An Experimental Comparison of Multi-Agent and Monolithic Prompt Architectures Across LLM Scales for Hardware Diagnostics

João P. A. Lima[1], Carlos Caminha[1], Victor A. E. Farias[1], João P. P. Gomes[1], Francisco L. F. Pereira[1], Maria de Lourdes M. Silva[1], Javam Machado[1], José S. Andrade Jr.[2]

[1]Laboratório de Sistemas e Bancos de Dados (LSBD)
Departamento de Computação (DC)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil.

{joaopaulo.lima,carlos.caminha,victor.farias}@lsbd.ufc.br

{joao.pordeus,lucas.falcao,malu.maia,javam.machado}@lsbd.ufc.br

[2]Departamento de Física
Universidade Federal do Ceará (UFC), Campus do Pici
Fortaleza, CE – Brasil.

soares@fisica.ufc.br

Abstract. *Monolithic prompting is a common baseline but struggles in complex multi-task settings, especially with smaller models, whereas multi-agent approaches decompose workflows into specialized prompts. In this study, we compare both across 26 open-source LLMs in a hardware diagnostics chatbot, using a benchmark covering guardrails, diagnostic reasoning, and constrained dialogue, with a genetic algorithm-based prompt optimizer to ensure a fair comparison. Results show that multi-agent benefits depend on task complexity and model scale, improving consistency and average performance primarily for smaller models on complex tasks, suggesting that task decomposition can mitigate the limitations of smaller LLMs under controlled conditions.*

1. Introduction

The rapid evolution of Large Language Models (LLMs) has shifted the focus of artificial intelligence from text generation to the development of autonomous agents capable of executing complex, multi-step workflows. Traditionally, these systems have relied on a monolithic prompt architecture, where a single and complex textual instruction drives the model to execute a task. While this architecture is simple to implement, LLMs struggle to reliably follow all the instructions described in a single prompt within a single context window, which can lead to performance degradation or hallucinations, especially for smaller models with reduced context windows.

To address these limitations, recent works (Wu et al. 2023; Liu et al. 2025) are increasingly turning toward modular, Multi-Agent strategies. This architecture aims to enhance overall reliability by breaking a complex system into specialized sub-modules, each directed by its own task-specific prompt.

The scale of an LLM, measured by the number of parameters, plays a crucial role in this scenario. It remains unclear whether the reasoning power of high-parameter models, with tens or hundreds of billions of parameters, can overcome the inherent bottlenecks (Dushkin et al. 2026) of a monolithic structure or if smaller models can achieve parity by utilizing a modular, multi-agent approach.

In this work, we investigate this question in the context of hardware-diagnostic chatbots. Rather than seeking universally optimal prompting strategies, we perform a controlled comparison between monolithic and multi-agent architectures using a benchmark specifically designed for hardware diagnostics (Caminha et al. 2025; Caminha et al. 2026). Hardware troubleshooting provides a challenging evaluation setting because it requires the integration of three heterogeneous tasks: i) diagnostic module prediction: mapping conversation history to the hardware components most likely to be faulty; ii) conversational interaction: accurately communicating diagnostic results; and iii) user message validation: enforcing "guardrails" to keep inputs in scope and prevent prompt injection attacks.

To reduce confounding effects introduced by manual prompt engineering, we employ a genetic algorithm-based prompt optimization framework that automatically generates and refines prompts for both architectural paradigms (Schulhoff et al. 2024). This design allows the comparison to focus on architectural differences rather than subjective prompt construction choices. We evaluate 26 open-source LLMs spanning multiple model families and parameter scales, using both deterministic metrics and LLM-as-a-Judge evaluations to assess diagnostic accuracy, response fidelity, and adherence to application constraints. The contributions of this work are summarized as follows:

- We present a controlled empirical comparison between monolithic and multi-agent prompting architectures in a hardware diagnostics benchmark, evaluating 26 open-source LLMs spanning multiple parameter scales and model families.
- We introduce the use of an automated genetic algorithm-based prompt optimizer as a methodological control to promote a fair comparison, suggesting that the observed performance gains result from structural architectural decomposition rather than subjective manual tuning bias.
- We investigate the relationship between prompt decomposition, model scale, and task complexity through both performance and error analyses. We find that multi-agent architectures provide greater benefits for smaller models and more demanding tasks, while exhibiting distinct patterns of reasoning failures, formatting errors, and task-interference effects.

