

Mecanismos de Automação usando SLMs

Hugo Ribeiro Belonia Gonçalves, Brivaldo Alves da Silva Jr

¹Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79.070-900 – Campo Grande – MS – Brazil

{hugo.belonia, brivaldo.junior}@ufms.br

Abstract. This work investigates and validates the viability of Small Language Models (SLMs) as a cost-effective alternative to Large Language Models (LLMs) for automation tasks. In response to the high operational and computational costs inherent in LLMs, the use of specialized SLMs is proposed, demonstrating potential for optimizing resources. The objective of this project is to develop and implement practical Artificial Intelligence agents using the low-code n8n platform in conjunction with lightweight self-hosted models based on the Ollama engine. The application aims to solve everyday problems in academic environments, but can be easily expanded to other scenarios.

Resumo. Este trabalho investiga e valida a viabilidade dos Pequenos Modelos de Linguagem (SLMs) como uma alternativa eficaz e de baixo custo aos Grandes Modelos de Linguagem (LLMs) para tarefas de automação. Em resposta aos elevados custos operacionais e computacionais inerentes aos LLMs, propõe-se o emprego de SLMs especializados, que demonstrem potencial para otimizar recursos. O objetivo deste projeto é desenvolver e implementar agentes de Inteligência Artificial práticos, utilizando a plataforma low-code n8n em conjunto com modelos leves auto-hospedados baseados no motor Ollama. A aplicação tem como objetivo a resolução de problemas cotidianos em ambientes acadêmicos, mas que podem ser expandidos para outros cenários sem dificuldades.

1. Introdução

O avanço da Inteligência Artificial (IA), especialmente com o desenvolvimento dos Grandes Modelos de Linguagem (LLMs), transformou o cenário tecnológico com sua impressionante capacidade de resolver diversas tarefas cotidianas. Por exemplo, tarefas como: transcrição de áudio para texto, tradução de idiomas, resumo de documentos longos, criação de código de programação, são amplamente realizadas por modelos de IA [Sakib et al. 2025].

Contudo, o sucesso dos modelos de grande escala gera custos significativos e limitações práticas [Licardo and Tankovic 2025]. O treinamento e a operação de LLMs demandam recursos computacionais consideráveis, incluindo infraestrutura especializada e grandes volumes de dados [Sakib et al. 2025]. Por outro lado, existem ainda as desvantagens relacionadas aos custos operacionais gerados pelo uso de serviços de API, a criação de dependência de fornecedores (vendor *lock-in*), e crescentes preocupações em relação ao uso excessivo de energia, água e carbono [Licardo and Tankovic 2025]. Devido à escala massiva e os altos custos computacionais, o uso destas tecnologias na borda da rede

(escolas, faculdades, postos de saúde, etc.), os tornam pouco adequados para implantação e uso, principalmente devido às restrições de acesso à Internet e problemas relacionados à latência. [Rang et al. 2025, Nguyen et al. 2024, Sakib et al. 2025].

Uma solução que tem sido estudada na contramão da tendência de uso das LLMs, são os Pequenos Modelos de Linguagem (SLMs) [Licardo and Tankovic 2025]. Esses modelos não são tão *inteligentes* pois a quantidade de parâmetros e dados de treinamentos usados em seu desenvolvimento é limitada. Contudo, os SLMs ganharam atenção recentemente pela sua capacidade de executar diversas tarefas com recursos computacionais mínimos [Sakib et al. 2025, Nguyen et al. 2024].

A existência de cenários que precisam destes tipos de aplicação, mas possuem poder computacional limitado torna o uso de SLMs, ideais, principalmente para ambientes envolvendo dispositivos móveis e sistemas de borda. Pesquisas mostram que modelos menores, quando especializados para um domínio, podem atingir desempenho comparável ou superior aos LLMs maiores e de propósito geral [Licardo and Tankovic 2025, Sharma et al. 2024, Li et al. 2023]. Esta abordagem de adaptação ao domínio (DAPT) resulta em uma proposta de valor atraente, com a capacidade de reduzir o Custo Total de Propriedade (*Total Cost Ownership - TCO*), ou seja, a soma das despesas necessárias para adquirir, operar e manter a tecnologia funcionando, em aproximadamente 90% a 95% em comparação com os LLMs [Sharma et al. 2024].

Este trabalho tem como objetivo o desenvolvimento e a implementação de agentes de IA [Russell and Norvig 2010, Wooldridge 2001] para execução de tarefas simples. Para isso, será utilizada uma plataforma de automação com baixa produção de código (*low-code*) n8n [n8n-io 2025], combinado com modelos leves como, por exemplo, modelos da família Qwen2.5.

Ao utilizar uma plataforma *low-code*, o alvo da implementação passa a ser a resolução de problemas, simplificando a integração da lógica de IA com fluxos de trabalho e sistemas de gerenciamento de processos de negócios, uma área de crescente importância na ciência da computação [n8n-io 2025, Wooldridge 2001]. Além disso, o trabalho busca fomentar a resolução prática de problemas complexos com SLMs, saindo do domínio puramente teórico para o aplicado, e contribuindo para a democratização da IA.

Este trabalho está organizado da seguinte forma: a Seção 2 avalia os trabalhos relacionados, fundamentando a arquitetura de agentes e a viabilidade dos SLMs. Em seguida, a Seção 3 descreve os Modelos de Linguagem de Baixo Custo (SLMs) e a plataforma de orquestração n8n que utiliza *low-code*. A Seção 4 detalha a implementação e a Seção 5 os resultados dos agentes de automação desenvolvidos para os casos de uso acadêmicos. Por fim, a Seção 6 apresenta a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

[Retamales et al. 2025], por exemplo, investigou a viabilidade de diferenciar automaticamente atendimentos de emergência de atendimentos de rotina. O estudo utilizou o modelo Phi-3 sem treinamento específico, ou seja, o modelo base, para analisar as narrativas clínicas registradas por profissionais de saúde. A validação baseou-se em uma análise retrospectiva de 7.036 visitas, confrontada com um ‘padrão-ouro’ estabelecido por auditoria médica manual em uma amostra estratificada de 384 casos. O modelo demonstrou alta

eficácia na classificação, apresentando sensibilidade de 99,4%, especificidade de 70,4% e acurácia de 95,3% na detecção de urgências sintomáticas frente a demandas de rotina.

Por outro lado, [Fernandes et al. 2025] investigou a viabilidade de realizar o ajuste fino supervisionado (*Supervised Fine-Tuning* - SFT) em modelos de linguagem menores, especificamente o **Llama 2 : 7B** e o **Llama 3 : 8B**, para tarefas de raciocínio jurídico no *Multi state Bar Exam* (MBE). O MBE é a fase de múltipla escolha do exame da ordem dos advogados nos Estados Unidos. A metodologia utilizou um conjunto selecionado de 1.514 questões, cujas respostas foram refinadas pelo modelo **Llama 3 : 70B** para seguir o IRAC, uma estrutura padrão de raciocínio jurídico que organiza a resposta em Problema (*Issue*), Regra (*Rule*), Aplicação (*Application*) e Conclusão (*Conclusion*). A validação utilizou 200 questões do exame MBE de 2022 e avaliou os resultados usando acurácia e a taxa de erros de *parsing* (falhas em seguir a formatação). Concluiu-se que o ajuste fino supervisionado melhora, consideravelmente, a precisão, mesmo com poucos dados.

Em 2024, um segundo estudo [Labrak et al. 2024] comparou a eficácia de modelos baseados no **Mistral 7B Instruct** v0.1 em cenários relacionados à biomedicina. O estudo comparou o desempenho do modelo base com suas variantes especializadas em uma coleção de textos da literatura médica e otimizadas por meio de quantização. O objetivo principal do estudo foi avaliar a capacidade de generalização do modelo, tanto na versão original quanto nas otimizadas. Para a validação, os autores estabeleceram um *benchmark* contendo 10 tarefas de perguntas e respostas do domínio em questão. O desempenho foi avaliado utilizando a acurácia como métrica principal, complementada pelo Erro de Calibração Esperado (ECE) para medir a confiabilidade das previsões e pelo *benchmark TruthfulQA* para verificar a veracidade e a mitigação de alucinações nas respostas geradas. O *TruthfulQA*, é um conjunto de testes projetado para induzir o modelo ao erro, verificando se ele é capaz de distinguir fatos reais de mitos ou falácias, frequentemente encontradas na Internet.

Por fim, [Chen et al. 2025] abordou o uso de SLMs para tarefas de navegação robótica multiponto em dispositivos de borda, visando superar as limitações de latência e privacidade das APIs de grandes modelos. A metodologia realizou um ajuste fino de parâmetros para padronizar as saídas em formato JSON com um módulo iterativo, chamado de *professor-aluno*, onde um LLM robusto (GPT-4) atua como professor fornecendo *prompts* e *feedback* para refinar o raciocínio de SLMs menores, incluindo modelos como **tau-0 .5B**, **TinyLlama-1 .1B**, **h2o-danube-1 .8B** e **openllama-3B**. Para a validação, os autores conduziram experimentos tanto em simulação quanto em ambientes do mundo real utilizando um robô guiado por um SLM para se deslocar por corredores e ambientes complexos. Estes modelos não foram avaliados no trabalho pois não são multi-linguagem.

3. Fundamentos e Conceitos

Modelos de Linguagem de Grande Escala (LLMs) fornecem desempenho de ponta em tarefas de compreensão e geração de linguagem natural, no entanto, esse sucesso tem um custo significativo. Os LLMs usam recursos, de forma intensiva, para treinar e executar seus modelos, exigindo muito esforço computacional e dados o que, frequentemente, implica na necessidade de hardware especializado [Nguyen et al. 2024].

Por outro lado, as despesas operacionais, latência e altos custos dificultam a im-

plementação dos LLMs mais avançados em tarefas especializadas, como no comércio eletrônico.[Licardo and Tankovic 2025]. Além disso, o consumo de energia desses modelos de larga escala tem sido uma preocupação crescente do ponto de vista da IA responsável [Li et al. 2023]. Dessa forma, é crescente o interesse em Pequenos Modelos de Linguagem (*Small Language Models* - SLMs) pois é uma resposta direta aos desafios de uso dos LLMs.

Os SLMs visam manter a acurácia ou adaptabilidade dos modelos maiores operando sob restrições estritas de recursos, como hardware de treinamento ou inferência limitada, largura de banda ou tempo de geração. A definição do que constitui um modelo pequeno é flexível, pois, dependendo do contexto, as classificações são atualizadas. Por exemplo, o modelo GPT-2, com 1.5 bilhão de parâmetros, já foi classificado como *grande* e hoje é menor do que muitos SLMs. O paradigma SLM também enfatiza a importância da qualidade dos dados no treinamento. Modelos como o **phi-1.5:1.3B** (1.3 bilhão de parâmetros) alcançaram desempenho comparável a modelos cinco vezes maiores ao serem treinados em bases de dados de alta qualidade, o que ficou conhecido como *textbook-quality* [Li et al. 2023]. Ou seja, o objetivo principal das SLMs é entregar o máximo possível, com a menor quantidade possível de recursos computacionais.

3.1. Suas vantagens (eficiência, privacidade, custo) e desafios

A adoção de Pequenos Modelos de Linguagem (SLMs) oferece uma resposta prática aos desafios de custo e escala dos LLMs. Contudo, essa abordagem envolve perda de precisão (*trade-off*) nas respostas. As seções a seguir exploram tanto as vantagens operacionais e econômicas que os tornam viáveis, quanto os desafios técnicos e de desempenho inerentes à sua capacidade mais restrita.

3.1.1. Vantagens

Os SLMs são cruciais para a acessibilidade e sustentabilidade da IA em ambientes com recursos limitados [Australian Institute for Machine Learning (AIML) 2024]. Sendo assim, alguns aspectos são muito importantes, por exemplo, a **eficiência e implementação em dispositivos (edge/on-device)**, ou seja, os SLMs são ideais para implantação em ambientes restritos, como **dispositivos móveis**, sistemas embarcados e dispositivos de borda (*edge systems*), em que o uso de LLMs é impraticável devido às limitações de memória e poder de processamento dos dispositivos.

Além disso, realizar o processamento localmente reduz o tempo de espera (latência) que, neste cenário, significa quanto tempo demora para a primeira palavra da resposta ser gerada. Em seguida, é observado o **Custo Total de Propriedade (Total Cost Ownership - TCO)**, que é o quanto custa usar um modelo para um domínio específico. É possível diminuir o TCO de 90% a 95% em comparação com modelos de ponta de propósito geral, com os benefícios de custo se tornando mais evidentes à medida que a escala de implantação se expande; Por fim, a **privacidade**, pois devido a necessidade de garantir a privacidade dos dados, isso faz com que empresas ou setores críticos da sociedade exijam que os modelos sejam implantados em sua própria infraestrutura (*on-premises*), ou seja, o uso de SLMs.

