Architecture over Scale: Multi-Agent Reasoning for ENEM Mathematics

Paulo Victor C. Ferreira[1], **Erica C. Silveira**[2], **Guilherme L. dos Santos**[2], **Francisco Leudes B. Neto**[2], **Marcos A. de Oliveira**[3]

[1] Engenharia de Software – Universidade Federal do Ceará (UFC), Quixadá

[2]Ciência da Computação – Universidade Federal do Ceará (UFC), Quixadá

[3]Programa de Pós-Graduação em Computação (PCOMP) – UFC, Quixadá
Av. José de Freitas Queiroz, 5003 – Cedro Novo –
CEP 63902-580 – Quixadá – CE – Brazil

{paulovictorcosta,ery.cxtr,guilhermelopess,leudesz}@alu.ufc.br
marcos.oliveira@ufc.br

Abstract. *This paper evaluates a lightweight multi-agent architecture for mathematical reasoning in small language models. On 301 self-contained textual ENEM mathematics questions, we compare direct chain-of-thought prompting, two 20B multi-agent configurations separating orchestration from code-based calculation, and direct prompting on a 120B model from the same family. The verified 20B multi-agent system reaches 97.7% accuracy, matching direct 120B prompting and outperforming direct 20B prompting at 91.7%. This gain increases token use and cost, but enables large-model accuracy with a smaller model suitable for local deployment.*

Resumo. *Este artigo avalia uma arquitetura multiagente leve para raciocínio matemático em modelos de linguagem pequenos. Em 301 questões textuais e autocontidas de matemática do ENEM, comparamos o raciocínio direto, duas configurações multiagente de 20B e a inferência direta em um modelo de 120B da mesma família. O sistema multiagente verificado de 20B alcança 97,7% de acurácia, igualando o modelo de 120B e superando os 91,7% do modelo de 20B direto. Embora aumente o uso de tokens e o custo, a arquitetura permite obter desempenho de modelos maiores com um modelo adequado à execução local.*

1. Introdução

Modelos de linguagem de grande porte (*Large Language Models* – LLMs) já apresentam desempenho elevado em tarefas educacionais e exames padronizados, especialmente quando combinados com estratégias de *chain-of-thought* (CoT) [Wei et al. 2022]. No contexto brasileiro, Nunes et al. [Nunes et al. 2023] avaliaram GPT-3.5 e GPT-4 em questões do Exame Nacional do Ensino Médio (ENEM) e mostraram que o GPT-4 com CoT alcançou desempenho expressivo, inclusive na edição de 2022, usada como avaliação posterior ao período de treinamento dos modelos. O benchmark *Brazilian Leading Universities Entrance eXams* (BLUEX) amplia esse cenário com questões de vestibulares da UNICAMP e da USP, reforçando a importância de recursos de avaliação em português [Almeida et al. 2023]. Nesse cenário, a questão de pesquisa deixa de ser apenas se LLMs

conseguem resolver provas como o ENEM, e passa a ser em que condições esse desempenho pode ser obtido sem depender de modelos cada vez maiores. Essa distinção é relevante porque modelos de maior escala tendem a exigir mais recursos computacionais, maior dependência de provedores externos e menor viabilidade para execução local, fatores importantes em aplicações educacionais, cenários de privacidade e ambientes com restrições de infraestrutura.

Este trabalho investiga se parte do ganho normalmente associado à escala pode ser recuperada por meio de organização arquitetural. Em vez de delegar toda a resolução a uma única chamada ao modelo, avaliamos arquiteturas multiagentes leves que separam o papel de raciocínio, atribuído a um orquestrador, do papel de execução e verificação de cálculos, atribuído a um agente executor de código. Essa escolha se apoia na hipótese de que, em questões matemáticas textuais, muitos erros não decorrem necessariamente da incapacidade de compreender o problema, mas de falhas locais em cálculos intermediários ou na verificação da resposta. Essa motivação se aproxima dos *Program-Aided Language Models* (PAL), nos quais o modelo de linguagem interpreta e decompõe o problema, enquanto a execução computacional é delegada a um interpretador externo [Gao et al. 2022].