2. Related Work

Recent research in medical and technical diagnostics highlights a division between monolithic and multi-agent strategies. (Caminha et al. 2025) used a monolithic approach to diagnose hardware faults, demonstrating that prompt strategies enable even small-scale LLMs to perform efficiently on local devices. Within the medical domain, (Wu et al. 2023) further confirms that diagnostic precision is closely tied to the quality of prompt engineering within a single specialized model. In contrast, (Liu et al. 2025) introduced a multi-agent framework that orchestrates specialized LLMs. Unlike the single-model strategy, this approach leverages collaborative agents, specialized in areas such as radiology and patient history, to improve diagnostic accuracy through collective skills.

Beyond diagnostic applications, (Dushkin et al. 2026) observes that monolithic models often suffer from "contextual interference" when processing prompts with multiple tasks. To mitigate this, they propose a hierarchical multi-agent framework employing GPT-4o mini to decompose scholarly workflows, such as literature synthesis, citation formatting, and technical translation, into sub-tasks. Their findings indicate that this distributed approach not only reduces operational costs but also surpasses the accuracy of monolithic GPT-4o in specialized tasks.

While the aforementioned studies address performance through different architectural strategies, another line of research is motivated by the principle that prompt quality is a critical determinant of overall model performance (Schulhoff et al. 2024). Accordingly, recent works investigate the efficacy of prompt optimization to enhance the reasoning of existing models, especially through methods based on evolutionary algorithms. For instance, (Guo et al. 2023) creates a framework that enables evolutionary algorithms to work on discrete prompts. They apply this framework using the genetic algorithm (GA) and differential evolution (DE), and design their respective mutation and crossover steps for LLM prompts. However, this method primarily focuses on structural variations in the limited scope of the prompt space. (Cui et al. 2024) proposes a two-phase strategy for in-context prompt optimization that jointly optimizes the instructions and examples. The first phase is Exploration, with evolution operators leveraged for global search, and the second phase is Exploitation, which involves feedback-guided gradient-based mutation using examples.

3. Our Approach

This section presents the evaluation strategies and methodology for comparing monolithic and multi-agent approaches. We conduct this comparison in the context of a hardware diagnostic chatbot, a suitable scenario given its requirement for multiple well-defined tasks, enabling natural task-driven prompt decomposition and a structured multi-agent design. The evaluation includes three main tasks: user message validation (guardrail), identification of relevant hardware components (diagnostic), and conversational interaction (conversation). Each task exhibits distinct objectives and constraints, allowing the use of heterogeneous evaluation metrics and leading to a robust analysis of model behavior across different functional dimensions.

Our multi-agent architecture follows an orchestrated workflow as illustrated in Figure 1. When a user message is received, it is first processed by the Guardrail Agent, which determines whether the input is valid. If valid, the request is forwarded to the Diagnostic Agent, which decides whether to execute diagnostic procedures and returns a possibly empty list of relevant modules. These modules are executed by the Diagnostic Tool, generating structured results. The Conversation Agent then produces the final response by combining the user input with the diagnostic outputs. If the input is classified as invalid, the orchestration skips the diagnostic stage and directs the user to the Conversation Agent, which enforces scope constraints and provides appropriate guidance.

3.1. Guardrail Task

The guardrail task involves validating user messages by classifying the latest input as valid or invalid according to the chatbot's scope, making it a binary classification problem. This

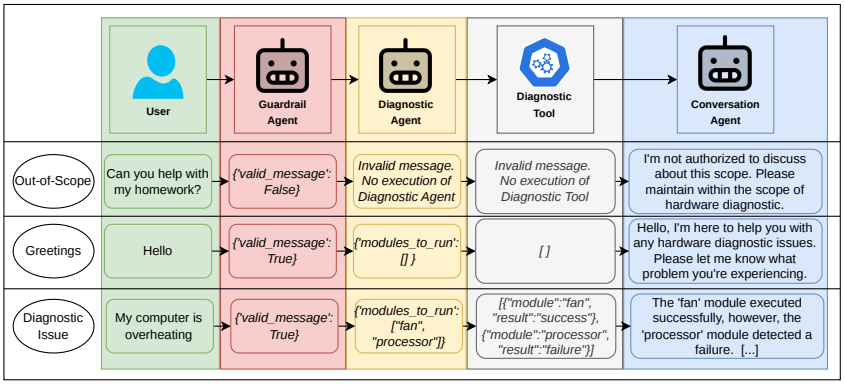


Figure 1. Overview of the multi-agent orchestration pipeline for hardware diagnostic troubleshooting.