3.1.2. Desafios

Os SLMs, apesar de suas vantagens operacionais, enfrentam desafios críticos relacionados à sua capacidade e implantação. Por exemplo, eles não são capazes de **raciocínio profundo**, ou seja, em tarefas que exigem avaliação profunda das informações ou instruções com várias etapas [Kim et al. 2025]. Geralmente, os SLMs são mais propensos a tarefas baseadas em memorização e heurísticas simples.

Outro fator é a **perda de precisão (trade-off) e dependência de hardware** específico, o que significa que os ganhos de eficiência obtidos por meio de técnicas de otimização, como a quantização¹, estão ligados ao *hardware* e ao ecossistema de software deste *hardware*. Por exemplo, a quantização GPTQ de 4 bits, embora tenha reduzido o uso de VRAM (video RAM) em 41% em uma arquitetura de GPU mais antiga (NVIDIA T4), resultou, de forma inversamente proporcional, em uma piora de 82% nas inferências devido ao excesso de quantização [Licardo and Tankovic 2025].

Por fim, ainda existem os **problemas éticos e de qualidade (alucinação/viéis)**, pois os SLMs enfrentam os mesmos problemas que existem nas LLMs, mas por possuírem bases de dados menores ou precisão de parâmetros reduzidas, estão mais suscetíveis a alucinação² e vieses. [Sakib et al. 2025].

3.2. Introdução a Automação Low-Code e a Arquitetura do n8n

Nesta Seção, iremos abordar a relevância das ferramentas low-code para a criação de fluxos de trabalho (*workflows*) e a automação de processos. A plataforma [n8n-io 2025] combina a automação low-code com capacidades de Inteligência Artificial, permitindo que o desenvolvimento se concentre na resolução de problemas utilizando blocos de ações. Essa abordagem simplifica a integração da lógica dos SLMs com os fluxos de trabalho, possibilitando que o projeto seja direcionado na implementação de agentes de IA.

A *Automação Low-Code* (LCA) permite que o desenvolvimento de soluções complexas seja realizado com mínimo esforço de codificação, utilizando interfaces visuais. O n8n se insere neste contexto como uma ferramenta de automação de fluxo de trabalho que combina capacidades de IA com automação de processos de negócios. O n8n adota o tipo de licença *fair-code*, ou seja, este modelo busca combinar a transparência do código aberto, que permite a visualização e modificação, com restrições contratuais que protegem o modelo de negócios do autor [n8n-io 2025].

O n8n opera sobre uma estrutura de *workflows* e nós. Ou seja, um *workflow* é definido como uma coleção de nós para automatizar um processo. Cada nó é um componente individual que realiza uma tarefa específica e quando estes nós são encadeados, geram um fluxo de trabalho que realiza uma atividade. O n8n é uma plataforma moderna e utiliza os dados organizados em JSON (formato de armazenamento de dados de forma hierárquica,

¹Técnica de otimização que reduz a precisão dos parâmetros com o objetivo de tornar o modelo menor, mais rápido e com menor consumo de memória. Em um modelo que utiliza parâmetros de 32 bits (FP32) ou 16 bits (FP16) em ponto flutuante, esse valor é reduzido, usando quantização, para números inteiros de 8 bits (INT8) ou menos. Um modelo com FP32 convertido para INT8 pode reduzir em até 75% de tamanho.

²Geração de uma resposta que parece plausível, mas que é incorreta, inventada, ou inconsistente com o contexto.

como um dicionário) e permite otimizações utilizando linguagens como JavaScript ou Python.

Contudo, o n8n não é apenas uma ferramenta para a criação de *workflows* simples, ele permite o uso de nós especializados com integração com modelos de inteligência artificial como por exemplo o ChatGPT [Openai 2025] ou o Ollama [Ollama 2025]. O uso desses nós especializados, conhecidos como *agentes*, são diferentes das cadeias de processamento de fluxos (quando um nó é encadeado com outro para realizar apenas tarefas simples). Ou seja, enquanto uma cadeia segue uma sequência predeterminada de chamadas, um agente usa um modelo de linguagem para determinar quais ações tomar, atuando como um tomador de decisões, dentro um conjunto pré-determinado de escolhas.

Além disso, agentes são a única opção se o *workflow* precisa ter memória, ou seja, precisa lembrar do contexto, o que é essencial para manter conversas contínuas e naturais entre sistemas e usuários, por exemplo. Mas o n8n não está limitado apenas a cadeias ou agentes, ele possui ferramentas que agem como extensões que podem adicionar, além de contexto e memória, acesso a APIs de várias ferramentas públicas, bancos de dados ou mesmo outras cadeias do próprio n8n, gerando fluxos de trabalho aninhados complexos.

4. Metodologia

Neste trabalho foram implementados quatro fluxos de trabalho com o objetivo de determinar a capacidade e funcionalidade, na prática, de modelos SLM. Todos os fluxos foram executados em um servidor Linux com Debian 12, kernel 6.1.0, Docker 29.0.2. O Docker [Docker 2023], foi utilizado para permitir a simplificação da integração das ferramentas e tornar as soluções implementadas portáveis entre sistemas. Além disso, as SLMs foram executadas em uma placa NVIDIA Quadro P1000, com 4GB de VRAM, com driver na versão 535.247.01 e o CUDA na versão 12.2.

A especificação dos drivers, e versões, é importante nesse cenário, pois a cada nova versão, atualizações podem mudar os resultados obtidos. Além disso, todos os sistemas utilizaram suas versões em contêiner e o dispositivo NVIDIA foi utilizado utilizando a técnica de *passthrough* de encaminhamento direto do hardware para o contêiner.

Como descrito anteriormente, foi utilizado o sistema n8n para criação dos fluxos de tarefas em *low code*. Para hospedagem dos modelos de IA SLM, utilizou-se o OLLama. O OLLama é responsável por hospedar e gerenciar o acesso às SLMs. Por exemplo, nele foram instalados os SLMs:

Tabela 1. Modelos usados no trabalho

Nome	Versão	Qtd. de Parâmetros
Qwen	2.5	0.5B
Qwen	2.5	1.5B
Qwen	2.5	3B
Llama	3.2	1B
Llama	3.2	3B
Phi	3	3.8B

Neste cenário, o número n em nB , significa o número em bilhões de parâmetros.

Todos os modelos adicionados ao Ollama possuíam menos de 4Gb para que fosse possível sua execução totalmente na VRAM da placa de vídeo, reduzindo o gargalo de execução. Contudo, o OLLama permite o uso de modelos maiores, segmentando uma fração do modelo na memória RAM do computador e outra fração (até o limite da VRAM), alocado na memória da placa de vídeo. Essa fragmentação pode gerar um maior *trade off* ao gerar as respostas do modelo as interações.

Contudo, a fim de testar o potencial de integração do n8n, foram utilizados dois sistemas públicos: o Google Email e Agenda e o Telegram. O Google Agenda e o Gmail foram utilizados como ferramentas-alvo (atuadores) para os agentes de agendamento e verificação de trabalhos enviados (seu uso será detalhado nas seções seguintes).

Como o n8n possui suporte a nós para serviços como Google Agenda e Gmail, eles podem ser usados diretamente nos *workflows*, ou usados como ferramentas que os agentes de IA usam para realizar uma ações. De forma prática, um email pode ser enviado e isso disparar uma ação de um nó no n8n que vai consultar o SLM, via OLLama, e determinar se deve ou não agendar um compromisso ou responder ao email recebido.

Por outro lado, o Telegam foi utilizado para testar as funcionalidades de chatbot, uma vez que sua API é gratuita e só precisa que um token seja gerado na ferramenta deles para criação de um bot de interação com usuários. Ou seja, ele atua como um canal de *comunicação entre agentes*, pois ele permite que o sistema (n8n) receba entradas (comandos do usuário, atuando como sensor, e envie respostas aos agentes (operações de *chat* ou *callback*). As seções a seguir serão divididas em cada uma dos quatro fluxos de atividades que foram criados para testar e avaliar o uso do n8n e os modelos de SLM.

4.1. Casos de Uso

A fim de avaliar a capacidade de uma SLM em diversos cenários, foram definidos quatro problemas e como seria possível usá-las para solucioná-los. Contudo, antes de avançar para os exemplos é necessário entender como funcionam os fluxos do n8n.

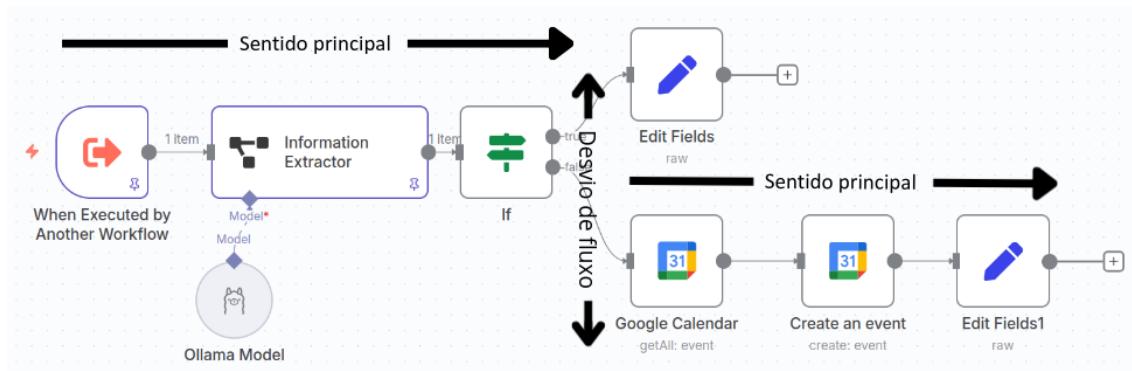


Figura 1. Sentido da execução de um fluxo no n8n.

A Figura 1 apresenta o fluxo de como os dados são processados. Embora seja possível ocorrerem bifurcações, no geral, eles sempre vão iniciar no nó mais a esquerda até o nó final, mais a direita. Ou seja, o fluxo se inicia no nó de nome “*When Executed by Another Workflow*”, que só é acionado por meio de um “gatilho” (*trigger*) e, após

iniciado, segue para o Agente de IA *Information Extractor* e assim suscetivamente, até alcançar um nó terminal. O nó AI Agent atua como um orquestrador dentro do fluxo de automação. Diferente dos nós convencionais que processam dados linearmente, o Agente estabelece, dinamicamente, uma topologia estrela durante sua execução. Ao ser acionado, o nó compila a entrada do usuário junto a uma instrução estrutural (*System Message*) e envia esses dados via API para o modelo de linguagem (neste caso, o OLLama).

Em seguida, o modelo analisa a solicitação e determina se é necessária uma ação externa. Caso isso seja verdadeiro, o Agente interrompe a geração de texto e aciona a ferramenta (*tool*) correspondente. No contexto do n8n, essas ferramentas operam frequentemente como subfluxos autônomos. Ou seja, o processamento é desviado para um novo fluxo independente e, após a lógica do fluxo ser executada, o resultado é retornado (*output*) ao nó do Agente. Com este retorno, o Agente consulta o modelo novamente para sintetizar a resposta final. O ciclo encerra-se apenas quando o modelo julga que a tarefa foi concluída, liberando o fluxo para prosseguir para o próximo nó.

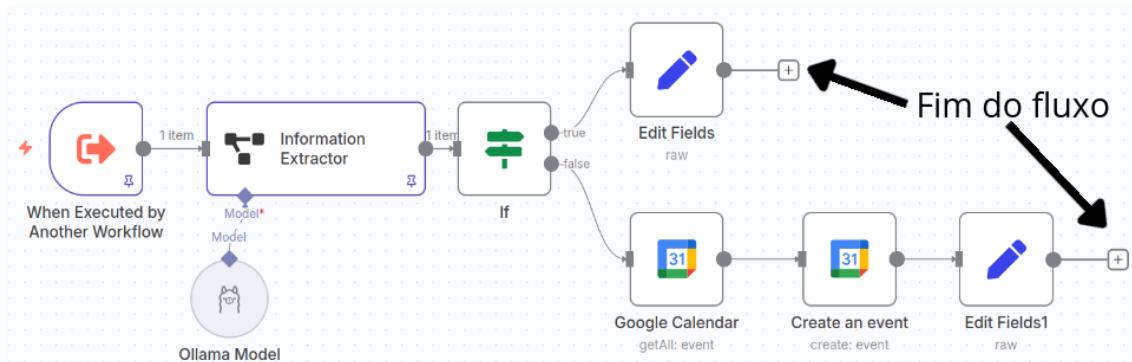


Figura 2. Os nós não conectados no final indicam o fim de fluxo no n8n.

