0.10,
retained earnings/total assets = 0.05,
EBIT/total assets = 0.08,
market value of equity/liabilities = 0.40,
sales/total assets = 0.80
</question>

<think>
This is a quantitative bankruptcy
prediction task requiring financial
ratio aggregation. FinancialQuantAgent
is most appropriate.
</think>

<action>
{"agent": "FinancialQuantAgent",
 "task": "Predict bankruptcy:
 WC/Assets=0.10, RE/Assets=0.05,
 EBIT/Assets=0.08, MVE/Liabilities=0.40,
 Sales/Assets=0.80"}
</action>
```

Phase 2: Final Answer Generation

```
<observation>
Z = 1.2*0.10 + 1.4*0.05 + 3.3*0.08
+ 0.6*0.40 + 1.0*0.80 = 1.494
The score is below the threshold of 1.81,
indicating high bankruptcy risk.
</observation>

<think>
The computed Z-score is 1.494, which
falls in the distress zone.
</think>

<final_answer>yes</final_answer>
```

E.4.2 Specialist Agent Design

Each agent operates within a constrained functional scope:

- **FinancialKnowledgeAgent:** CFA, financial concepts, numerical reasoning
- **FinancialNLPAgent:** Sentiment, NER, classification
- **FinancialQuantAgent:** Credit scoring, bankruptcy prediction, stock movement

Each specialist:

- Cannot delegate further
- Uses restricted tools
- Must produce concise <final_answer>

E.5 Design Rationale

The XML-based structure attempts to reduce output format drift, tool hallucination, nested JSON errors, and infinite reasoning loops.

In MAS, the Supervisor handles coordination only, while the Specialists handle domain execution. This separation enables controlled comparison of architectural effects under identical tooling constraints.

Both systems use identical tool definitions, follow identical structured reasoning format, and use the same base models. The only difference is reasoning decomposition (monolithic vs delegated), isolating the architectural effect.