task is essential for preventing prompt injection and maintaining proper interaction boundaries. The prompt is structured so that the LLM's expected output includes a reasoning section, followed by a JSON-formatted text containing information about the model's prediction. If the model fails to produce the output in the correct structure, the prediction is automatically considered incorrect and classified as a **Incorrect Format** error. This type of structural validation is relevant to agent-based systems and is used in all evaluations of this study. For this task, classification errors will be referred to as **Miss**. To evaluate this task, we constructed a Guardrail dataset reflecting realistic user behavior in a hardware diagnostic chatbot. Each sample includes the full conversation history, with varying lengths and turn counts, requiring the model to retrieve contextual information from previous turns. The dataset contains 254 instances (142 invalid, 112 valid) and covers diverse scenarios, including clarification requests, contextual follow-ups, device-related information, out-of-scope inputs, conversational markers (e.g., greetings and goodbyes), and empty or random messages.

3.2. Diagnostic Task

The Diagnostic task focuses on integrating chatbot conversational context with hardware diagnostic APIs. Diagnostic capabilities are organized into modular tools, each responsible for analyzing hardware components or related groups of components. The task is to identify which diagnostic modules should be invoked based on the conversation history.

For this evaluation, we developed a multi-label hardware diagnostic dataset containing 23 diagnostic modules and 282 instances. The dataset associates user queries with troubleshooting domains such as performance, thermal behavior, peripherals, connectivity, and combined or unclear conditions. Each instance contains the conversation history as input and two target variables: mandatory modules and optional modules. This distinction allows the evaluation to adapt to different levels of user-provided information. Generic descriptions may not clearly indicate mandatory modules, whereas detailed reports of specific issues, such as graphics-related problems, make certain modules (e.g., "video card") clearly required.

A prediction is considered correct if it satisfies two criteria: i) at least one predicted module appears in the mandatory module list (when it is not empty); and ii) no predicted module falls outside the combined set of mandatory and optional modules. The first criterion ensures that the model identifies modules directly related to the issue and

violations are classified as **Omission** errors. The second prevents recommendations of irrelevant modules, with violations classified as **Overprediction** errors.

3.3. Conversation Task

The final task investigated in this study is conversational interaction. Considering this task is critical, since the chatbot's effectiveness depends on how well these results are reported to the user, regardless of the diagnostic results' accuracy. Unlike the other tasks considered in this study, which rely on classification-based predictions and can be objectively assessed using standard quantitative metrics, conversational interaction requires a qualitative evaluation of the generated textual responses.

Due to the intrinsic complexity of the conversation task, its validation requires multiple evaluations. Our study divided it into two different metrics: the Reporting Faithfulness Metric and the Scope Metric. Because these metrics depend on analyses by LLM extractors and evaluators, respectively, both have calibration datasets. These datasets contain conversation history, diagnostic results, and simulated responses from the Conversation Agent. The responses include both high-quality and flawed examples, either manually written by a human or generated by LLMs from different families and sizes. Each instance is annotated with the expected evaluation outcome (e.g., a score or a list of error types, as detailed in each metric subsection). Importantly, all evaluation targets were produced by a human annotator, providing an external reference independent of the evaluator model. This design allows the calibration process to assess whether the evaluator reproduces the expected human judgments rather than the judge model's inherent preferences or reasoning patterns. The diversity of response sources further reduces the risk of overfitting the evaluator to a particular model family or response style. The model used for these evaluations was "gpt-oss-120b", configured with a temperature of 0.0, as this configuration achieved the highest agreement and stability on the calibration datasets.

3.3.1. Reporting Faithfulness

This metric evaluates whether the diagnostic information communicated by the Conversation Agent is consistent with the Diagnostic Tool's results. It measures the model's ability to leverage conversational context and accurately report the outcomes of executed diagnostic modules.

After inference, an auxiliary extraction model identifies the diagnostic module names and results mentioned in the response, producing a structured list called Extracted Diagnostics. This information is compared against the Diagnostic History, which contains all previously executed diagnostics, and the Recent Diagnostics, which includes only newly generated results that must be communicated to the user.

Three error categories are considered: i) **Hallucination**: a diagnostic module appears in the Extracted Diagnostics but not in the Diagnostic History; ii) **Omission**: a diagnostic module present in the Recent Diagnostics is missing from the Extracted Diagnostics; and iii) **Inaccuracy**: a diagnostic module appears in both the Extracted Diagnostics and Diagnostic History, but the reported outcome differs from the ground truth. A response is considered correct only when none of these errors occur.