Nas Figuras 1 e 2 o fluxo poderá ser desviado em uma única bifurcação, o nó If, onde a álgebra booleana determina o desvio do fluxo dependendo se a condição for satisfeita (*True*) e, neste caso, a execução segue pela ramificação superior ou, caso contrário (*False*), pela inferior, ativando sequencialmente os nós da rota selecionada. Ao atingir o último nó da cadeia, o sistema consolida todos os metadados processados e filtrados em um objeto JSON, que assume um comportamento distinto dependendo do contexto. Ou seja, se este fluxo operar como um subfluxo (uma ferramenta do Agente), esse JSON é retornado imediatamente ao nó principal para dar continuidade ao processamento da IA; caso contrário, se for um fluxo independente, a execução se encerra.

4.1.1. Agendamento de Horário do Professor

O primeiro caso de uso foi o de geração de um fluxo capaz de avaliar a agenda de trabalho de um professor e interagir com o usuário a fim de conseguir agendar reuniões dentro das janelas pré-definidas. Neste cenário, o objetivo era que por meio de uma interação de chat; informando dados como email, nome, e horário desejado; um acadêmico pudesse requisitar o agendamento de forma automatizada em dias e horários pre-definidos, mas sem causar colisão com outros agendamentos prévios.

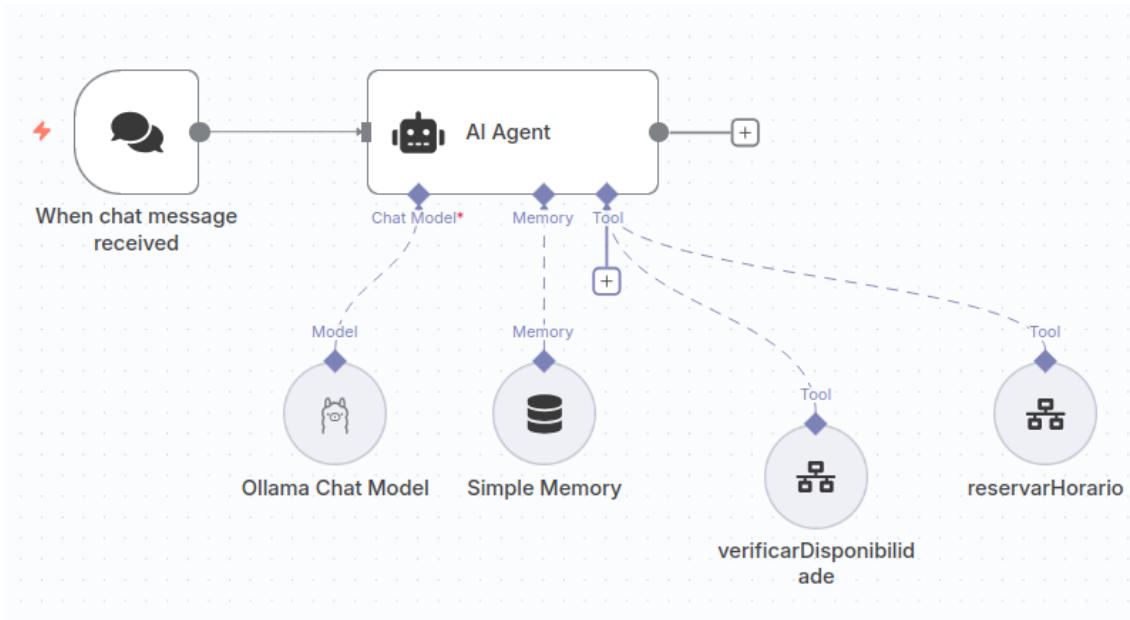


Figura 3. Fluxo principal de execução do agente de agendamento de reuniões.

A Figura 3 mostra como o processo deve ocorrer. Primeiro, o acadêmico interage com o chat. Ao realizar as requisições, o agente de IA interage com este aluno para entender a sua demanda. Neste cenário, o agente de IA interpreta as solicitações em linguagem natural, por exemplo, *Quero marcar uma reunião com o professor semana que vem*. Para lembrar de interações anteriores ou salvar o estado de uma conversa uma memória simples é utilizada (*Simple Memory*) e o agente de IA é vinculado a API do Ollama.

Dentre os vários modelos de SLM disponíveis, o modelo **Qwen2 . 5 : 3B** foi o que se manteve mais assertivo entre as várias rodadas de execução e testes. Ele demonstrou uma capacidade robusta em seguir as instruções da *systemMessage* (mensagens que são utilizadas para instruir como a SLM deve se comportar), desde que elas sejam extremamente claras e diretas como, por exemplo: *Nunca pule nenhum passo*. Uma vez que a interação inicial foi realizada, o agente de IA usa as ferramentas disponíveis (*Tool*). No cenário descrito, foram definidas duas ferramentas: *verificarDisponibilidade* e *reservarHorario*.

O desempenho esperado só foi atingido mediante o envio de diretrizes estritas e livres de ambiguidade ao modelo via *System Message*. Além da objetividade, verificou-se que os SLMs dependem de exemplos concretos para compreender corretamente os cenários de uso das ferramentas e o fluxo de conversação. Esse conjunto de características define o que foi caracterizado neste trabalho como '*'prompts rígidos'*', uma abordagem que será utilizada na construção dos comandos nos próximos casos de uso.

A primeira ferramenta, *verificarDisponibilidade*, acessa a Agenda do Google e verifica se no horário desejado já existe uma reunião prévia marcada ou não. Em ambos os casos, a ferramenta *reservarHorario* só será invocada se existir um horário disponível. Como descrito anteriormente, o **Qwen2 . 5 : 3B** usa instruções simples e diretas. No caso, a instrução completa definida foi: *Só execute reservarHorario*

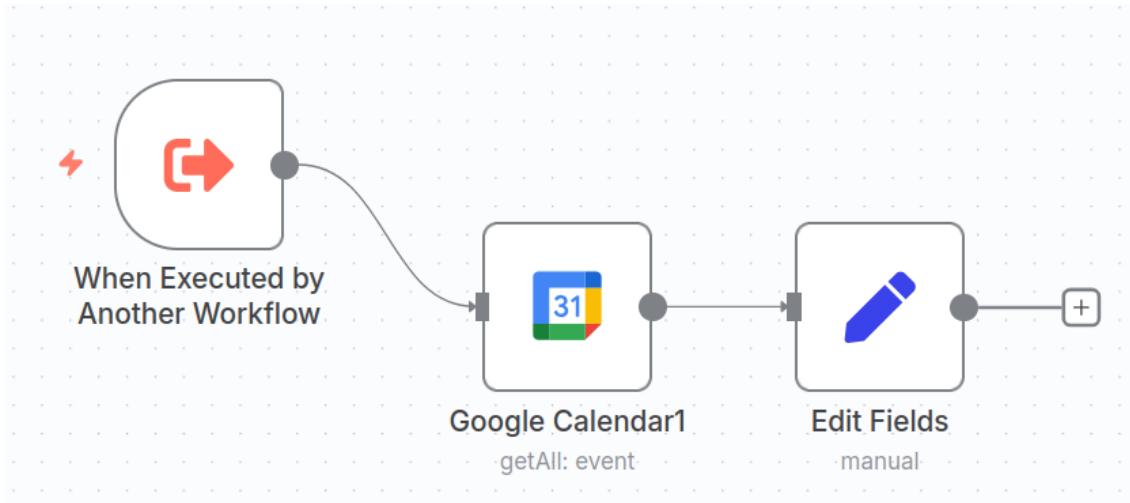


Figura 4. Sub-Fluxo de Execução (Subworkflow) do Agente de agendamento de reuniões VerificarDisponibilidade

após o aluno confirmar os dados. Além disso, o Agente usa a hora atual (`$now`) do servidor como contexto extra.

Cada uma das ferramentas descritas são definidas, recursivamente, como subfluxos que são invocados pelo fluxo principal. A Figura 4 mostra o fluxo executado para verificar os horários já agendados. O subfluxo é simples porque ele só retorna para o fluxo principal os horários já ocupados no dia desejado e os horários disponíveis futuros (mechanismo de sugestão).

Por outro lado, o agendamento em si é um subfluxo bem mais complexo. A Figura 5 mostra os nós utilizados para validação e inserção de um novo agendamento para o professor.

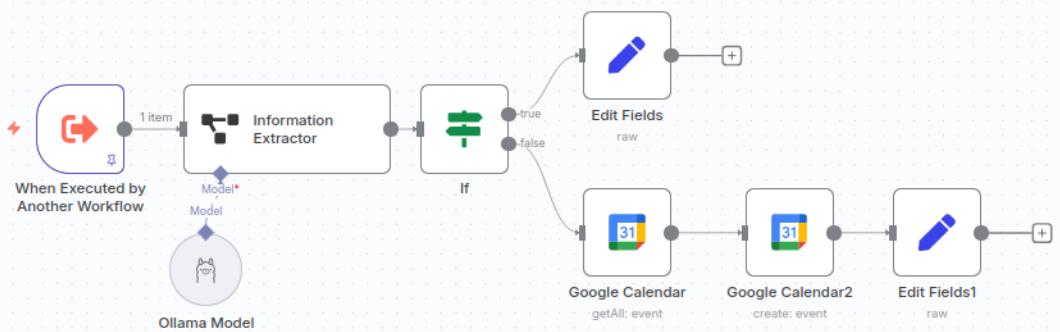


Figura 5. Sub-Fluxo de Execução (Subworkflow) do Agente de agendamento de reuniões ReservarHorario

Uma vez que a ferramenta anterior retornou True para o contexto do acadêmico, este subfluxo extrai as informações utilizando um nó Extractor `Information` vinculado ao Ollama,. A separação das informações não ocorre por regras simples de texto,

mas sim por meio da imposição de um esquema de dados (schema) ao modelo de linguagem. O nó restringe a saída probabilística do Modelo SLM a um formato estruturado pré-definido (JSON) forçando o modelo a atuar como um extrator estruturado, transformando a linguagem natural em um objeto JSON contendo estritamente os atributos: nome, email, assunto e data e hora.

Após a extração, um nó condicional (`If`) realiza a validação dos dados, verificando se algum campo obrigatório retornou vazio ou nulo, nesse caso um objeto JSON contendo `{"resultado": false}` será retornado. Se a validação for bem-sucedida, o fluxo verifica uma última vez a disponibilidade no Google Calendar e realiza a inserção do evento. Por fim, um objeto JSON com resultado `true` é retornado ao agente principal que, ao receber esse retorno de sucesso no agendamento, devolve uma resposta positiva ao acadêmico. Vale ressaltar que, neste fluxo, o primeiro nó do Google Calendar recupera todos os horários da agenda alocados para a aplicação a partir do horário definido pela IA. Dessa forma, o próximo nó do Google Calendar irá alocar uma reunião no próximo horário livre.

Por fim, durante a fase experimental, observou-se que o Agente de IA utilizado pelo nó `Information Extractor` conseguiu ser mais estável na extração de saídas estruturadas e padronizadas do que o Agente de IA, mitigando erros de sintaxe e garantindo a integridade dos dados ao longo das execuções.

4.1.2. Lista de Tarefas Seguindo Regras (Pomodoro)

O segundo problema foi criar um mecanismo para gerenciar atividades utilizando como base a técnica Pomodoro [Cirillo 2018]. O objetivo era receber uma lista de tarefas e organizá-las em uma Agenda (neste caso foi utilizada a do Google), seguindo a metodologia Pomodoro³ para a alocação de tempo.

A Figura 4 apresenta o fluxo de execução principal deste agente. O processo se inicia com o recebimento de mensagens via Telegram (interação por meio de chatbot). Como os modelos SLM são mais restritos, é necessário evitar o uso de caracteres fora do padrão ASCII (como acentos e caracteres especiais). Dessa forma, ao receber a mensagem via Telegram, é executado um código que normaliza o texto, ou seja, ele remove os caracteres como acentos e caracteres especiais antes de enviar a mensagem para o modelo. Nos testes realizados, isso reduziu alucinações e erros de avaliação do modelo.

Neste cenário, temos a atuação estrita do agente como um Roteador de Informações. Diferente de um chatbot, este agente foi configurado com uma *System Message* objetiva e direta com o objetivo de analisar a entrada do usuário e direcioná-la para a ferramenta (*Tool*) correta. As ferramentas disponíveis, no caso desta automação, são subfluxos para tarefas específicas como: `addTask` (adicionar tarefas), `getTaskList` (visualizar a lista de tarefas), `delTask` (remoção de tarefas), `changePriority` (reordenar e mudar a prioridade de uma tarefa) e `None` (uso genérico).

Quando o agente principal identifica uma entrada como uma tarefa a ser adici-

³A metodologia Pomodoro define que atividades devem ser realizadas em janelas de 25 minutos com intervalo de 5 minutos durante quatro sessões e um intervalo de 20 minutos após essas quatro sessões, antes da próxima.