Para avaliar essa hipótese, construímos um conjunto de dados com 301 questões autocontidas de Matemática do ENEM, cobrindo as edições de 2009 a 2025, em formato puramente textual e unimodal. Comparamos uma linha de base de chamada direta a um modelo de 20 bilhões de parâmetros, duas configurações multiagentes usando o mesmo modelo de 20B e uma linha de base direta com um modelo de 120 bilhões de parâmetros da mesma família. Os resultados mostram que a arquitetura multiagente com verificação obrigatória por código alcança 97,7% de acurácia, igualando o modelo de 120B em chamada direta e superando a linha de base direta de 20B, que obteve 91,7%. Dessa forma, a contribuição principal do trabalho não está em propor um novo estado da arte para resolução do ENEM, mas em mostrar empiricamente que decomposição arquitetural e verificação computacional podem reduzir a dependência de escala em tarefas educacionais de raciocínio matemático em português.

2. Trabalhos Relacionados

A literatura que sustenta o presente trabalho organiza-se em três eixos complementares. O primeiro trata da concepção de sistemas multiagentes baseados em modelos de linguagem, no qual se discute como a colaboração entre componentes especializados pode compensar limitações de modelos isolados. O segundo aborda o raciocínio matemático em modelos de linguagem, evidenciando tanto os ganhos obtidos com técnicas de *prompting* explícito quanto os modos de falha persistentes nesse domínio. O terceiro trata de mecanismos de verificação e autocorreção de respostas, particularmente aqueles que utilizam execução de código como forma de validação independente. A seguir, discutimos cada um desses eixos e situamos, ao final, como o presente estudo se posiciona na interseção entre eles.

A evolução recente das arquiteturas baseadas em LLMs tem se afastado do modelo monolítico, em direção a sistemas em que múltiplos agentes coordenam tarefas sob um orquestrador central. Essa mudança é motivada pela observação de que decompor problemas complexos em subtarefas atribuídas a agentes especializados permite melhor uso de ferramentas externas e reduz a carga cognitiva sobre qualquer compo-

nente isolado. O método ReAct intercala raciocínio e ações sobre ambientes externos [Yao et al. 2023], enquanto o Toolformer treina o próprio modelo para decidir quando e como invocar ferramentas por meio de APIs [Schick et al. 2023]. Trabalhos como o de [Belcak et al. 2025] argumentam que modelos pequenos, organizados em pipelines agente-centrados, são uma alternativa viável aos modelos monolíticos de grande porte, por motivos de latência, custo e especialização funcional. Nessa direção, [Żywot et al. 2026] examinam empiricamente se a colaboração entre modelos pequenos pode rivalizar com um único modelo grande, propondo uma arquitetura multiagente minimalista com planejamento centralizado em um orquestrador e comunicação restrita entre componentes mediada por memória estruturada. Os autores variam sistematicamente a posição do raciocínio explícito ao longo da arquitetura e mostram que o pensamento contribui de forma assimétrica: melhora a decomposição quando aplicado ao orquestrador, mas tende a desestabilizar a orquestração quando aplicado de forma irrestrita aos sub-agentes. O presente trabalho se relaciona também ao *Graph of Thoughts*, que representa passos de raciocínio e suas dependências como um grafo, permitindo combinar e revisar resultados intermediários [Besta et al. 2024]. Nossa proposta se inscreve nessa linha ao adaptar o desenho de orquestração centralizada para o domínio da matemática do ENEM em português e introduzindo a escala do modelo como variável independente de comparação.

