Small Language Models for Real-World Task Automation in Resource-Constrained Environments

Hugo R. B. Gonçalves, Pedro B. G. Neto, Guilherme Shinohara, Jonathan de Andrade Silva, Brivaldo A. da Silva Jr

Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79.070-900 – Campo Grande – MS – Brazil

{hugo.belonia,pedro.b.g.neto,g.shinohara}@ufms.br
{jonathan.andrade,brivaldo.junior}@ufms.br

Abstract. *This work investigates the viability of Small Language Models (SLMs) as cost-effective alternatives to Large Language Models (LLMs) for automation tasks in resource-constrained environments. We implement AI agents using the low-code n8n platform and lightweight, self-hosted models served through the Ollama engine, thereby reducing operational costs and computational overhead. We evaluate the semantic similarity between the outputs of these SLMs and those of high-capacity LLMs, including Gemini 3 Flash and Claude Sonnet 4, using BERTScore. The results show that the best-performing SLM achieved F1 scores close to 80%, indicating strong semantic alignment despite limited computational resources and network access.*

1. Introduction

Artificial Intelligence (AI), particularly through Large Language Models (LLMs), has been widely adopted to automate a variety of daily tasks. However, these models require substantial computational resources for training and deployment, in addition to presenting practical limitations such as high API operational costs and increasing concerns regarding energy, water, and carbon consumption [Licardo and Tankovic 2025, Sakib et al. 2025]. Due to their massive scale and infrastructure requirements, deploying LLMs at the network edge (e.g., schools and healthcare centers) remains challenging, especially in scenarios with limited Internet connectivity and latency constraints [Rang et al. 2025, Nguyen et al. 2024, Sakib et al. 2025].

Small Language Models (SLMs) have emerged as an alternative to the prevailing use of LLMs [Licardo and Tankovic 2025]. Their smaller parameter counts and more limited training data can constrain their capacity for *deep reasoning*. Nevertheless, SLMs have recently gained attention for their ability to perform a variety of everyday tasks with minimal computational resources [Sakib et al. 2025, Nguyen et al. 2024].

SLMs are therefore well suited to scenarios in which LLM deployment is impractical because of limited computing resources, especially on mobile devices and edge systems. Research shows that smaller models, when adapted for a specific domain, can achieve performance comparable to or superior to larger and general-purpose LLMs [Licardo and Tankovic 2025, Sharma et al. 2024, Li et al. 2023]. This domain adapting pretraining approach (DAPT) offers an attractive value proposition and can reduce Total Cost Ownership (TCO), i.e., the sum of expenses required to acquire,

operate and maintain the technology, in approximately 90% to 95% compared to LLMs [Sharma et al. 2024].

This study aims to develop AI agents [Russell and Norvig 2010, Wooldridge 2001] for lightweight tasks using such models as inference engines integrated with the n8n orchestration platform and the Ollama runtime. We evaluate the agents on two tasks designed to represent real-world conditions: *chatbot-based task scheduling* and *pattern recognition and extraction from large text files*. Both are routine workplace or everyday activities that are often performed manually and are monotonous. To this end, we use n8n [n8n-io 2025], a low-code automation platform, combined with lightweight models such as those from the **`Qwen2.5`** family and **`Llama 3.2`**. This work evaluates lightweight models against high-capacity LLMs, including Gemini 3 Flash and Claude Sonnet 4, using BERTScore. The best models achieve F1 scores close to 80%. It also advances the practical adoption of SLMs in resource-constrained environments, helping democratize AI and bridge the gap between research and real-world applications.

The presentation of this work is organized as follows: Section 2 presents related works, providing the basis of agent architecture and the feasibility of SLMs; Section 3 describes the SLMs, and details the implementation with n8n; and Section 4 shows the results of automation agents developed for academic use cases. Finally, Section 5 presents the conclusion and future works.

2. Related Works

This section reviews strategies for applying Small Language Models (SLMs) in specialized, resource-constrained contexts. Prior work commonly relies on domain adaptation through fine-tuning or on knowledge distillation, in which a smaller student model learns from a larger teacher model. Although effective, these approaches may require substantial computational resources and specialized training datasets. By contrast, our work investigates prompt engineering and modular orchestration through chains of compact models, avoiding intensive retraining.