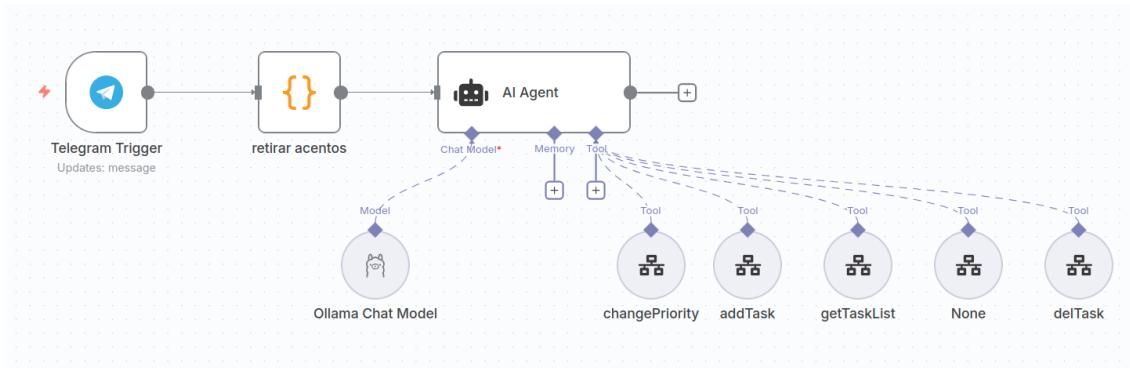


Figura 6. Fluxo de principal do agente de gerenciamento de tarefas (Pomodoro).

onada, um JSON RAW (sem modificações) com a entrada do usuário é enviada para a ferramenta `addTask`. Ao receber esse dado, por meio do nó *Information Extractor* (usando o modelo **Qwen2 .5 : 3B**), um esquema de dados JSON, que define os campos estruturados da ação (como resumo da tarefa, prazo e prioridade), é usado para extrair a descrição da tarefa da mensagem.

Em seguida, antes de inserir os dados da tarefa em uma planilha eletrônica (nó *Google Sheets*), é realizado um processamento para recuperar as tarefas que já existem nessa planilha. Neste momento, algumas decisões precisam ser tomadas. Ou a tarefa já existe ou é ela é uma nova tarefa. Se a tarefa for nova, o fluxo é redirecionado para a inserção de uma nova tarefa ao final da planilha eletrônica (*Append row in sheet*). Uma vez que a nova tarefa foi adicionada na planilha, o subfluxo **CompletaAgendaPomodoro** é invocado para encerrar a execução. Por outro lado, se a tarefa já existir, a saída é mais simples, apenas informando ao usuário que uma tarefa com aquele nome já existe na planilha, evitando duplicações de atividades.

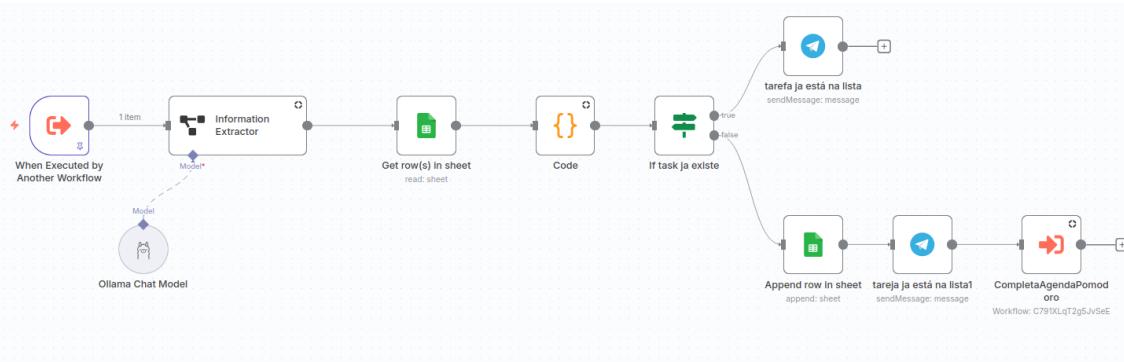


Figura 7. Sub-Fluxo de Execução (Subworkflow) do Agente de Gerenciamento de Tarefas (Pomodoro) ListarTarefas

Assim, quando o subfluxo **CompletaAgendaPomodoro** é invocado, ele reconstrói a sequência de atividades da agenda conforme novas tarefas são adicionadas, retiradas ou alteradas. No primeiro passo são recuperados todos as tarefas utilizando a API do Google Calendar e Sheets. Em seguida, todas as tarefas que já foram executadas são removidas da agenda.

Após os nós do tipo *code* transformar a lista de tarefas em uma lista numerada (JSON), o cruzamento dos dados da lista de tarefas e do calendário ocorre no nó *Merge* que unifica as tarefas a fazer e define as restrições como, por exemplo, os horários ocupados. Infelizmente, alguns tratamentos do n8n não são totalmente livres de programação. Neste exemplo desenvolvido, foi necessária a implementação de um conversor utilizando JavaScript.

A alocação das tarefas seguiu as seguintes restrições:

1. A alocação das tarefas deve ocorrer entre 09:00 e 17:00;
2. Não alocar tarefas entre 11:00 e 13:00 (horário de almoço);
3. Utilizar somente dias úteis;
4. Se o horário proposto colidir com um evento existente, o algoritmo avança para o início da tarefa para o final desse evento conflitante;
5. Cada tarefa é alocada em um bloco de 30 minutos (25 minutos de foco + 5 minutos de pausa), conforme a técnica Pomodoro.

Após o cálculo dos horários, os dados passam pelo nó *fuso*. Esta etapa de tratamento de dados é necessária para corrigir a formatação das strings de data geradas pelo JavaScript, removendo milissegundos e garantindo que o fuso horário local (GMT-04) seja, corretamente, utilizado. Por fim, o nó Google Calendar (*Create an event*) cria os eventos na Agenda com o sufixo [POMODORO], permitindo fácil identificação visual.

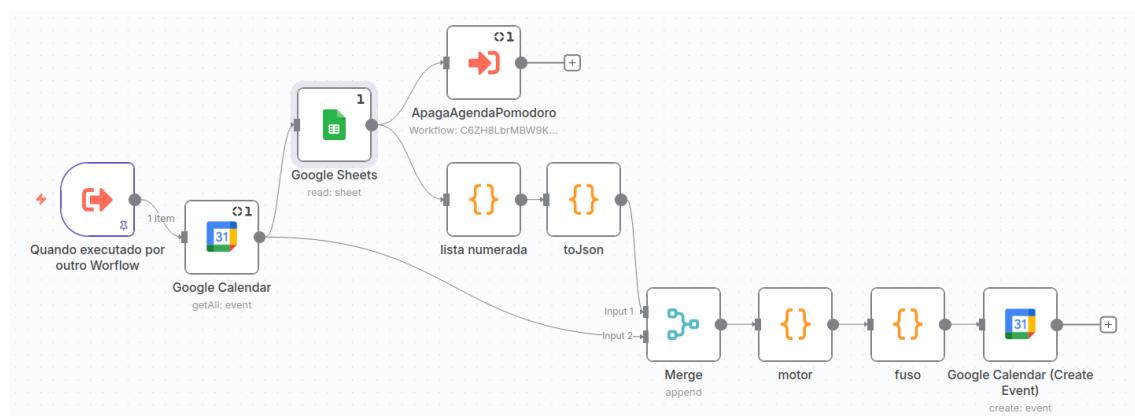


Figura 8. Sub-Fluxo de Execução (Subworkflow) do Agente de Gerenciamento de Tarefas (Pomodoro) CompletaAgendaPomodoro

Por outro lado, o subfluxo **ApagaAgendaPomodoro** começa realizando uma operação `getAll`, que faz uma varredura na agenda do usuário. Essa busca (*query*) utiliza uma restrição específica para garantir que outros agendamentos não sejam alterados. Ela busca, especificamente, pela string [POMODORO].

Uma vez recuperados os objetos dos eventos, o fluxo encaminha seus identificadores únicos (IDs) para o nó subsequente, *Delete an event*. Este nó realiza a exclusão iterativa de cada item listado. A existência deste fluxo de limpeza é pré-requisito para o funcionamento da alocação, pois permite que a grade de horários seja recalculada do zero a cada nova interação, evitando a duplicidade de tarefas e garantindo que a agenda esteja, corretamente, no estado mais atual da lista de prioridades.

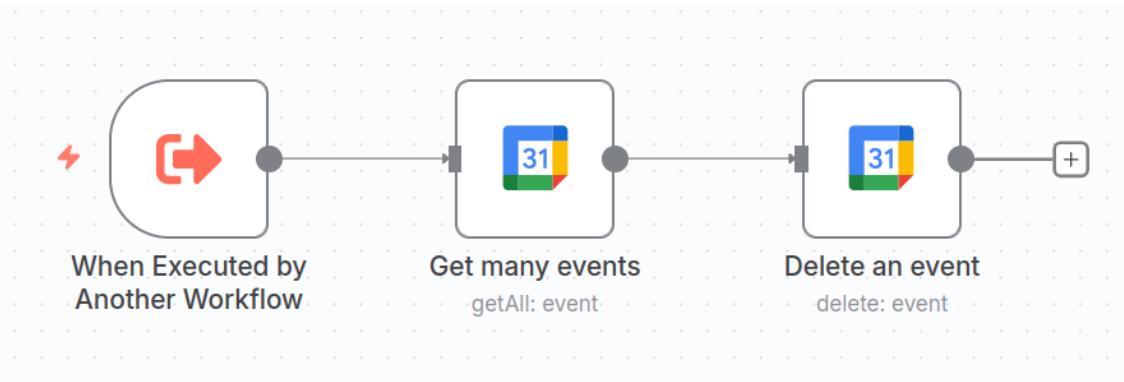


Figura 9. Sub-Fluxo de Execução (Subworkflow) do Agente de Gerenciamento de Tarefas (Pomodoro) ApagaAgendaPomodoro.

Em seguida, no fluxo **changePriority**, um nó condicional *If* verifica se a mensagem recebida do fluxo principal inicia com “c/”. Em caso negativo, o subfluxo **retornarListaTarefas** é acionado e uma mensagem informando a maneira correta de trocar prioridades é enviada ao usuário por meio do Telegram. Por outro lado, um nó de Código *lista numerada* recebe a lista atual por meio do nó *Google Sheets* (operação *Get row(s) in sheet*) e os índices fornecidos pelo usuário (ex: “c/ 1 2”), o nó transforma as tarefas em uma lista numerada. No próximo passo, o nó *trocarPrioridade* realiza a troca de posições na lista de acordo com sua numeração e os números recebidos na mensagem do fluxo principal. Após isso, uma mensagem é enviada ao usuário pelo Telegram informando que a troca de prioridades foi realizada. Com isso, o último passo é a tabela ser limpa (*Clear sheet*), e uma nova lista de tarefas é adicionada na planilha (*Append row in sheet*), finalizando a reconstrução do calendário.

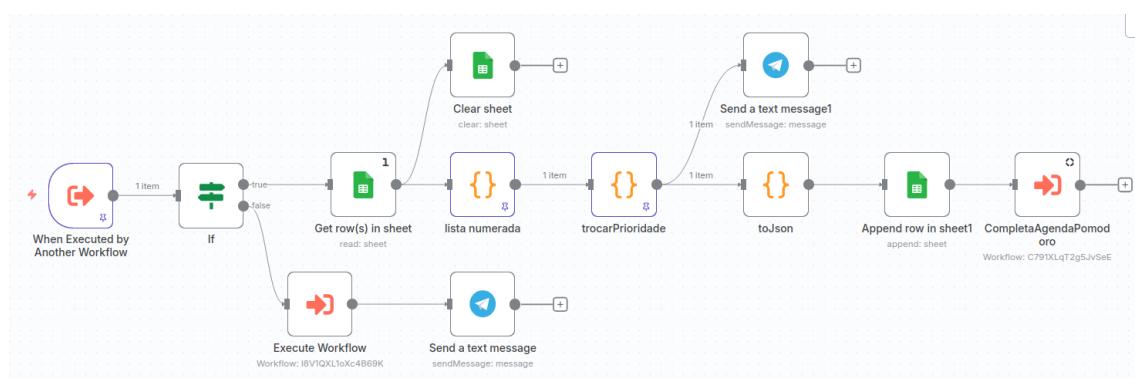


Figura 10. Sub-Fluxo de Execução (Subworkflow) do Agente de Gerenciamento de Tarefas (Pomodoro) AlterarPrioridade

O subfluxo **delTask** opera de maneira semelhante ao **changePriority**. Após o nó **IF** verificar se a string recebida começa com **r** (ex: **r/ 2 4**). Se a resposta for negativa, ele retorna uma mensagem ao usuário pelo **Telegram**. Caso contrário, converte o vetor de tarefas original em um conjunto de números e usa esses números como filtro, excluindo os itens selecionados e renumerando os remanescentes, sequencialmente. O fluxo segue com uma mensagem no **Telegram** informando a exclusão da tarefa (ou tarefas) e, por fim, o subfluxo **CompletaAgendaPomodoro** é invocado.