3.3.2. Conversational Scope Metric

The second evaluation metric for the conversational task focuses on scope adherence, assessing whether the model's responses comply with the constraints defined for the assistant. These constraints include, but are not limited to, remaining strictly within the domain of hardware diagnostics, using appropriate language for different user groups, responding in a correct format, and respecting text size limitations.

To evaluate scope adherence, we adopt an LLM-as-a-Judge approach. In this setting, a judge model analyzes the assistant's response and assigns a continuous score in the range $[0, 1]$, where 0 indicates an entirely out-of-scope response and 1 denotes complete compliance with the application's constraints.

3.4. Prompt Optimizer

A central difficulty in comparing monolithic and multi-agent prompting lies in the bias introduced by manual prompt design, which involves iterative refinement and subjective decisions, making it hard to ensure comparable effort and quality across architectures.

To mitigate this issue, our study employs an automated prompt-optimization pipeline as a methodological control. Both approaches are optimized under identical conditions, including parameters, computational budget, and evaluation criteria. By standardizing the prompt development process, the optimizer reduces the influence of subjective prompt engineering decisions and helps limit biases arising from unequal manual refinement efforts. Importantly, the objective of this methodology is not to find globally optimal prompts, nor to guarantee complete comparability between architectures, but rather to provide a more systematic and consistent optimization procedure for both approaches. In our experiments, we use a genetic algorithm to optimize prompts. Genetic algorithms are optimization techniques inspired by the principles of natural selection. They operate by maintaining a population of candidates that evolve through evaluation, crossover, and mutation. Prompts are treated as candidate solutions, undergo textual modifications, and are evaluated on the provided datasets and evaluators.

The optimization pipeline takes as input a dataset and a set of evaluation specifications, with configuration and metadata stored in JSON files. The dataset is split into training and validation subsets: the training data is used to generate textual feedback that guides the iterative optimization of the prompt, while the validation data is used to evaluate prompt performance and assign fitness scores.

3.4.1. Optimizer Methodology

The prompt optimization pipeline leverages a genetic algorithm structured around five main components: `Initializer`, `Evaluator`, `Crossover`, `Mutation`, and `Selector`.

The genetic algorithm used a population of 20 candidates over 10 generations, with crossover and mutation rates of 60% and 40%, respectively. Datasets were split into train/validation/test sets using stratified 20/40/40 splits, with the smaller training set used primarily for semantic feedback generation. The choice of 20% for the training set is due

to this subset's lower relative importance, as it serves only to generate semantic feedback for mutation and crossover operators.

No hyperparameter optimization was performed due to the high computational cost of prompt optimization. The number of generations was determined empirically, as fitness plateaued after 10 generations. The population size balanced prompt diversity and computational cost, while the crossover and mutation rates were selected to balance exploitation of complementary prompt behaviors and exploration of the prompt space.

The optimizer begins with the `Initializer`, which processes the evaluation specification to generate seed prompts via reverse engineering, optionally applying techniques such as Chain-of-Thought and Few-Shot while enforcing output formats. The `Evaluator` then computes fitness on the validation set and uses a Feedback LLM ("gpt-oss-120b") to analyze low-scoring training cases and identify directions for improvement, following (Cui et al. 2024).

This feedback guides the evolutionary operators. Our `Crossover` uses semantic feedback to modify error-related sections while preserving correct behaviors. `Mutation` applies feedback-guided changes at four intensities: Small, Medium, and Large, which progressively increase modification scope, and New, which generates a prompt from scratch using the `Initializer`. These steps balance refinement and exploration. Finally, the `Selector` combines elitist selection of top performers with fitness-proportional probabilistic selection to preserve population diversity.

4. Experiments

Our experimental study evaluates monolithic and multi-agent prompting architectures across 26 open-source LLMs from multiple model families, such as LLaMA, Qwen, Gemma, Phi, GPT-OSS, and DeepSeek. Furthermore, our benchmark also covers models of various sizes to analyze differences in behavior across approaches. To support analysis of size behavior, models are grouped according to their number of parameters into four size categories: Very Small ($\leq$ 2B parameters), Small (2B $< n \leq$ 7B), Medium (7B $< n \leq$ 27B), and Large ($n >$ 27B), where n denotes the number of parameters. The benchmark includes the following categories distribution: 5 Very Small models, 5 Small models, 10 Medium models, and 6 Large models.