A capacidade de LLMs em raciocínio matemático avançou substancialmente nos últimos anos, em larga medida pela introdução de técnicas de *prompting* que tornam explícitos os passos intermediários de solução. O *chain-of-thought prompting* [Wei et al. 2022] mostrou que solicitar explicitamente uma cadeia de raciocínio antes da resposta final melhora de forma consistente o desempenho em problemas matemáticos, em particular nos modelos de maior porte. Benchmarks como GSM8K, composto por problemas matemáticos escolares em linguagem natural [Cobbe et al. 2021], e MATH, formado por problemas de competições de matemática [Hendrycks et al. 2021], tornaram explícita a dificuldade de manter cadeias corretas de raciocínio mesmo com modelos grandes. Apesar desses ganhos, a literatura documenta limitações persistentes: erros aritméticos pontuais no traço de raciocínio, inconsistências lógicas entre passos sucessivos e falhas de generalização para problemas estruturalmente semelhantes mas com numerais diferentes. Esses modos de falha aparecem mesmo quando o modelo demonstra compreensão correta do enunciado, indicando que o gargalo do raciocínio matemático em LLMs não se restringe à capacidade de planejar a solução, mas inclui a execução confiável dos cálculos intermediários. Em outras palavras, o desafio deslocou-se de habilitar o raciocínio para garantir a confiabilidade do resultado produzido. O *Program of Thoughts* separa essas funções ao delegar a computação a um interpretador de programas [Chen et al. 2023]; de forma complementar, o ToRA intercala raciocínio em linguagem natural com o uso de bibliotecas computacionais e solucionadores simbólicos [Gou et al. 2024]. Essa observação fundamenta a decisão metodológica central deste trabalho: separar, no nível da arquitetura, o componente que *raciocina* do componente que *calcula*, de modo que cada passo suscetível a erro aritmético possa ser independentemente verificado em vez de aceito por inferência.

Em resposta às limitações de confiabilidade do raciocínio puramente textual, uma linha de pesquisa vem propondo mecanismos de verificação externa e autocorreção como complemento à geração. Abordagens de *self-verification* e de *generate-and-verify* solicitam ao próprio modelo que confira a resposta produzida, frequentemente comparando

soluções obtidas por caminhos distintos; estratégias de reflexão iterativa estendem essa ideia, permitindo revisão sucessiva diante de inconsistências detectadas. O Reflexion, por exemplo, registra feedback linguístico em uma memória episódica para orientar tentativas posteriores [Shinn et al. 2023]. Entretanto, sem feedback externo confiável, a autocorreção puramente introspectiva pode não melhorar o raciocínio e até degradar o desempenho [Huang et al. 2024]. Uma vertente particularmente relevante para raciocínio matemático utiliza programas como verificadores: [Zhou et al. 2023] demonstram que, ao habilitar o GPT-4 *Code Interpreter* a gerar e executar código para conferir suas próprias soluções, a acurácia em problemas matemáticos desafiadores cresce de forma expressiva; [Toh et al. 2024] generalizam essa intuição com o método *Prove*, que traduz cada cadeia de raciocínio em um programa verificável e usa o resultado da execução como heurística para filtrar soluções incorretas antes da agregação por votação. O ponto em comum dessas abordagens é o reconhecimento de que a verificação externa, por ser independente do mesmo mecanismo que produziu o erro, oferece uma garantia qualitativamente distinta da autoavaliação puramente textual. Neste trabalho, a configuração com verificação obrigatória aplica esse princípio em escala arquitetural: em vez de instalar a verificação como passo opcional dentro de um agente único, dissocia-a em um sub-agente próprio, cuja função no protocolo é executar código de verificação e devolver o resultado ao orquestrador para reconciliação.

Em síntese, o presente trabalho se posiciona em relação à literatura discutida em três aspectos. (I) Adota a arquitetura multiagente de orquestração centralizada proposta por [Żywot et al. 2026], mas a aplica a um domínio e idioma ainda pouco cobertos por essa linha, matemática do ENEM em português, e adiciona a comparação entre modelos de 20B e 120B parâmetros, ausente no estudo original, permitindo distinguir o ganho da arquitetura do ganho da escala do modelo. (II) Reconhece os ganhos do *chain-of-thought prompting* [Wei et al. 2022] para raciocínio matemático, mas trata os erros aritméticos remanescentes como problema de arquitetura, não de *prompting*: em vez de buscar melhorar o traço de raciocínio, separa quem raciocina de quem calcula. (III) Incorpora o princípio de verificação por execução de código, articulado em [Zhou et al. 2023] e [Toh et al. 2024], mas o transpõe da escala intra-agente, em que verificação é decisão do próprio modelo, para a escala arquitetural, em que a verificação é um sub-agente independente do orquestrador, com contexto isolado e papel fixo no protocolo. Essas três escolhas, articuladas, definem o objeto deste estudo: investigar, sob condições experimentais controladas, em que medida a coordenação entre componentes pequenos e a verificação por código separada do raciocínio podem compensar a ausência de escala em um modelo subjacente.