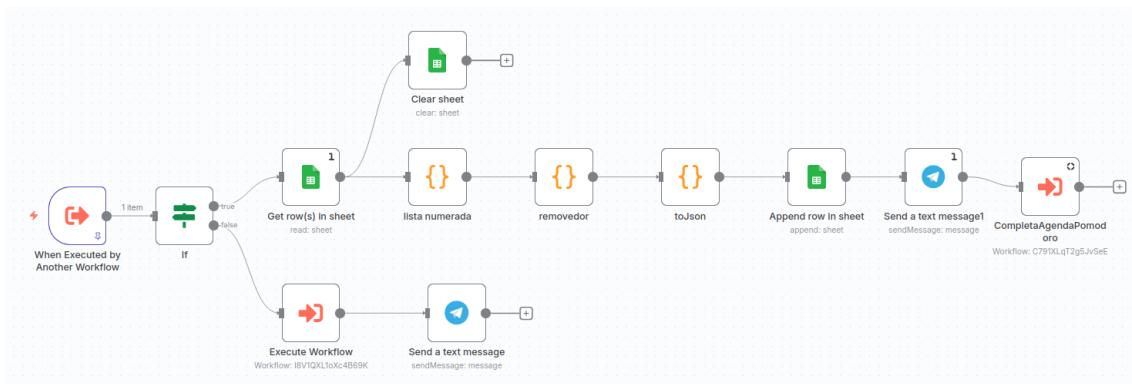


Figura 11. Sub-Fluxo de Execução (Subworkflow) do Agente de Gerenciamento de Tarefas (Pomodoro) delTask

A outra ferramenta disponível é a **getTaskList** que aciona o subfluxo **RetornarListaTarefas**. Ele atua como um mecanismo de *feedback* e recuperação de estado para o usuário. Sua função é informativa, ou seja, ele é acionado quando o agente principal identifica a intenção de visualização (ferramenta **getTaskList**) ou quando algum outro fluxo precisa retornar a lista de tarefas ao usuário. O fluxo começa com a recuperação da lista de tarefas usando para isso o nó **Google Sheets** na operação **Get row(s) in sheet**.

Dessa forma, quando os retornos são processados, eles são transformados pelo nó **Code** que gera a saída para o nó do **Google Sheets**. Essa indexação fornece ao usuário, tanto o estado da planilha, que funciona como um banco de dados, como os identificadores numéricos necessários para executar comandos subsequentes de manipulação, como remover ou reordenar tarefas. Por fim, o nó **Telegram** recebe esta *string* tratada e a envia ao usuário.

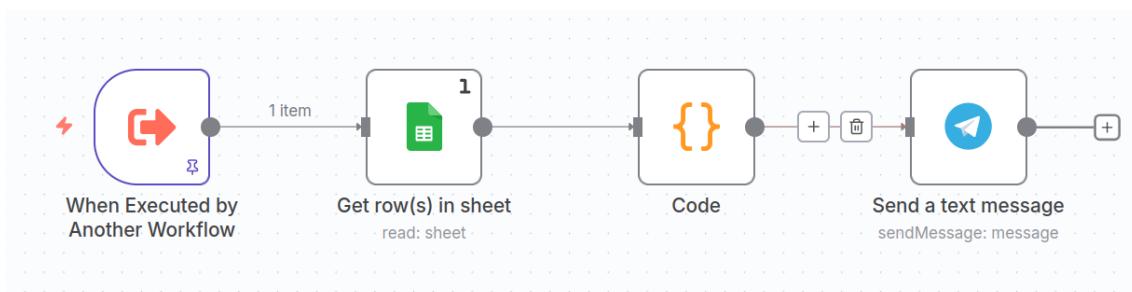


Figura 12. Sub-Fluxo de Execução (Subworkflow) do Agente de Gerenciamento de Tarefas (Pomodoro) RetornarListaTarefas

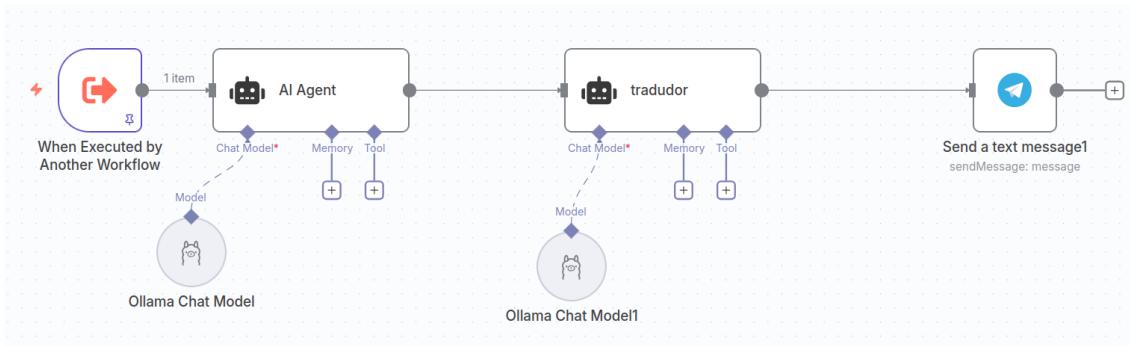


Figura 13. Sub-Fluxo de Execução (Subworkflow) do Agente de Gerenciamento de Tarefas (Pomodoro) None

Por fim, o subfluxo **None** atua como um mecanismo de tratamento de exceções e interface de ajuda, sendo acionado quando o agente roteador não identifica um comando de ação explícito (como adicionar ou remover tarefas). Ou seja, esse é o gatilho de *fuga*, caso nenhuma ferramenta seja selecionada, qualquer que seja o motivo.

A Figura 13, possui dois estágios e ambas envolvem o uso de modelos SLM. No primeiro estágio, o nó do agente de IA foi configurado com uma persona⁴ configurado como um *Assistente de Triagem de Diálogo*. Neste primeiro estágio foi utilizado o modelo **Qwen2 .5 : 3B**. Este agente possui uma (*System Message*) estrita e sua função é puramente analítica, ou seja, ele classifica a intenção da mensagem do usuário em categorias pré-definidas (ex: saudação, pedido de ajuda, relato de erro, etc.) e gera uma resposta padronizada em inglês, orientando o usuário sobre a sintaxe correta dos comandos disponíveis no sistema.

Em seguida, o segundo estágio introduz o agente *tradutor*. Este nó recebe a resposta gerada pelo estágio anterior e utiliza um modelo diferente, o **Llama3 .2 : 3B**, com a tarefa, exclusiva, de realizar a tradução do texto de Inglês para o Português do Brasil. Essa divisão de responsabilidades foi necessária pois, após testes empíricos, usar o prompt do primeiro modelo em inglês era mais efetivo do que utilizá-lo em inglês.

Por fim, o nó *Telegram* envia a resposta traduzida ao usuário, fechando o ciclo de *feedback* e garantindo que interações fora do escopo de execução sejam tratadas de forma instrutiva e natural. Dessa forma, para algumas situações o uso de estratégias híbridas entre múltiplos agentes de IA podem gerar melhores resultados.

4.1.3. Ler Documentos em Busca de Padrões de Interesse

O terceiro caso de uso foi realizar a “leitura” de um documento em PDF (no caso o Boletim Diário da Universidade), por menções a pessoas (servidores, alunos) ou temas específicos (tipos de projetos, compra de equipamentos, etc.), evitando a necessidade de leitura manual completa do documento para encontrar esses assuntos e, possivelmente, informando ao usuário quando estes padrões ocorriam.

⁴Perfil de personagem que você atribuiu ao modelo para moldar o seu estilo de comunicação, personalidade, objetivos ou restrições e conjunto de habilidades.

A Figura 14 mostra o seu único fluxo de funcionamento. Ele implementa uma arquitetura de processamento de documentos longos baseada em filtragem prévia, que utiliza expressões regulares⁵ e summarização generativa⁶. Este caso de uso, em especial, devido a limitações do site que fornece o PDF a ser processado, exige o uso de um gatilho. Esse gatilho ocorre por meio de um *Telegram Trigger* que espera o envio de um arquivo (PDF do boletim diário) acompanhado de uma legenda. Esta legenda atua como a variável de filtragem, contendo a palavra-chave de interesse.

Após a conversão do binário do PDF em texto pelo nó *Extract from File*, o fluxo de dados entra em uma etapa de pré-processamento que é executada por três nós de código consecutivos, visando contornar as limitações de janela de contexto dos SLMs.

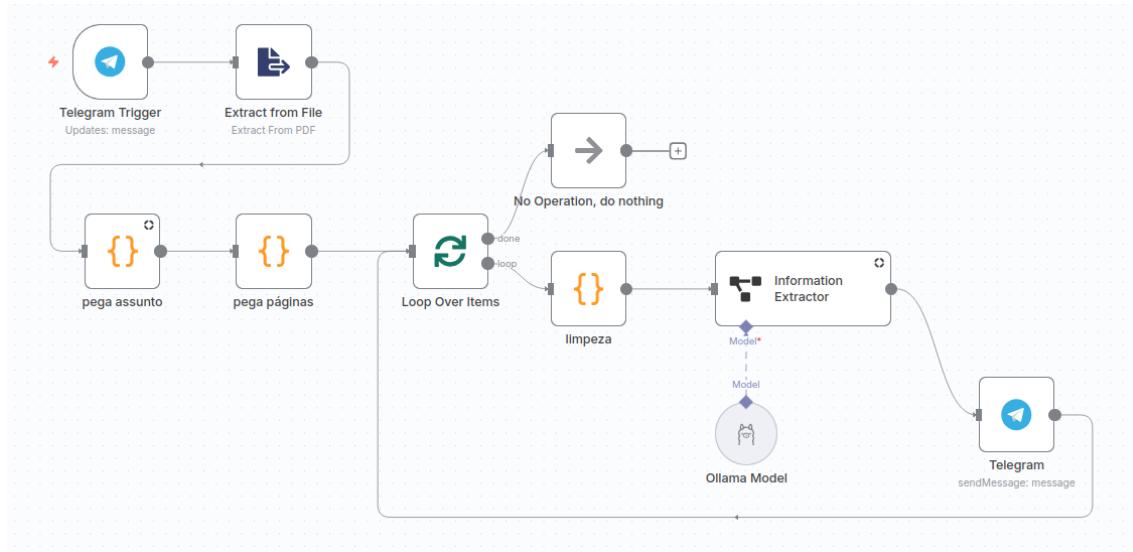


Figura 14. Fluxo de Execução (Workflow) do Leitor de Documentos PDF.

O primeiro nó de código, *pegar assunto*, aplica as expressões regulares para segmentar o texto contínuo em blocos lógicos de atos administrativos (iniciados pelo padrão de data e termos específicos). O algoritmo itera sobre esses blocos e descarta imediatamente qualquer segmento que não contenha a palavra-chave fornecida pelo usuário, reduzindo o volume de dados.

Em seguida, o nó *pegar páginas* realiza a extração dos metadados. Ele varre os blocos filtrados em busca de marcadores de paginação e executa uma normalização numérica, convertendo a string encontrada em números inteiros e aplicando um incremento unitário ($n + 1$) para corrigir a discrepância de indexação entre o texto extraído e a visualização do PDF.

Além disso, antes de enviar o texto para a SLM, o nó *limpeza* aplica uma técnica de otimização de *inputs* (entradas). Caso o texto exceda 2000 caracteres, um algoritmo de truncagem preserva o cabeçalho (as primeiras 30 linhas), o rodapé (as últimas 10 linhas) e uma janela de contexto ao redor da palavra-chave, descartando o *meio* irrelevante. Isso assegura que o modelo **Owen2 .5 : 3B**, utilizado pelo nó *Information Extractor*, receba as

⁵Uma sequência de caracteres que define um padrão que pode ser utilizado para procurar textos.

⁶Processo de criar um resumo baseado em um texto de entrada.

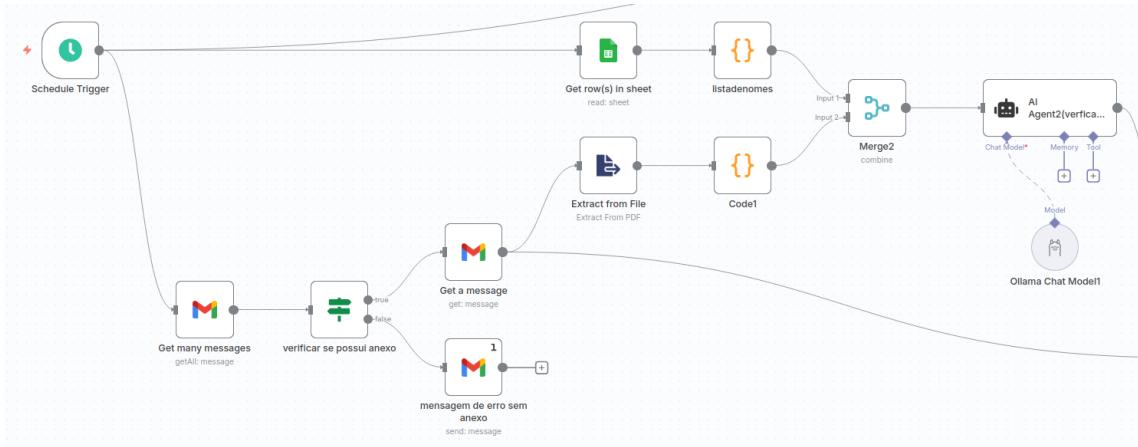


Figura 15. Fluxo de Execução (Workflow) do Agente de Conferência de Trabalhos Acadêmicos (Parte1).

informações essenciais para extração de dados e com o contexto necessário para entender o assunto tratado, maximizando a precisão da sumarização.