[Fernandes et al. 2025] investigated the feasibility of domain adaptation through supervised fine-tuning (SFT) of smaller language models, including **`Llama 2:7B`** and **`Llama 3:8B`**, for legal reasoning on a U.S. multiple-choice examination. The authors selected 1,514 questions and used **`Llama 3:70B`** to refine the answers according to IRAC, a legal reasoning framework that structures responses into Issue, Rule, Application, and Conclusion. They validated the models on 200 questions from the 2022 examination, using accuracy and parsing error rate (failures to follow the formatting).

The study concluded that supervised fine-tuning can substantially improve performance even with limited training data. The adapted models achieved results competitive with GPT-4, produced fewer parsing errors, and demonstrated that smaller open-weight models can be effectively adapted to specialized domains in resource-constrained environments.

[Barcelar et al. 2025] proposed a knowledge-distillation method that combines **`Qwen 2.5-14B`** as the teacher and **`Qwen 2.5-1.5B`** as the student to summarize police reports. Their approach reduced VRAM use during inference by approximately 90%. However, it targets a single domain and requires a large teacher model during

distillation, preventing a direct comparison under our hardware constraints. Our evaluation instead considers scenarios in which memory offloading and latency make even 7B or 8B models impractical. We use chains of compact models ranging from 0.5B to 3B parameters for tasks such as chatbot-based automation and document analysis within a strict 4GB VRAM limit.

Finally, [Chen et al. 2025] investigated SLMs for multi-point robotic navigation on edge devices, seeking to overcome the latency and privacy limitations of APIs that provide access to large models. The authors fine-tuned the models to standardize their JSON outputs and used an interactive *teacher–student* module in which GPT-4 supplied prompts and feedback to refine the reasoning of smaller models, including **`tau-0.5B`**, **`TinyLlama-1.1B`**, **`h2o-danube-1.8B`**, and **`openllama-3B`**. They conducted experiments in both simulated and real-world environments, using an SLM-guided robot to navigate corridors and other complex spaces. We did not evaluate these models because they lack adequate multilingual support.

Although prior studies report promising results for SLMs in specialized test settings, they provide limited evidence about the use of general-purpose SLMs for everyday automation under strict hardware constraints. This study addresses that gap through two practical tasks in environments where limited computing resources hinder the deployment of LLMs.

3. Building Blocks

The adoption of SLMs offers a practical response to the cost and scale challenges of LLMs. However, this approach involves a trade-off in terms of response accuracy. The following sections explore both the operational and economic advantages that make them viable, as well as the technical and performance challenges inherent in their more limited capacity.

3.1. Small Language Models (SLMs)

SLMs are compact, transformer-based language models that typically range from millions to a few billion parameters, whereas LLMs often contain tens or hundreds of billions. Their smaller size makes SLMs more efficient and better suited to resource-constrained tasks while allowing strong performance in domain-specific applications [Wang et al. 2025, Vakili et al. 2024].

SLMs seek to preserve the accuracy or adaptability of larger models while operating under constraints on training and inference hardware, bandwidth, and generation time. They can therefore improve the accessibility and sustainability of AI in resource-constrained environments [Australian Institute for Machine Learning (AIML) 2024]. Despite these operational advantages, SLMs have limited capacity for *deep reasoning*, and their efficiency may depend on the underlying hardware and software ecosystem. This dependence is particularly relevant to optimization techniques such as quantization[1]

Finally, SLMs raise *ethical and quality concerns* similar to those associated with LLMs. Their lower model capacity and potentially narrower training corpora may make

[1]Quantization reduces parameter precision to make a model smaller, faster, and less memory-intensive. For example, parameters represented using 32-bit (FP32) or 16-bit (FP16) floating-point values can be converted to 8-bit integers (INT8) or lower-precision formats. Converting an FP32 model to INT8 can reduce its size by up to 75%.

them more susceptible to hallucination[2] and bias [Sakib et al. 2025].

3.2. Computational Environment

We present two workflows that evaluate the practical capabilities of SLMs. The workflows ran on a Linux server with Debian 12, kernel 6.1.0, and Docker 29.0.2. All models ran on a resource-constrained NVIDIA Quadro P1000 with 4GB of VRAM, using driver version 535.247.01, CUDA 12.2, and GPU passthrough for the containers.