All models are deployed using 4-bit (Q4_K_M) quantization, which compresses weights into low-bit representations while preserving critical distributional structure through grouped scaling and mixed precision. Prior studies have shown that Q4_K_M generally preserves most of the capabilities of the original models, resulting in relatively modest performance degradation, even for small and medium sized models, while substantially reducing memory requirements. (Dettmers and Zettlemoyer 2023; Kurt 2026), making it one of the most commonly recommended and utilized configurations for practical LLM deployment. Although quantization may introduce some performance degradation relative to higher-precision representations, both the monolithic and multi-agent approaches are evaluated using exactly the same quantized model instances, ensuring that any quantization-induced effects are applied equally to both conditions. As a result, while quantization may affect absolute performance, it does not compromise the validity of the relative comparison between prompting architectures.

4.1. Results

This section presents the results of our comparison between monolithic and multi-agent prompting architectures, examining how the benefits of prompt decomposition vary with model scale across three benchmark tasks: Guardrail, Hardware Diagnostic, and Conversation. Results are presented in Size $\times$ Performance plots, where each point corresponds to a model under both approaches. To better characterize performance trends across heterogeneous model scales, we report bootstrap-based 95% confidence intervals and LOWESS trend estimations. The confidence intervals quantify uncertainty in the observed estimates, while LOWESS provides a non-parametric smoothing of global performance patterns, reducing sensitivity to local fluctuations and model-specific variations. Confidence intervals for the LOWESS curves are also estimated through bootstrap resampling. Vertical dashed lines separate model size groups (Very Small, Small, Medium, and Large), and the x-axis uses a logarithmic scale to capture trends across scales.

Table 1. P-values from the one-sided Wilcoxon signed-rank test (H_1: Multi-Agent $>$ Monolithic) across tasks and model size groups.

Task	Very Small	Small	Medium	Large	Overall
Guardrail	0.0938	**0.0312***	0.1611	0.4219	**0.0108***
Diagnostic	**0.0312***	0.0625	0.3262	0.6875	**0.0055***
Conversation	**0.0312***	**0.0312***	**0.0049***	0.4219	**0.0001***

* Statistically significant at $\alpha = 0.05$. P-values were computed using a one-sided Wilcoxon signed-rank test with the alternative hypothesis that the Multi-Agent approach outperforms the Monolithic approach.

Additionally, to assess whether the observed performance differences are statistically significant, we apply a one-sided Wilcoxon signed-rank test (H_1: Multi-Agent $>$ Monolithic) across model size groups and tasks. The test is performed on paired observations corresponding to the final performance scores obtained by each evaluated LLM under both prompting approaches. Table 1 reports the resulting p-values, highlighting the conditions under which the multi-agent approach achieves statistically significant improvements over the monolithic. Furthermore, Table 2 presents the error distribution for each task and size group, enabling comparisons of error patterns between the two approaches.

Table 2. Distribution of errors across model size groups and prompt approaches.

		Conversation				Diagnostic			Guardrail	
Size Group	Method	Hallucination	Imprecision	Incorrect Format	Omission	Incorrect Format	Overprediction	Omission	Incorrect Format	Miss
Very Small	Mono	25.8 [16.6, 34.0]	2.0 [1.3, 2.6]	91.8 [66.5, 120.5]	9.6 [6.5, 12.2]	131.2 [100.8, 161.6]	47.6 [34.0, 59.7]	36.8 [28.0, 45.6]	77.4 [35.7, 111.7]	107.0 [94.7, 120.8]
	Multi	11.8 [7.9, 14.7]	2.6 [1.5, 3.6]	39.4 [19.0, 57.9]	16.2 [12.0, 20.7]	19.8 [4.2, 32.8]	54.4 [41.7, 67.5]	57.6 [48.8, 65.8]	28.6 [16.7, 39.3]	110.8 [106.1, 116.0]
Small	Mono	32.0 [21.5, 41.0]	0.8 [0.4, 1.1]	85.4 [59.8, 106.8]	6.8 [5.0, 8.5]	79.6 [48.8, 111.9]	75.4 [58.1, 92.5]	25.4 [20.0, 30.2]	47.2 [20.8, 76.7]	97.6 [84.7, 109.5]
	Multi	8.0 [4.2, 11.2]	1.2 [0.7, 1.7]	34.2 [22.9, 44.7]	7.2 [3.9, 10.0]	3.6 [1.8, 5.4]	41.4 [30.6, 51.9]	52.8 [42.7, 63.0]	6.8 [3.6, 9.8]	98.6 [79.0, 117.5]
Medium	Mono	32.5 [27.9, 37.4]	0.2 [0.1, 0.3]	34.1 [23.4, 43.6]	3.2 [2.8, 3.6]	14.9 [9.6, 19.7]	61.5 [55.8, 66.8]	14.6 [12.8, 16.2]	8.4 [4.6, 11.8]	59.3 [54.3, 63.9]
	Multi	1.2 [0.9, 1.5]	0.1 [0.0, 0.2]	5.7 [0.2, 10.0]	3.5 [2.8, 4.1]	0.8 [0.1, 1.3]	53.3 [46.6, 59.9]	32.4 [26.7, 37.0]	13.0 [3.1, 21.2]	60.6 [52.8, 67.5]
Large	Mono	24.0 [19.2, 29.0]	0.0 [0.0, 0.0]	2.3 [1.6, 2.9]	2.8 [2.2, 3.5]	2.0 [1.2, 2.8]	60.5 [54.9, 65.9]	8.8 [7.6, 10.1]	2.2 [1.2, 3.0]	43.3 [39.8, 46.5]
	Multi	1.7 [1.3, 2.0]	0.0 [0.0, 0.0]	28.8 [9.2, 45.3]	3.2 [2.7, 3.6]	0.5 [0.3, 0.7]	48.8 [42.8, 54.6]	24.2 [22.3, 26.1]	1.2 [0.2, 2.0]	45.8 [40.8, 51.4]