Por fim, o fluxo itera sobre os resultados processados (*Loop Over Items*) e envia um resumo estruturado para o usuário via Telegram, contendo o ato e a página de referência. Isso encerra o fluxo do caso de uso de busca de padrões em documentos muito grandes, como são, por exemplo, os boletins de serviço, o diário oficial da união ou livros em PDF.

4.1.4. Resposta Automatizada de Tarefas de Acadêmicos

O quarto problema foi o de receber os trabalhos de alunos por meio de uma conta de email e realizar a conferência básica, ou seja, se foi enviado e se possui conteúdo pertinente ao tema, etc. A solução deste problema utiliza um único fluxo e, por este motivo, se tornou muito longo e difícil de ser retratado em uma única imagem. Desta forma, ele foi em três partes (Figuras 15, 16 e 17) para melhor visualização e compreensão de seu funcionamento.

O fluxo desenvolvido implementa uma arquitetura de monitoramento ativo, diferindo dos agentes anteriores que operavam por gatilhos de eventos diretos. A execução é regida pelo nó *Schedule Trigger*, configurado para despertar o sistema em intervalos regulares de 10 segundos, garantindo a verificação quase em tempo real de novas submissões a uma conta de email do Gmail.

O processamento começa por meio do nó *Get many messages*, que consulta a API do Gmail filtrando mensagens não lidas com o assunto "[checkbot]" recebidas nas últimas 24 horas. O fluxo segue para uma verificação usando um nó *if verificar se possui anexo*, que inspeciona o cabeçalho MIME (*multipart/mixed*). Caso o e-mail não contenha anexo, o sistema desvia para um fluxo de rejeição (*mensagem de erro sem anexo*), notificando o aluno imediatamente.

Confirmada a presença do arquivo, condição *true* do nó *if verificar se possui anexo*, o nó *Get a message* realiza o download do binário em anexo ao email, que é

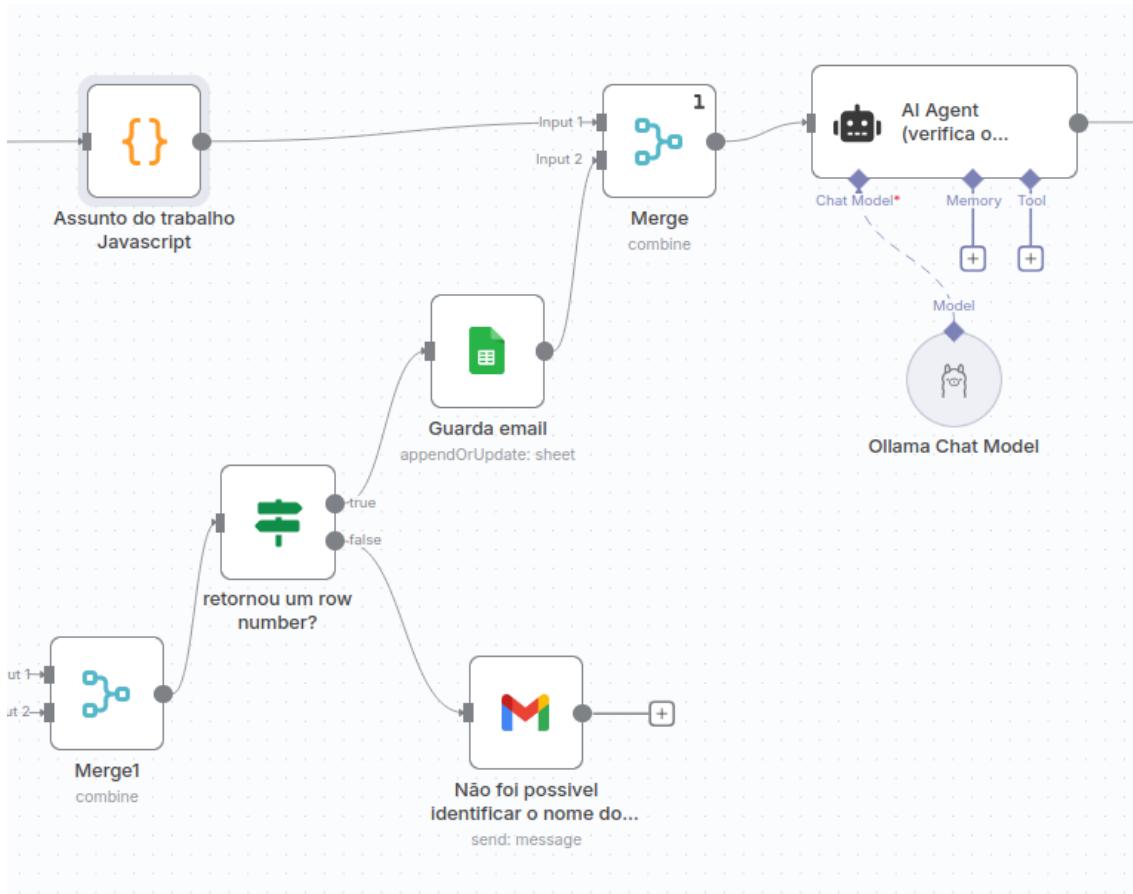


Figura 16. Fluxo de Execução (Workflow) do Agente de Conferência de Trabalhos Acadêmicos (Parte2).

convertido em texto plano pelo nó *Extract from File*. Em seguida, o nó *Code1*, que recebe um JSON, tanto com o conteúdo do email quanto com o texto do binário, isola a primeira linha do texto como sendo o metadado “Nome do Aluno” e concatena o restante como o “Corpo do Trabalho”. Sabe-se que a primeira linha contém o nome do aluno pois essa formato de email, em tese, seria padronizado anteriormente.

Paralelamente, o sistema carrega a lista de alunos da planilha por meio dos nós *Get row(s) in sheet* e *listadenomes*, formatando essa saída como um vetor para a busca. O nó *AI Agent2*, impulsionado pelo modelo **Qwen2 .5 : 3B**, recebe o nome extraído do PDF e a lista de classe. Sua função é a encontrar o aluno na lista e retornar estritamente o número da linha (*RowIndex*) correspondente na planilha, caso o nome se encontre na lista.

Após passar pelo primeiro agente, um nó condicional verifica se o agente anterior retornou um número inteiro válido (via Regex `^ [1-9] \d*`), ele fará isso caso o nome do aluno esteja na lista de alunos, ainda, em caso positivo, o email do aluno é armazenado na planilha do Google sheets, que serve como banco de dados. Se falhar, o fluxo encerra enviando um e-mail de “Aluno não identificado”. Se for bem-sucedido, o fluxo avança para a análise qualitativa.

Nesta etapa, o nó *AI Agent (verifica o conteúdo)* utiliza o modelo **Llama3 .2 : 3B**

para realizar uma validação cognitiva. Ele compara o texto do corpo do trabalho (extraído no passo 2) com um tópico de referência injetado pelo nó *Assunto do trabalho Javascript*. O modelo retorna um *booleano* (*true/false*) indicando se o conteúdo do documento corresponde ao tema exigido.

Antes de aplicar a lógica de decisão, o sistema necessita localizar o registro exato do aluno no banco de dados para atualização pois, a partir desse ponto, teremos o JSON retornado pelo último agente de IA que será necessário para integrar ao JSON gerado a partir do ponto referente ao aluno que enviou o email processado pelo workflow.

Em seguida, o nó *Get row(s) in sheet1* utiliza o índice da linha (RowNumber), identificado previamente pelo primeiro agente de IA, para recuperar o estado atual do aluno na planilha. Em seguida, o nó *Code2* atua como um normalizador de dados, injetando metadados auxiliares no objeto JSON para garantir que a estrutura de dados esteja sincronizada antes de entrar no nó condicional, garantindo que o nó condicional subsequente (*If*) receba uma entrada única e válida contendo os metadados do aluno, prevendo erros de iteração ou perda de referência durante o cruzamento de dados entre ramos paralelos

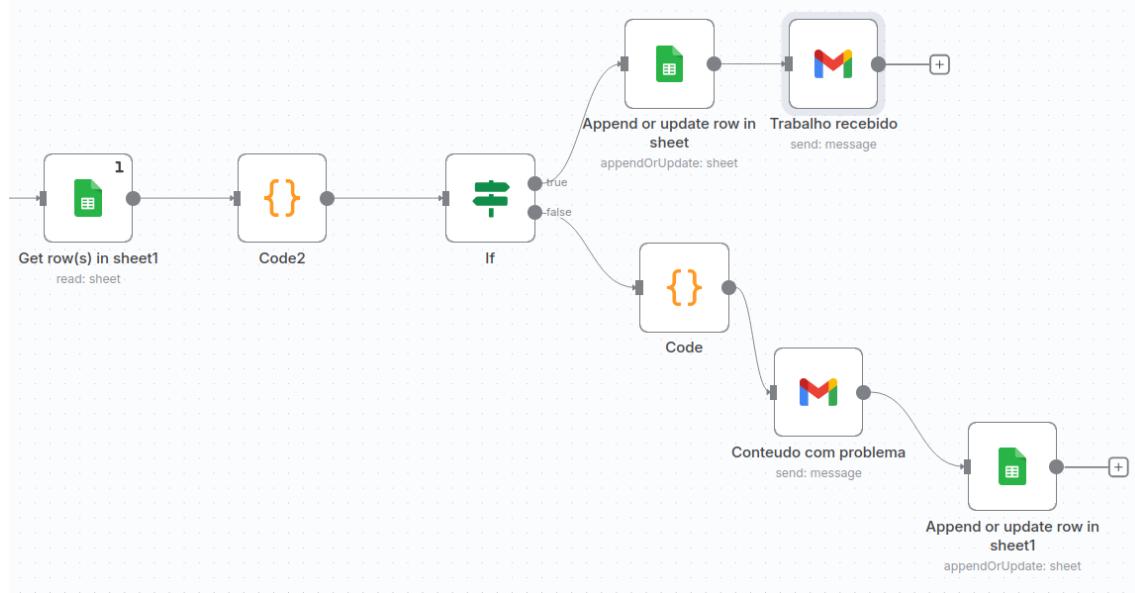


Figura 17. Fluxo de Execução (Workflow) do Agente de Conferência de Trabalhos Acadêmicos (Parte3).

O nó *If* atua como o portão lógico do sistema. Ele avalia a saída booleana gerada pelo agente de análise semântica anterior. Se o valor de saída for estritamente igual a “*true*”, o fluxo segue para o ramo de aprovação; caso contrário, desvia para o ramo de rejeição/alerta.

Caso o trabalho esteja em conformidade com o tema o nó *Append or update row in sheet* executa uma operação de *Update*. Ele marca a coluna “Entregue” como “OK” e, crucialmente, valida a coluna “Conteúdo” também como “OK”, sinalizando que a submissão foi aceita automaticamente.

Na sequência, o nó *Trabalho recebido* utiliza a API do Gmail para disparar um

email transacional ao aluno, confirmando o recebimento e a validação preliminar do documento.

Caso o conteúdo divirja do tema proposto o nó *Code* realiza uma sanitização do objeto de dados para garantir que as variáveis de email e nome estejam acessíveis para os nós subsequentes de erro.

Em seguida, o nó *Conteudo com problema* envia um e-mail de alerta ao aluno com o assunto “Aviso sobre o conteúdo”, informando que, embora o arquivo tenha sido recebido, o conteúdo aparenta estar incorreto. O nó *Append or update row in sheet1* atualiza a planilha registrando o recebimento (“Entregue”: “OK”), mas marca as colunas qualitativas com indicadores de atenção (ex: “Conteúdo”: “?” ou “Ver conteúdo”), alertando o docente para uma revisão manual posterior.

5. Resultados

Esta Seção faz um sumário dos resultados obtidos pela implementação dos quatro mecanismos de automação propostos no trabalho, detalhando situações adicionais que ocorreram durante sua implementação e testes.

Agendamento de Horário do Professor:

O agente obteve sucesso na maioria dos casos. Em testes práticos, o sistema conseguiu interpretar solicitações em linguagem natural (ex: *Quero marcar uma reunião com o professor na semana que vem*), consultar os horários disponíveis na Agenda do Google, apresentar as opções ao usuário e, após a confirmação, criar o evento com os dados corretos do acadêmico. O modelo **Qwen2 . 5 : 3B** demonstrou capacidade para seguir as instruções corretamente, desde que elas fossem **extremamente claras e diretas** como, por exemplo, *Nunca pule nenhum passo*.