We selected models that could operate within the 4GB VRAM limit to minimize the system-RAM bottleneck: **`Qwen 2.5`** (**`0.5B`**, **`1.5B`**, and **`3B`**), **`Llama 3.2`** (**`1B`** and **`3B`**), and **`Phi 3:3.8B`**. Ollama can execute larger models by retaining part of a model in GPU memory and offloading the remainder to system RAM, but this split substantially increases response latency. When generation exceeds 10 minutes, n8n times out, terminates the workflow, and interrupts the user interaction. This latency issue was particularly prevalent with **`Phi 3:3.8B`**, so we excluded it from subsequent experiments.

3.3. Experimental Scenarios

This section describes two problems employed to evaluate the limitations of SLMs, along with the subsequent solutions developed to address the identified issues.

3.3.1. Task List Following Rules (Pomodoro)

The goal was to receive task lists through a ChatBot interface, such as n8n Chat, and schedule them in Nextcloud Calendar according to the Pomodoro method[3].

Figure 1 illustrates the primary execution flow of the agent. The process begins when a message is received via the n8n chat interface. Because SLMs can be sensitive to character encoding, we normalize the input text by removing non-ASCII characters, such as accents and special symbols, before processing.

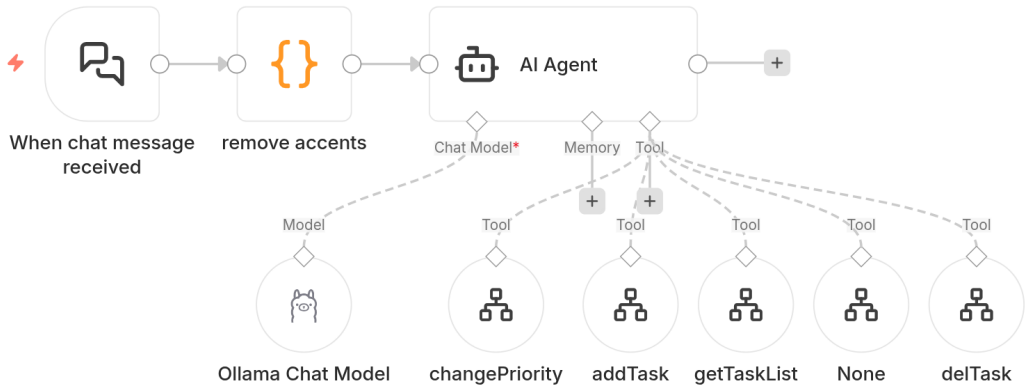


Figure 1. Main workflow of the task management agent (Pomodoro).

[2]The generation of a response that appears plausible but is incorrect, fabricated, or inconsistent with the context.

[3]The Pomodoro method divides work into 25-minute focus intervals separated by 5-minute breaks, with a 20-minute break after every four focus intervals.

In this scenario, the agent acts strictly as an Information Router. A concise, explicit system message instructs it to analyze each user input and route it to the appropriate tool. The available tools are task-specific subflows: **addTask** (add tasks), **getTaskList** (view the task list), **delTask** (remove tasks), **changePriority** (reorder and change the priority of a task), and **None** (handle unmatched inputs).

When the main agent identifies an input as a new task, it sends the unmodified user input as raw JSON to the **addTask** tool. The `Information Extractor` node, powered by **`Qwen2.5:3B`**, then applies a JSON schema to extract structured fields such as the task summary, deadline, and priority.

Before writing task data to the Nextcloud spreadsheet, the system retrieves the existing entries and checks for duplicates. A conditional gate then selects one of two paths. If the task is new, the workflow proceeds to the *Write File and Upload to Nextcloud* node and invokes the **CompleteCalendarPomodoro** subflow. If the task name already exists, the workflow informs the user and terminates, thereby preventing duplication.