Values represent the average count of each error type per model within the group. Values in brackets correspond to the 95% Bootstrap confidence interval.

The guardrail task is the least complex in our study, involving simple user message validation. As shown in Figure 2, the overall curves of both approaches are close, sug-

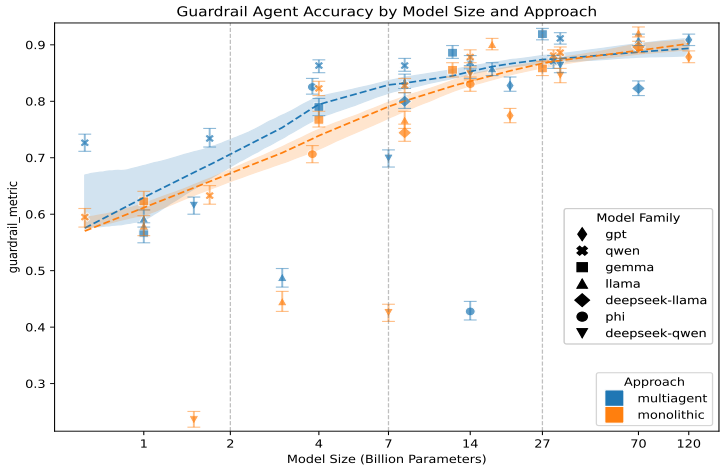


Figure 2. Illustration of Model Size vs Guardrail Metric, with points categorized according to the prompt approach used.

gesting that prompt decomposition has limited impact on easier tasks and that multi-agent benefits may depend on task complexity.

For models with fewer than 14B parameters, the multi-agent approach shows a slight advantage. Table 2 suggests this gap is mainly driven by formatting errors, which are more frequent in the monolithic case, while semantic errors (Miss) remain similar. This suggests that prompt decomposition primarily reduces structural issues in smaller models by limiting interference between instructions rather than improving semantic reasoning. For larger models, the difference becomes negligible as both approaches converge. This is also supported by Table 1, where significant differences favoring the multi-agent approach are observed for smaller model groups and overall, but not for Medium and Large models.

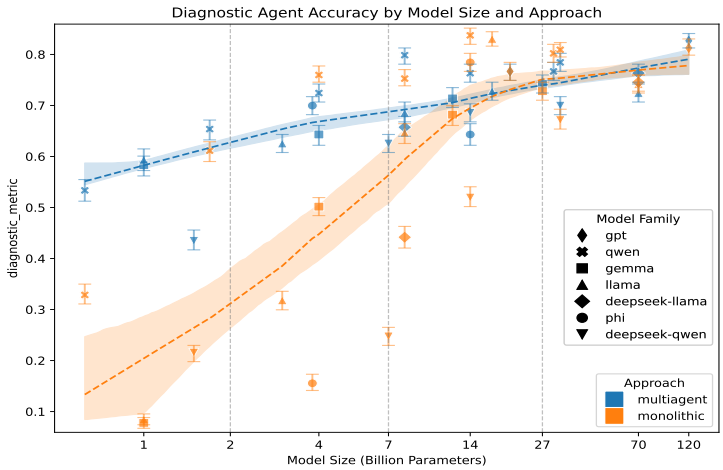


Figure 3. Illustration of Model Size vs Diagnostic Metric, with points categorized according to the prompt approach used.