Lista de Tarefas Seguindo Regras (Pomodoro):

O agente de roteamento funcionou bem na maioria dos casos. O modelo **Qwen2 . 5 : 3B** conseguiu diferenciar, com sucesso, a maioria das entradas de diálogo comuns (tratadas pelo sub-workflow *None*) de comandos de tarefa. Tarefas enviadas foram corretamente adicionadas ao Google Sheets e, subsequentemente, o sub-workflow de agendamento (*CompletaAgendaPomodoro*) preencheu a Agenda do Google do usuário com os blocos Pomodoro. Quanto mudança de prioridade, após vários testes, ficou claro que entradas do tipo (c/) para mudança e (r/) para remoção funcionaram de forma adequada por serem simples de serem compreendidas pelo modelo SLM utilizado.

Ler Documentos em Busca de Padrões de Interesse:

O modelo demonstrou eficácia no monitoramento de padrões de interesse. Nos testes, ao enviar um boletim em PDF com a string “Professor Fulano”(ou apenas o sobrenome “Sicrano”), o workflow sintetizou o documento de centenas de páginas em alguns poucos blocos relevantes que mencionavam o nome. O *Information Extractor* (*Qwen2 . 5 : 3B*)

conseguiu ler os blocos filtrados e retornar um resumo preciso do que se tratava o texto, por exemplo, *Portaria de designação para comissão* ou *Convocação para reunião* e a página exata da menção. A combinação de filtragem por código e extração por IA foi capaz de sintetizar o documento. Além disso, embora não implementado, poderia gerar gatilhos de aviso ao usuário quando o termo desejado aparecesse no documento.

Resposta Automatizada de Tarefas de Acadêmicos:

O sistema automatizou com sucesso o processo de conferência do envio de atividades acadêmicas. O Agente 1 (**Qwen2 .5 :3B**) demonstrou capacidade de identificar corretamente os alunos, mesmo com pequenas variações no nome, na grande maioria das vezes. Por outro lado, o Agente 2 (**Qwen2 .5 :3B**) conseguiu diferenciar, semanticamente, trabalhos que estavam dentro do escopo da atividade (*True*) de trabalhos fora do tema (*False*), por exemplo, um trabalho cujo escopo é “Implementação de uma Árvore AVL”, não deve receber um trabalho cuja resolução é a implementação do *BubbleSort*. O workflow também conseguiu, corretamente, atualizar a planilha de controle e envio de email de confirmação ou aviso ao aluno.

5.1. Modelos SLM Utilizados

A comparação entre os Pequenos Modelos de Linguagem utilizados como **motor** do agente de IA nos workflows de automação, tinha como objetivo a redução do *trade-off* entre a capacidade de compreensão, acurácia das respostas e eficiência computacional em hardware de baixo custo. Por exemplo, o **Llama 3 .2 :3B**, embora eficiente em conversação, demonstrou dificuldades significativas no uso das ferramentas (*tools*) pelo agentes, com suas versões 1B e 3B apresentando níveis elevados de alucinação e inviabilidade prática. Por outro lado, o **Phi 3:3 .8B**) não foi suportado pelo hardware, pois seu custo operacional (VRAM) e o tempo de inferência foram demasiadamente longos. Como o n8n tem um *timeout* de no máximo 10 minutos, este modelo não atendeu aos requisitos mínimos.

Por outro lado, a família de modelos **Qwen 2 .5** apresentou o melhor *trade-off*. Por exemplo, a versão 0 .5B teve baixo desempenho, sendo incapaz de usar ferramentas e erros frequentes no JSON de saída. Já a versão 1 .5B teve uma melhora, mas ainda falhou na avaliação de fluxos de trabalho complexos. Por fim, a versão 3B teve os melhores resultados, apresentando um bom balanço entre a eficiência computacional (mantendo-se dentro dos limites do hardware) e a capacidade de realizar as tarefas, compreendendo entradas (*inputs*), utilizando ferramentas de forma confiável (saída esperada) e gerando os resultados de forma estruturada e correta.

5.1.1. Resultados do uso de Linguagem Natural pelos Modelos

As Tabela 2 3 e 4 apresentam os resultados dos testes de processamento de linguagem natural. As três mensagens a seguir foram enviadas para a automação do Pomodoro:

- “Apagar a luz da cozinha antes de dormir” (Texto 1),
- “Preciso que você anote uma coisa para mim” (Texto 2)
- “Bom dia, não esqueça de comprar leite” (Texto 3).

Tabela 2. Comparação de saída dos modelos ao receberem a entrada do Texto 1.
A saída com estado OK, representa o resultado correto e esperado e Erro, caso contrário.

Modelo	Resposta ao Texto 1	Estado
Qwen 2.5:0.5B	>"tool": "changePriority", "argument": "c/ 1 6"	Erro
Qwen 2.5:1.5B	"tool": "addTask", "argument": "pagar a luz..."	Erro
Qwen 2.5:3B	"Apagar a luz..." {"r/ 2"}	Erro
Llama 3.2:1B	"type": "func", "func": {"name": "None"..."}	Erro
Llama 3.2:3B	["apagar a luz da cozinha..."]	OK

Tabela 3. Comparação de saída dos modelos ao receberem a entrada do Texto 2.

Modelo	Resposta ao Texto 2	Estado
Qwen 2.5:0.5B	"tool": "addTask", "arguments": "preciso que voce anote..."	Erro
Qwen 2.5:1.5B	"tool": "addTask", "argument": "preciso que vc anote isso"	Erro
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..."	Erro

Em cada coluna das tabelas são observadas a resposta do modelo ao texto correspondente. Ou seja, a linha para o modelo **Qwen 2.5:0.5B** para a coluna Resposta ao Texto 1, representa a resposta do modelo a entrada de Texto 1. Em seguida, a coluna Estado, apresenta o resultado avaliado por um ser humano, se a saída foi correta em relação ao contexto (OK), ou cometeu um erro (Erro). O Erro, neste caso, significa que o modelo alucinou ou não teve a saída esperada.

Tabela 4. Comparação de saída dos modelos ao receberem a entrada do Texto 3.

Modelo	Resposta ao Texto 3	Estado
Qwen 2.5:0.5B	"fazer a lista de compras"	OK
Qwen 2.5:1.5B	"tool": "addTask", "argument": "comprar leite"	Erro
Qwen 2.5:3B	"comprar leite"	OK
Llama 3.2:1B	'''	Erro
Llama 3.2:3B	"comprar leite"	OK

A seguir avaliamos a capacidade dos modelos **Llama 3.2 : 3B** e **Qwen 2.5 : 3B** ao processar linguagem natural (PLN) em cenários de chat, variando entre solicitações formais, informais e contextos emocionais. O objetivo foi a proximidade de um resposta humana e a coerência do modelo em relação aos textos de entrada fornecidos pelos usuários. Os outros modelos foram desconsiderados pois suas saídas foram consideradas robotizadas (Tabela 5).

Texto 1 (Input): “Bom dia, gostaria de marcar uma reunião”

Llama 3.2 : 3B> “Olá! Eu sou João, assistente do professor. Você gostaria de marcar um horário de reunião? Você tem alguns horários disponíveis: 2025-12-09T16:00... [Lista de datas]. Qual é o horário que prefere?”

Qwen 2.5 : 3B> “Olá! Eu sou João, assistente do professor. Você gostaria de marcar um horário de reunião?”

Texto 2 (Input): “Vê se consegue encaixar uma call rapidinho pra mim hoje.”

Llama 3.2 : 3B> “Olá! Eu sou João, assistente do professor. Você gostaria de marcar um horário de reunião?”

Qwen 2.5 : 3B> “Olá! Eu sou João, assistente do professor. Você gostaria de marcar um horário de reunião?”

Texto 3 (Input): “Cheguei em casa moído do trabalho e ainda tenho que lavar a louça. Ninguém merece.”

Llama 3.2 : 3B> “Certo, fico à disposição caso precise. Tenha um ótimo dia!”

Qwen 2.5 : 3B> “Peço desculpas por sua situação de hoje, mas parece que você está ocupado. Vamos agendar sua próxima reunião com o professor. Olá! Eu sou João, assistente do professor. Você gostaria de marcar um horário de reunião?”

A análise comparativa revela uma distinção significativa na qualidade das respostas dos dois modelos. O **Llama 3.2 : 3B** demonstrou ser consistente no processamento de linguagem natural, exibindo proatividade ao buscar dados de agenda quando solicitado (Texto 1) e inteligência contextual ao encerrar polidamente a conversa diante de um desabafo (Texto 3).

Por outro lado, o **Qwen 2.5** apresentou um comportamento mecânico, respostas redundantes, além de dificuldades severas em manter a coerência do fluxo de diálogo. Portanto, para interfaces de chat que exigem naturalidade e robustez lógica, o modelo Llama 3.2 se mostrou ser a solução mais adequada.

5.2. Avaliação dos Modelos

A Tabela 5 apresenta uma comparação empírica de testes realizados entre os modelos: **Llama 3.2 : 1B**, **Llama 3.2 : 3B**, **Qwen 2.5 : 0.5B**, **Qwen 2.5 : 1.5B** e **Qwen 2.5 : 3B**. O modelo **Phi 3 : 3.8B**, foi desconsiderado, mesmo ele, aparentemente, usando o máximo limite da VRAM do hardware, suas respostas excediam o tempo máximo limite de espera do n8n.

Tabela 5. Dados obtidos nos testes práticos de automação e agendamento em relação aos modelos de SLM disponíveis no repositório do OLLama e que não ultrapassavam o limite de uso da memória VRAM da placa de vídeo.

Critério	Llama 3.2:1B	Llama 3.2:3B	Qwen 2.5:0.5B	Qwen 2.5:1.5B	Qwen 2.5:3B
Objetivo	Não	Sim	Não	Sim	Sim
Consistência	-	Baixa	-	Média	Alta
Qty. Aluc.	Alta	Média	Alta	Média	Baixa
Grau Aluc.	Crítico	Crítico	Crítico	Crítico	Moderado
Tools	Falha	Não confiável	Falha	Não confiável	Confiável
Chat	Bom	Fluido/Natural	Ruim	Direto/Robótico	Direto/Robótico
Uso	Conversação	Conversação	Não recomendado	Não recomendado	Uso na extração de texto
Prompt	Baseado em Persona	Baseado em Persona	Rígido	Rígido	Diretivo, claro e curto

É possível observar na Tabela 5 que, em termos de **objetivo**, os modelos foram classificados como: *sim*, quando ele completou o fluxo de trabalho com sucesso e *não*, caso contrário. Com relação a **consistência**, os modelos foram classificados como: *alta*, se acertou corretamente as execuções repetidamente, *média*, se apresentava erros ocasionais gerando instabilidade nas respostas, *baixa*, se o sucesso era caso isolado ou –, quando o modelo sempre falhava.

Por outro lado, com relação a **Quantidade de Alucinações (Qty. Aluc.)**, ou seja, a quantidade de vezes que o modelo gerava respostas falsas ou erros de lógica, os modelos foram classificados como: *alta* se o modelo tinha alucinações constantes, *média* se, ocasionalmente, era necessária intervenção e verificação humana em relação as respostas e *baixa* se os erros eram raros e o índice de precisão satisfatório. Na mesma linha, o **Grau de Alucinação (Grau Aluc.)** classifica a severidade do erro cometido. Essa classificação pode ser: **crítica**, quando o erro quebra a estrutura (ex: JSON inválido) ou impede a continuidade do fluxo ou **moderado**, quando o formato está correto, mas o conteúdo da resposta é impreciso ou parcialmente incorreto, exigindo filtragens adicionais.

Por fim, do ponto de vista operacional em relação a integração com o n8n, os modelos foram categorizados pela sua capacidade de uso como **Tool**, **Chat**, característica de **Uso** do modelo e tipo de **Prompt** utilizado para instruir o modelo nas interações. Com relação ao uso como **Tool**, os modelos foram classificados como *confiável*, ou seja aciona a ferramenta correta com os parâmetros certos de forma consistente, *não confiável*, pois tenta utilizar usar as ferramentas, mas erra os parâmetros ou o momento de uso, de forma frequente e, por fim, *falha*, por não conseguir usar as ferramentas corretas no momento correto ou ignora o seu uso. Com relação ao **Chat** apenas determina a capacidade de conversação em linguagem natural, o **Uso**, qual a melhor utilização do modelo e, finalmente, o **Prompt**, determina como devem ser parametrizadas as informações para que o modelo funcione corretamente.