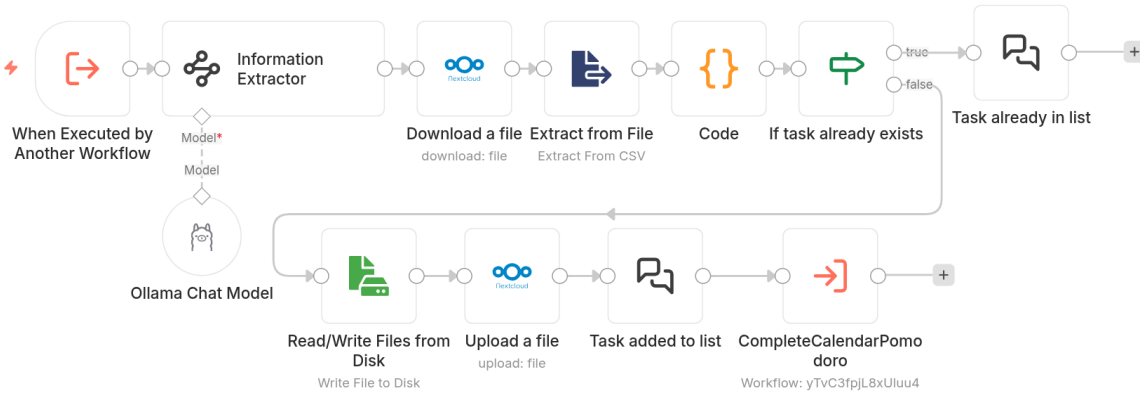


Figure 2. Task Management Agent (Pomodoro) Sub-Workflow ListTasks.

Thus, when the **CompleteCalendarPomodoro** subflow is invoked, it reconstructs the sequence of activities in the schedule as new tasks are added, removed, or changed. In the first step, all tasks are retrieved using the Nextcloud Calendar and Sheets API. Next, all tasks that have already been executed are removed from the schedule.

Figure 3 illustrates a two-stage process, both utilizing SLMs. In the first stage, an AI agent is configured with the persona of a *Dialogue Screening Assistant* using the **`Qwen2.5:3B`** model. Guided by a strict *System Message*, this agent performs a purely analytical role: it classifies user intent into predefined categories (e.g., greeting, help request, or error report) and generates a standardized response in English outlining the system's command syntax.

The second stage employs a *Translator* agent powered by the **`Llama3.2:3B`** model. Its only task is to translate the previous stage's output from English into Brazilian Portuguese. This modular approach was adopted because empirical testing revealed that the classification performance of the first model was superior when prompted in English, despite the final output requiring Portuguese.

Finally, the *Chat* node delivers the translated response to the user, closing the feedback loop and ensuring that out-of-scope interactions are handled in an instructive

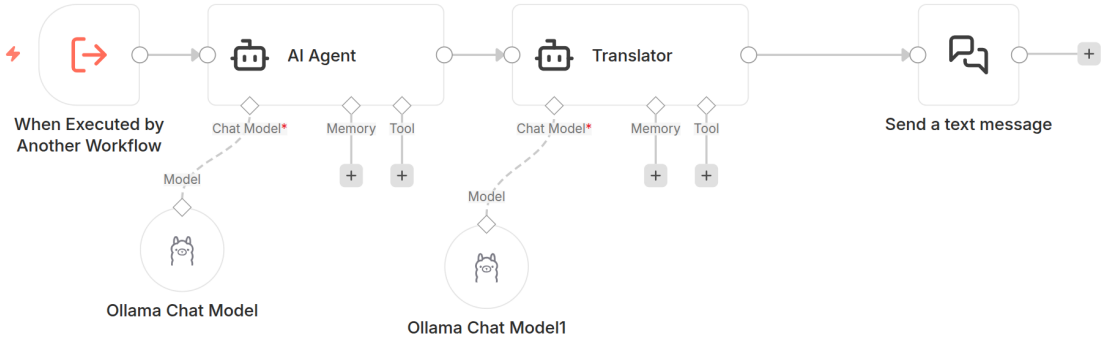


Figure 3. Task Management Agent (Pomodoro) Sub-Workflow None.

and natural manner. This architecture demonstrates that for complex scenarios, a hybrid strategy involving multiple coordinated AI agents can yield superior results compared to a single-model approach.

3.3.2. Pattern Recognition and Extraction

Our second use case involves processing PDF documents, specifically the University's Bulletin, to identify mentions of specific individuals (staff or students) or targeted topics, such as projects or equipment acquisitions. This automated approach eliminates the need for manual review of the entire document and triggers a notification to the user whenever these predefined patterns are detected.

Figure 4 illustrates the document processing workflow. It implements a long-document architecture based on a pre-filtering strategy, which combines regular expressions[4] with generative summarization[5].

Following the conversion of the PDF to text via the `Extract from File` node, the data stream enters a three-stage preprocessing pipeline designed to circumvent the context window limitations of SLMs. The first code node, `get subject`, uses regular expressions to segment the continuous text into logical blocks of administrative acts, identified by date patterns and specific terminology. The algorithm iterates through these blocks and immediately discards any segment that does not contain the user-provided keyword, significantly reducing the data volume.