For the diagnostic task, as shown in Figure 3, the difference between the curves is more pronounced. For smaller and some medium models (up to 10B parameters), the monolithic approach shows a strong drop in accuracy, while the multi-agent curve is more stable. Table 2 shows that this is mainly caused by formatting errors in the mono-

lithic setup, which dominate its failures in these ranges. For larger models (above 27B), formatting issues become more similar across both approaches, indicating that higher capacity helps reduce structural errors. Still, differences persist: the monolithic approach tends to overpredict weakly relevant modules (**Overprediction**), while the multi-agent approach more often misses required modules (**Omission**). Statistical analysis supports this pattern. Table 1 shows significant gains for the multi-agent approach in the Very Small group and in the overall comparison, while no significant differences appear for larger groups. This aligns with the trend curves and suggests that prompt decomposition is most useful when model capacity is limited and the task requires handling multiple constraints at once.

Even so, individual results show exceptions. Among models up to 7B, qwen3-4b achieves the best performance in the monolithic setup. This suggests that the limitations of monolithic prompting depend on model-specific behavior. Some models handle multiple constraints better without decomposition. Overall, multi-agent prompting improves average performance and stability for smaller models, while monolithic prompting can still work better for certain architectures with strong instruction-following and structured reasoning.

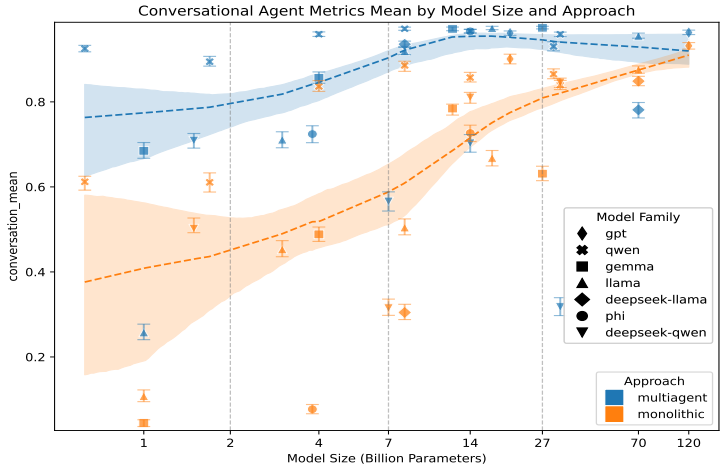


Figure 4. Illustration of Model Size vs Conversation Metric, with points categorized according to the prompt approach used.

For the Conversation task, as shown in Figure 4, the multi-agent approach consistently outperforms the monolithic setup for very small to medium models. As model size increases, the monolithic approach reduces this gap, reaching similar performance only at the largest scale (120B parameters). Table 2 shows that this difference is mainly driven by higher formatting error rates in the monolithic configuration. An exception appears in the Large group, where the multi-agent setup shows a spike in formatting errors due to an outlier model, "DeepSeek-R1-Distill-Llama-70B", which skews the results.

The statistical analysis reinforces these findings. Table 1 shows significant improvements for the multi-agent approach in the Very Small, Small, Medium, and Overall groups, while no significant difference is found for Large models. Compared to the other tasks, this wider range of significant gains suggests that conversational behavior is more sensitive to prompt decomposition, with benefits extending beyond only the smallest models and gradually decreasing as model size grows. The monolithic setup also produces

more hallucination errors in the Reporting Faithfulness metric, especially for very small to medium models. This suggests that conversational performance is sensitive to prompt interference, since diagnostic, guardrail, and conversational instructions compete within a single prompt. This often leads to hallucinations, overly verbose responses, and irrelevant technical details. Larger models handle this separation better, which explains the convergence in performance.

Although multi-agent prompting improves performance for smaller models and complex tasks, it also increases computational cost. While the monolithic approach requires a single inference pass, the multi-agent setup executes three sequential agents (Guardrail, Diagnostic, and Conversation). Consequently, token consumption increases from an average of 1,094 to 2,438 tokens per interaction (+123%), while the inference time increases only modestly, from 1.66 to 1.84 seconds (+11%). This suggests that the main overhead comes from additional token processing rather than latency. Despite the higher cost, the observed performance gains indicate a worthwhile trade-off, particularly for smaller language models at diagnostic and conversational tasks, where prompt decomposition improves reliability and task specialization with only a moderate latency penalty.