5.2.1. Problemas encontrados

O desenvolvimento da solução não seguiu um caminho linear, o percurso até a definição de configurações e organização dos fluxos de trabalho ideais foi marcado por iterações e ajustes forçados por limitações práticas. A busca pelo equilíbrio (trade-off) entre eficiência operacional e precisão nas respostas trouxe à tona problemas críticos, especialmente, com relação à tendência de alucinação em modelos compactos e à inconsistência ao tratar

instruções (cadeias de comandos) complexas. Na próxima Seção serão discutidas as dificuldades encontradas e como cada uma impactou as decisões de design dos sistemas.

5.2.2. Limitações da Janela de Contexto e Estratégias de Mitigação

Uma das restrições técnicas mais severas impostas pelo *hardware* de baixo custo (NVIDIA Quadro P1000) foi a limitação da janela de contexto a 2048 *tokens*. Durante os testes, constatou-se uma correlação direta entre a densidade de informação fornecida na entrada (*input*) e a taxa de erro. Ou seja, quanto maior a quantidade de texto inserido na janela de contexto, maior a probabilidade de todos os modelos testados apresentarem alucinações ou falharem na lógica de raciocínio.

Para contornar essa limitação física e cognitiva dos modelos, a arquitetura dos fluxos de trabalho (*workflows*) foi organizada para priorizar a atomicidade das tarefas. Buscou-se exigir dos agentes ações simples e, extremamente, objetivas, fornecendo apenas o fragmento de informação estritamente necessário para aquela etapa específica, minimizando o ruído.

Um exemplo prático dessa estratégia foi a implementação do **Ler Documentos em Busca de Padrões de Interesse**. Devido à impossibilidade de processar documentos extensos de uma única vez dentro do limite de 2048 *tokens*, foram utilizadas técnicas de programação estruturada para realizar uma pré-segmentação do arquivo. O documento foi segmentado em blocos menores baseados em palavras-chave antes de ser submetido à inferência do modelo. Dessa forma, a IA atuou apenas na análise semântica de trechos curtos, reduzindo a carga de contexto e, consequentemente, a incidência de alucinações.

Além disso, a Engenharia de Prompt também precisou ser adaptada. As instruções (*System Messages*) foram refinadas para serem extremamente objetivas, curtas e livres de ambiguidade, garantindo que o pouco espaço disponível na janela de contexto fosse utilizado para processamento de dados úteis e não desperdiçado com instruções que poderiam confundir os modelos menores.

5.3. Descartando o uso de RAGs e Limitações de Embedding

Uma técnica muito utilizada com LLMs são os fluxos de trabalho baseados em *Retrieval-Augmented Generation* (RAG). Ou seja, modelos que usam bancos de dados vetoriais para armazenar informações e responder, rapidamente, a perguntas com contexto.

Infelizmente, devido às restrições de *hardware*, foi necessário limitar o uso a apenas mecanismos de *embedding* (ferramentas de separação de palavras e textos curtos), como o **snowflake-arctic-embed:22m**. Testes práticos, contudo, revelaram que este modelo, devido ao seu tamanho reduzido, não foi capaz de capturar nuances semânticas complexas da língua portuguesa. Essa deficiência gerou um impasse insolúvel relacionado à janela de contexto limitada a 2048 *tokens* por dois motivos. Primeiro, em **cenário de alta recuperação**, tentou-se compensar a baixa precisão do *embedding* buscando um número maior de fragmentos (*chunks*) do banco de dados. O problema ocorreu por que a quantidade de texto excedia, rapidamente, o limite da janela de contexto, impedindo o processamento pelo modelo. Segundo, em **cenário de baixa recuperação**, pois ao reduzir a quantidade de *chunks* (pedaços) de texto enviados (para respeitar o limite de memória),

a baixa qualidade vetorial fazia com que os poucos fragmentos retornados fossem irrelevantes ou desconexos em relação ao contexto fornecido pelo usuário. Isso introduzia ruído no *prompt*, levando o modelo a gerar alucinações baseadas em premissas falsas.

Diante desse cenário, a implementação do RAG foi considerada inviável para a configuração de *hardware* proposta, optando-se pelo uso de programação estruturada para segmentação de contexto estático.

5.3.1. Limite de tempo de execução do n8n

Em uma tentativa subsequente de viabilizar a arquitetura RAG, buscou-se superar a baixa precisão semântica substituindo a primeira versão do modelo `snowflake` de entrada pela versão `snowflake-arctic-embed2:568m`. Reconhecendo que este modelo excedia a capacidade de memória de vídeo (VRAM) do *hardware* disponível, a estratégia adotada consistiu em transferir a carga de trabalho para a CPU. O objetivo era realizar o povoamento (*ingestion*) do banco de dados vetorial aceitando um tempo de processamento elevado, sob a premissa de que a lentidão inicial seria compensada pela disponibilidade posterior de vetores de alta qualidade para os agentes.

No entanto, essa abordagem encontrou restrições operacionais da plataforma n8n, especificamente, a sua janela de tempo de execução (*execution timeout*). O processamento intensivo (via CPU) necessário para gerar *embeddings* com o modelo de 568M de parâmetros, elevou a latência do fluxo, fazendo com que as execuções ultrapassassem, sistematicamente, o limite padrão de 10 minutos.

Foram realizadas extensas tentativas de mitigação técnica para contornar este bloqueio. Por exemplo, foi realizada uma reconfiguração de variáveis de ambiente no *Docker* visando estender os parâmetros de *timeout* do contêiner. Apesar dos esforços, não foi possível superar a interrupção forçada dos processos pelo orquestrador. A impossibilidade técnica de processar modelos de *embedding* robustos dentro da janela de tempo disponível, somada à incapacidade de alocação em VRAM, tornou a implementação de um RAG inviável para os problemas definidos neste trabalho.

5.3.2. Perda de Capacidade de *Tool Use* em Modelos Quantizados

Visando maximizar a eficiência do *hardware*, explorou-se a utilização de modelos quantizados disponíveis nativamente na biblioteca do OLLama. O objetivo era utilizar versões compactas (ex: `q4_k_m`) para reduzir o uso de memória. No entanto, durante a fase de validação, identificou-se uma limitação funcional nessas versões específicas, ou seja, a ausência ou degradação da capacidade de *Tool Use* (ou *Function Calling*) quando os modelos estavam quantizados.

Os testes demonstraram que, embora esses modelos sejam capazes de interagir com o usuário por meio de um *chat* de texto, mimetizando linguagem natural, eles eram incapazes de retornar um JSON estruturado da forma correta para ser enviado às ferramentas externas. O modelo falhava em reconhecer a instrução, *input*, necessária para usar uma ferramenta ou, em outros casos, não gerava o JSON com a sintaxe correta para a integração.

E, uma vez que a arquitetura de agentes autônomos proposta neste trabalho depende, estritamente, da capacidade do modelo de agir sobre o ambiente por meio de ferramentas, os modelos quantizados que apresentaram essa deficiência foram considerados inadequados para o escopo da automação, restringindo a seleção aos modelos que, comprovadamente, suportavam a orquestração de *tools*.

6. Conclusão

Este trabalho implementou e validou vários fluxos de execução de atividades utilizando Pequenos Modelos de Linguagem (SLMs), em contraste ao uso de Grandes Modelos de Linguagem (LLMs), no contexto de execução de operações em ambientes com uso de hardware com poder computacional restrito. Ou seja, o objetivo era criar soluções de baixo custo e alta eficiência para automação de problemas utilizando plataformas de *low-code* (como o n8n) e placas de vídeo simples, com não mais do que 4Gb de VRAM (como a NVIDIA Quadro P1000 utilizada nos experimentos).

Os resultados demonstraram que os agentes de IA desenvolvidos foram capazes de resolver problemas práticos como, por exemplo, o agendamento de reuniões com acadêmicos ou o monitoramento de documentos buscando por strings específicas, todas essas soluções recebendo instruções simples em linguagem natural. Foi possível verificar que os modelos foram capazes de utilizar as *tools* (ferramentas) disponibilizadas nos workflows do n8n de forma eficaz, acionando sub-workflows para, por exemplo, consultar a Agenda do Google ou interagir com planilhas eletrônicas, quando necessário.

Contudo, os prompts utilizados com os modelos foram mais importantes do que era esperado. Foi necessário quebrar as etapas de resolução dos problemas em pedaços bem pequenos. Isso permitiu que o agente recebesse instruções diretas e objetivas, garantindo que a tarefa fosse executada corretamente. Ou seja, o limite de mensagens que uma SLM consegue processar são baixos, então as instruções devem ser curtas, principalmente pelo limite de 2048 tokens de texto.

Além disso, foi necessária programação adicional para permitir que os modelos conseguissem tratar os problemas corretamente. Ou seja, os agentes trabalharam em conjunto com códigos de programação para filtrar e formatar dados e saídas, para garantir a lógica do processo. Isso acabou criando um sistema híbrido eficiente.

Por outro lado, na avaliação comparativa, o modelo Qwen 2.5:3B foi o que se destacou em todos os cenários práticos. Ele apresentou o melhor balanço entre eficiência computacional e a capacidade de seguir instruções complexas e utilizar ferramentas de forma confiável. Ou seja, os modelos SLMs são capazes de resolver problemas caso os comandos enviados a eles sejam simples e as instruções diretas e objetivas.

A vantagem dessa arquitetura é que os modelos se destacam frente à programação comum, pois podem tomar decisões em cenários onde cada condição possível precisaria ser pensada e codificada (por exemplo, estruturas condicionais complexas, como cadeias de IFs para interpretar a variabilidade da intenção do usuário).

Como conclusão deste trabalho, é possível afirmar que usar os modelos SLMs, quando orquestrados corretamente em plataformas *low-code* como o n8n, pode resolver problemas específicos com um baixíssimo custo computacional e de desenvolvimento, democratizando o acesso a soluções de automação inteligente.

6.1. Trabalhos Futuros

Embora este trabalho tenha demonstrado a viabilidade de agentes baseados em SLMs para tarefas específicas, as limitações técnicas identificadas durante o desenvolvimento abrem caminhos para diversas linhas de pesquisa futura, visando aumentar a robustez e a complexidade das automações em hardware de baixo custo. Embora na implementação atual o uso de RAGs tenha falhado, revisitá-lo usando novos modelos de embeddings pode tornar o seu uso prático.

Além disso, com a eficiência comprovada do modelo Qwen 2.5:3B, seria interessante investigar sua quantização para uso em sistemas portáteis como Orange Pi, Raspberry Pi 5 ou Jetson Nano, podendo levar a integrações de agentes com SLM para cenários totalmente desconectados da rede, como escolas ou postos de saúde em locais remotos ou sem acesso a hardware capaz de processamento usando LLMs.

Referências

- Australian Institute for Machine Learning (AIML) (2024). Beyond bigger models: The evolution of language model scaling laws. Acesso em: 26 set. 2025.
- 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.
- Cirillo, F. (2018). *The Pomodoro Technique: The Acclaimed Time-Management System That Has Transformed How We Work*. Currency, New York.
- Docker (2023). Docker compose documentation. Acessado em: 31 de março de 2025.
- 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.
- Kim, Y., Yi, E., Kim, M., Yun, S.-Y., and Kim, T. (2025). Guiding reasoning in small language models with Ilm assistance. Acesso em: 25 set. 2025.
- Labrak, Y., Bazoge, A., Morin, E., Gourraud, P.-A., Rouvier, M., and Dufour, R. (2024). Biomistral: A collection of open-source pretrained large language models for medical domains.
- 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. Acesso em: 26 set. 2025.
- Licardo, J. T. and Tankovic, N. (2025). Performance trade-offs of optimizing small language models for e-commerce. Acesso em: 26 set. 2025.
- n8n-io (2025). n8n: Secure Workflow Automation for Technical Teams. Repositório GitHub. Acessado em: 2025-11-05.
- Nguyen, C. V., Shen, X., Aponte, R., Xia, Y., Basu, S., Hu, Z., Chen, J., Parmar, M., Kunapuli, S., Barrow, J., Wu, J., Singh, A., Wang, Y., Gu, J., Dernoncourt, F., Ahmed, N. K., Lipka, N., Zhang, R., Chen, X., Yu, T., Kim, S., Deilamsalehy, H., Park, N., Rimer, M., Zhang, Z., Yang, H., Rossi, R. A., and Nguyen, T. H. (2024). A Survey of Small Language Models. Acesso em: 26 set. 2025.
- Qwen (2025). Get Up and Running with Large Language Models. <https://oLlama.com/>. [Online; accessed 17-03-2025].

Openai (2025). Chatgpt: modelo gpt-4. <https://chat.openai.com/>. Acesso em: 19 nov. 2025.

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. Acesso em: 26 set. 2025.

Retamales, J., Retamales, J. P., Demarchi, A. M., Gonzalez, M., Lopez, C., Ramirez, N., Retamal, T., and Sun, V. (2025). Leveraging artificial intelligence to uncover symptom burden in palliative care: Analysis of nonscheduled visits using a phi-3 small language model. *JCO Global Oncology*, (11):e2400432. PMID:.

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. Acesso em: 25 set. 2025.

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. Acesso em: 25 set. 2025.

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