Next, the `fetch pages` node scans the filtered blocks for pagination markers. It performs numerical normalization by converting string-based page numbers into integers and applying a unitary increment $(n + 1)$ to correct the indexing discrepancy, or "off-by-one" error, between the extracted text and the PDF viewer. Before reaching the SLM, the *cleaning* node applies a strategic truncation algorithm. If the text exceeds 2000 characters, the node preserves the header (first 30 lines), the footer (last 10 lines), and a focused context window around the keyword (20 lines), discarding the irrelevant middle section. This ensures the **`Qwen2.5:3B`** model receives a high-density "context snapshot",

[4]A sequence of characters defining a search pattern.
[5]The process of synthesizing a concise summary from an input text.

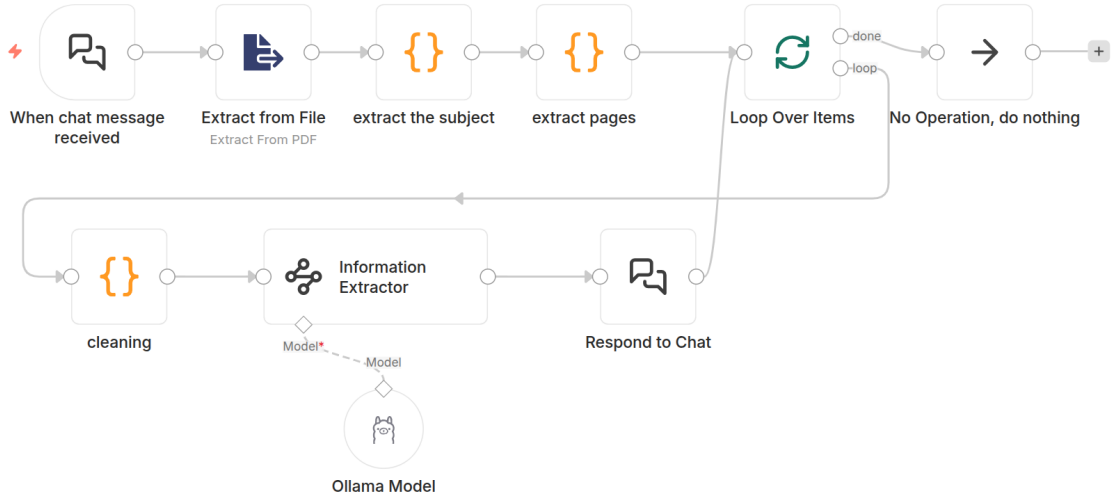


Figure 4. PDF Document Reader Workflow.

maximizing extraction accuracy within its token budget. Although we initially evaluated the Qdrant vector database, hardware constraints necessitated the use of regular expressions to minimize the system's memory and processing footprint.

Finally, the flow iterates over the processed results (*Loop Over Items*) to deliver a structured summary to the user via the *Chat* node, including the specific act and its reference page. This architecture efficiently handles pattern searching in large-scale documents such as service bulletins, official journals, or technical books.

4. Achievable Results

This section provides an evaluation of the results from the deployment of the proposed automation mechanisms and analyzes the edge cases identified during the implementation and validation.

4.1. Evaluation of SLMs

We benchmarked the SLMs to balance comprehension and accuracy against the constraints of low-cost hardware. **`Llama 3.2:3B`** proved inadequate because of frequent hallucinations and poor tool-integration performance, while **`Llama 3.2:1B`** performed even worse. We excluded **`Phi 3:3.8B`** because its VRAM requirements and inference latency repeatedly caused it to exceed n8n's 10-minute execution limit.

The **`Qwen 2.5:3B`** model provided the best performance-to-resource balance, producing reliable tool calls and schema-compliant JSON within the 4GB VRAM and 10-minute execution limits. Although the smaller variants were faster, their lower reliability made them unsuitable for workflow orchestration. Tables 1 and 2 summarize the runtime and qualitative results, respectively.

Table 1. Average response time and speed of the SLMs on the Quadro P1000. Results are reported as mean ± standard deviation over 100 repetitions.