5. Conclusion

This work presented a controlled comparison between monolithic and multi-agent prompting architectures across 26 open-source LLMs on a hardware diagnostics benchmark, evaluating guardrail enforcement, diagnostic reasoning, and constrained conversation. Results show that prompt architecture significantly affects smaller models: for complex tasks, prompt decomposition reduces instruction interference and errors such as hallucinations and formatting issues. For simpler tasks or larger models, the performance gap becomes marginal, suggesting that model capacity can compensate for architectural limitations.

A key methodological contribution is the use of a genetic algorithm-based prompt optimizer, which automatically generates and refines prompts for both configurations under identical budgets. This reduces biases from manual prompt engineering and enables a fairer assessment of architectural effects. These findings highlight the benefits of prompt modularity when smaller models are used and accuracy is prioritized. However, these gains must be balanced against the higher computational cost of multi-agent architectures, which require multiple inference calls and larger token budgets.

Although the benchmark provides a controlled environment for evaluating prompting architectures, extending it to new domains still requires considerable manual effort. Future work could investigate automated task decomposition methods that derive specialized subtasks from application specifications. Chain-based prompting could also enable finer-grained decomposition within a single task. These extensions would require adapting the optimization pipeline to optimize sequences of prompts, enabling broader evaluations across domains with reduced manual engineering.

Further research could explore orchestration strategies beyond monolithic, sequential multi-agent paradigms, including hierarchical agents, dynamic routing, and iterative reasoning. Comparing these approaches could provide a broader understanding of how task coordination affects the performance and efficiency of smaller models.

Acknowledgements This research was partially funded by Lenovo, as part of its R&D

investment under the Brazilian Informatics Law, and by the National Council for Scientific and Technological Development (CNPq) under grants 308635/2025-6.

References

[Caminha et al. 2025] Caminha, C., Silva, M. d. L. M., Chaves, I. C., Brito, F. T., Farias, V. A., and Machado, J. C. (2025). Evaluating llms and prompting strategies for automated hardware diagnosis from textual user-reports. In *2025: Anais do LII Seminário Integrado de Software e Hardware*.

[Caminha et al. 2026] Caminha, C., Silva, M. d. L. M., Chaves, I. C., Brito, F. T., Farias, V. A. E., and Machado, J. C. (2026). Diaghw: A compact llm for hardware failure diagnosis via a novel knowledge distillation pipeline. In de Freitas, R. and Furtado, D., editors, *Intelligent Systems*, pages 393–407, Cham. Springer Nature Switzerland.

[Cui et al. 2024] Cui, W., Zhang, J., Li, Z., Sun, H., Lopez, D., Das, K., Malin, B. A., and Kumar, S. (2024). Phaseevo: Towards unified long-context prompt optimization for large language models. In *First Workshop on Long-Context Foundation Models@ ICML 2024*.

[Dettmers and Zettlemoyer 2023] Dettmers, T. and Zettlemoyer, L. (2023). The case for 4-bit precision: k-bit inference scaling laws. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

[Dushkin et al. 2026] Dushkin, R. V., Klimov, V. V., Anzin, Y. A., Dushkin, K. R., and Zvorygin, V. A. (2026). Cost-efficient distributed ai architectures: Lightweight llm agents vs gpt-4o monoliths in academic workflows. In Samsonovich, A. V., Ramos, F., and Liu, T., editors, *Biologically Inspired Cognitive Architectures 2025*, pages 181–190, Cham. Springer Nature Switzerland.

[Guo et al. 2023] Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., Liu, G., Bian, J., and Yang, Y. (2023). Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

[Kurt 2026] Kurt, U. (2026). Which quantization should i use? a unified evaluation of llama.cpp quantization on llama-3.1-8b-instruct.

[Liu et al. 2025] Liu, P. R., Bansal, S., Dinh, J., Pawar, A., Satishkumar, R., Desai, S., Gupta, N., Wang, X., and Hu, S. (2025). Medchat: A multi-agent framework for multimodal diagnosis with large language models. In *2025 IEEE 8th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 456–462. IEEE.

[Schulhoff et al. 2024] Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., et al. (2024). The prompt report: a systematic survey of prompt engineering techniques. *arXiv preprint arXiv:2406.06608*.

[Wu et al. 2023] Wu, C., Lin, Z., Fang, W., and Huang, Y. (2023). A medical diagnostic assistant based on llm. In *China Health Information Processing Conference*, pages 135–147. Springer.