Model	Response Time (s)	Speed (tok/s)
`Llama 3.2:1B`	7.69 ± 1.05	40.21 ± 0.32
`Llama 3.2:3B`	14.38 ± 1.72	20.23 ± 0.22
`Qwen 2.5:0.5B`	7.15 ± 1.76	65.68 ± 0.16
`Qwen 2.5:1.5B`	11.09 ± 1.60	33.83 ± 0.44
`Qwen 2.5:3B`	$\mathbf{17.71 \pm 2.57}$	$\mathbf{19.60 \pm 0.22}$
`Phi 3:3.8B`	46.82 ± 145.56	17.02 ± 1.65

Table 2. Comparative Analysis of SLM Performance in Automation Workflows

Model	Parameters	Tool-Calling	JSON Integrity	Viability
`Qwen 2.5`	0.5B	Non-functional	Malformed	Works
`Qwen 2.5`	1.5B	Intermittent	Inconsistent	Works
`Qwen 2.5`	**3B**	**Reliable**	**Syntactic**	**Works**
`Llama 3.2`	1B / 3B	Deficient	High Hallucination	Works
`Phi 3`	3.8B	N/A	Timeout Error	Incompatible[1]

[1] Exceeded the 10-minute n8n execution threshold due to VRAM and inference latency.

4.1.1. Tool-calling Limitations

The following tables compare model responses to two prompts in Portuguese (English translations are provided in parentheses):

- "Apagar a luz da cozinha antes de dormir" (Text 1)
  ("Turn off the kitchen light before going to sleep")

- "Preciso que você anote uma coisa para mim" (Text 2)
  ("I need you to write something down for me")

This evaluation examines how each model processes natural-language requests, focusing on linguistic accuracy and instruction adherence. The tables report the raw response to each input and a manual qualitative assessment: 'OK' indicates that the response satisfies the contextual and functional requirements, whereas 'Error' indicates a hallucination or substantial deviation from those requirements. Tables 3 and 4 present the results for Texts 1 and 2, respectively. The following box contains the prompt used in the evaluation.

> You are a routing agent. Your only function is to analyze the user's message and send it to the correct tool.
>
> **PRIMARY DIRECTIVE (MAXIMUM PRIORITY):** If the user's message describes any kind of task, action, reminder, or to-do list item, it **MUST** be sent to the 'addTask' tool. This rule overrides all others in case of doubt. A task is any phrase that could be written on a checklist.
>
> Always respond in JSON format with the keys "tool" and "argument". The "argument" must be the original user's message.

Table 3. Model outputs for Text 1. The status is 'OK' when the output satisfies the expected requirements and 'Error' otherwise.

Model	Response to Text 1	State
Qwen 2.5:0.5B	>"tool": "changePriority", "argument": "c/ 1 6"	Error
Qwen 2.5:1.5B	"tool": "addTask", "argument": "pagar a luz..."	Error
Qwen 2.5:3B	"Apagar a luz..." {"r/ 2"}	Error
Llama 3.2:1B	"type":"func", "func":{"name":"None"...}	Error
Llama 3.2:3B	["apagar a luz da cozinha..."]	OK

Table 4. Model outputs for Text 2.

Model	Response to Text 2	State
Qwen 2.5:0.5B	"tool": "addTask", "arguments": "preciso que voce anote..."	Error
Qwen 2.5:1.5B	"tool": "addTask", "argument": "preciso que vc anote isso"	Error
Qwen 2.5:3B	"anote uma coisa"	OK
Llama 3.2:1B	"Preciso que voce anote uma coisa para mim"	OK
Llama 3.2:3B	"Preciso que voço anotar uma coisinha..."	Error

4.1.2. Rule-Based Task List Management (Pomodoro):

The **Qwen2.5:3B** model successfully distinguished ordinary dialogue from task commands in most cases. The workflow added submitted tasks to the local spreadsheet and then populated the user's Nextcloud Calendar with Pomodoro blocks. In repeated tests, the concise, structured commands "c/" for reprioritization and "r/" for removal were processed reliably.

4.1.3. Pattern Recognition and Extraction

The proposed model effectively identified and tracked patterns of interest. During validation, the system processed a university bulletin hundreds of pages long to assess the model's ability to retain context. Through the layered architecture of code-based heuristic filtering and AI-driven extraction described in 3.3.2, the workflow reduced the corpus to discrete, relevant segments. This setup allowed us to assess whether the model could recover the broader context from fragmented inputs. Powered by **Qwen 2.5:3B**, the *Information Extractor* parsed these segments, categorized the identified mentions, generated semantically consistent summaries, and mapped each citation to its corresponding page number.

This hybrid approach can also incorporate automated notification triggers that alert users in real time when a pattern is detected. As shown in Table 5, the best SLM achieved BERTScore F1 values close to 80% relative to outputs from Gemini 3 Flash, GPT-4o mini, and Claude Sonnet 4, indicating strong semantic alignment.

To obtain these scores, we constructed an evaluation corpus from the same university bulletin. A Python script extracted each administrative section and stored it as a separate spreadsheet entry, yielding 213 entries about general public interest or mentioning specific individuals and university departments. For each entry, a JavaScript routine divided the text into three segments representing its beginning (30 lines), middle (20 lines), and end (the last 10). These segments were submitted through Ollama to each model, which was instructed to generate a summary. This procedure reproduced the fragmented-input conditions of the *Information Extractor* and allowed us to assess whether the SLMs preserved semantic consistency under the system's context constraints.

We submitted the same 213 entries to Gemini 3 Flash, GPT-4o mini, and Claude Sonnet 4. We then compared each SLM-generated summary with the corresponding summary from each reference LLM using BERTScore precision, recall, and F1. Table 5 reports the scores aggregated across the 213 comparisons for each model. The **`Qwen 2.5:3B`** model achieved the highest F1 score with all three references: 0.7960 with Gemini 3 Flash, 0.7827 with GPT-4o mini, and 0.7743 with Claude Sonnet 4. Because the reference summaries were generated automatically, these scores quantify semantic similarity to the selected LLM outputs rather than providing a comprehensive measure of summary quality.

Table 5. Global BERTScore Comparison.

	Gemini 3 Flash			GPT-4o mini			Claude Sonnet 4		
SLM Model	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Qwen 2.5:0.5B	0.6370	0.6659	0.6493	0.6552	0.6626	0.6571	0.6552	0.6591	0.6553
Qwen 2.5:1.5B	0.7739	0.7207	0.7453	0.7552	0.6828	0.7163	0.7555	0.6727	0.7104
Qwen 2.5:3B	0.8095	0.7852	**0.7960**	0.8135	0.7554	**0.7827**	0.8148	0.7401	**0.7743**
Llama 3.2:1B	0.4620	0.4721	0.4659	0.4613	0.4606	0.4599	0.4705	0.4560	0.4619
Llama 3.2:3B	0.7942	0.7663	0.7786	0.8037	0.7463	0.7729	0.8033	0.7315	0.7638

`Llama 3.2:1B` obtained BERTScore F1 values below 0.50 against all three reference models, reflecting its difficulty in maintaining semantic coherence during the evaluation.

4.1.4. Window Context Limitations and Mitigation Strategies

The deployment on resource-constrained hardware (NVIDIA Quadro P1000) imposed a strict 2048-token context window limitation. Our tests revealed a positive correlation between increased context density and the frequency of hallucinations across all evaluated models. To mitigate these hardware constraints, the workflow architecture was restructured to prioritize task atomicity. By decomposing complex operations into modular, objective sub-tasks, the agents were provided only with the specific data fragments required for each stage, thereby minimizing signal-to-noise interference.

We integrated this strategy into the Pattern Recognition and Extraction pipeline to address the hardware limitations. Because *single-shot* processing of long documents was infeasible within the 2048-token constraint, we implemented a keyword-based heuristic that partitions documents into manageable blocks before inference and preserves relevant content within each window.

Embedding models such as **snowflake-arctic-embed:22m**, however, proved unable to capture the relevant semantic nuances in Portuguese. Under the strict 2048-token limit, their low retrieval precision supplied the model with fragmented or insufficient context. We therefore abandoned the Retrieval-Augmented Generation (RAG) architecture in favor of more reliable keyword-based segmentation and direct extraction.

5. Conclusion

This work implemented and validated SLM-based workflows for two scenarios in resource-constrained environments: *task-list scheduling* and *pattern-based information extraction from PDF documents*. The goal was to provide low-cost, efficient automation using a low-code platform (n8n) and a GPU with 4GB of VRAM (the NVIDIA Quadro P1000 used in our experiments).

To fit within the limited reasoning capacity and context windows of SLMs, we decomposed each workflow into small sequential tasks controlled by modular prompts. Auxiliary routines filtered and formatted the input and validated the output. This hybrid approach achieved BERTScore F1 values close to 80% relative to commercial LLM outputs, indicating strong semantic alignment despite the models' small scale.

In the comparative evaluation, **`Qwen 2.5:3B`** performed best across the practical scenarios, offering the most favorable balance between computational efficiency, instruction following, and reliable tool use. These findings indicate that SLMs can solve well-defined problems when commands are simple and instructions are direct and explicit. This architecture complements conventional rule-based programming by handling decisions where hardcoding every condition is impractical, such as maintaining long chains of `IF` statements to account for variations in user intent.

In conclusion, when properly orchestrated through low-code platforms such as n8n, SLMs can solve well-defined problems with low computational and development costs, thereby broadening access to intelligent automation.

5.1. Future Work

Given the performance of **`Qwen2.5:3B`**, future work should investigate its quantization for compact systems such as the Orange Pi, Raspberry Pi 5, and Jetson Nano. These optimizations could enable fully offline SLM deployments and provide intelligent automation to schools and healthcare centers in remote locations without high-performance hardware or reliable network access.

References

Australian Institute for Machine Learning (AIML) (2024). Beyond Bigger Models: The Evolution of Language Model Scaling Laws. [Online; accessed 25-05-2026].

Barcelar, R., Garcia, L., Heleno, A., Ventura, T., and Oliveira, A. (2025). Knowledge Distillation in Compact Models: An Approach Applied to Text Processing for Public Security. In *Anais do XVI Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana*, pages 42–51, Porto Alegre, RS, Brasil. SBC.

Chen, Y., Han, Y., and Li, X. (2025). FASTNav: Fine-Tuned Adaptive Small-Language-Models Trained for Multi-Point Robot Navigation. *IEEE Robotics and Automation Letters*, 10(1):390–397.

Fernandes, R., Biedenkapp, A., Hutter, F., and Awad, N. (2025). A Llama Walks into the 'Bar': Efficient Supervised Fine-Tuning for Legal Reasoning in the Multi-state Bar Exam. [Online; accessed 25-05-2026].

Li, Y., Bubeck, S., Eldan, R., Giorno, A. D., Gunasekar, S., and Lee, Y. T. (2023). Textbooks Are All You Need II: phi-1.5 Technical Report. [Online; accessed 25-05-2026].

Licardo, J. T. and Tankovic, N. (2025). Performance Trade-offs of Optimizing Small Language Models for E-Commerce. [Online; accessed 25-05-2026].

n8n-io (2025). n8n: Secure Workflow Automation for Technical Teams. GitHub. [Online; accessed 25-05-2026].

Nguyen, C. V. et al. (2024). A Survey of Small Language Models. [Online; accessed 22-08-2026].

Rang, M., Bi, Z., Zhou, H., Chen, H., Xiao, A., Guo, T., Han, K., Chen, X., and Wang, Y. (2025). Revealing the Power of Post-Training for Small Language Models via Knowledge Distillation. [Online; accessed 25-05-2026].

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Sakib, T. H., Hosain, M. T., and Morol, M. K. (2025). Small Language Models: Architectures, Techniques, Evaluation, Problems and Future Adaptation. [Online; accessed 25-05-2026].

Sharma, A., Ene, T.-D., Kunal, K., Liu, M., Hasan, Z., and Ren, H. (2024). Assessing Economic Viability: A Comparative Analysis of Total Cost of Ownership for Domain-Adapted Large Language Models versus State-of-the-Art Counterparts in Chip Design Coding Assistance. [Online; accessed 25-05-2026].

Vakili, Y. Z., Fallah, A., and Sajedi, H. (2024). Distilled BERT Model in Natural Language Processing. In *2024 14th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 243–250.

Wang, F., Zhang, Z., Zhang, X., Wu, Z., Mo, T., Lu, Q., Wang, W., Li, R., Xu, J., Tang, X., He, Q., Ma, Y., Huang, M., and Wang, S. (2025). A Comprehensive Survey of Small Language Models in the Era of Large Language Models: Techniques, Enhancements, Applications, Collaboration with LLMs, and Trustworthiness. *ACM Trans. Intell. Syst. Technol.*, 16(6).

Wooldridge, M. J. (2001). *An Introduction to Multiagent Systems*. John Wiley & Sons, Ltd.