3. Criação do conjunto de dados

O conjunto de dados foi construído a partir de questões de Matemática do ENEM extraídas das provas oficiais publicadas anualmente pelo Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP)[1]. Repositórios públicos de questões resolvidas, como o Resolve Enem e o Curso Objetivo foram utilizados de forma auxiliar para apoiar a transcrição do conteúdo textual, e os enunciados e gabaritos foram conferidos contra as provas oficiais. A coleta abrangeu todas as edições do exame disponíveis

[1] https://www.gov.br/inep/pt-br/areas-de-atuacao/avaliacao-e-exames-educacionais/enem/provas-e-gabaritos

no período de 2009 a 2025.

O processo de coleta privilegiou questões autocontidas, ou seja, questões cuja resolução depende exclusivamente das informações presentes no enunciado e nas alternativas. Não foram utilizadas etiquetas oficiais de habilidade, competência ou conteúdo, o que torna o conjunto uma coleção heterogênea de itens matemáticos com diferentes níveis de dificuldade e demandas de resolução, em vez de um corpus organizado por categorias pedagógicas.

Como o estudo restringe-se à avaliação puramente textual, aplicou-se um procedimento de filtragem em duas etapas. Na primeira, questões que dependiam de elementos pictóricos, como gráficos, mapas, fotografias ou ilustrações, foram descartadas integralmente, uma vez que sua resolução demanda capacidade multimodal que extrapola o escopo deste trabalho. Na segunda, questões essencialmente textuais que continham elementos de formatação não-ASCII, como tabelas, matrizes, frações complexas ou equações em notação matemática estendida, foram preservadas mediante conversão desses elementos para código LaTeX, processo conduzido com auxílio do modelo *Gemini 2.5 Pro*. Essa conversão tem propósito metodológico específico: preservar a integridade estrutural da informação matemática numa representação puramente textual, de modo que modelos unimodais possam processá-la sem perda de conteúdo. Dessa forma, evita-se o descarte de questões cuja única obstrução seria a renderização visual de expressões, e não a presença de informação intrinsecamente visual.

Cada questão selecionada foi registrada com três campos: o enunciado, o bloco de alternativas (A–E) e o gabarito oficial. Em seguida, todas as questões passaram por uma etapa de validação cruzada entre dois colaboradores, com papéis distintos. O colaborador responsável pela coleta inseria a questão no conjunto preliminar, e um segundo colaborador, independente da etapa de coleta, verificava a integridade do enunciado, a segmentação das alternativas e a correspondência entre o gabarito registrado e a alternativa correta. Questões aprovadas nessa verificação recebiam o rótulo `ok` e eram incorporadas à versão final. Questões com problemas de transcrição, formatação ou perda de informação recebiam o rótulo `em_revisao` e retornavam ao colaborador original para correção; após o reenvio, o colaborador independente realizava nova verificação, podendo aprovar a questão ou mantê-la em revisão até que estivesse adequada. O procedimento preserva, por construção, a separação de papéis entre coletor e verificador: correções de conteúdo são sempre feitas por quem inseriu a questão, e a validação fica restrita ao colaborador independente. Apenas questões aprovadas pelos verificadores foram incluídas no conjunto final, totalizando 301 itens[2]. Esse fluxo de coleta, revisão e incorporação ao conjunto final é sintetizado na Figura 1.

4. Metodologia

Investigamos se a colaboração entre modelos de linguagem pequenos, mediada por uma arquitetura multiagente mínima, pode compensar a ausência de escala em questões de matemática do ENEM. Adotamos o desenho de orquestração proposto por [Żywot et al. 2026], no qual um único orquestrador coordena um conjunto reduzido de sub-agentes especializados sob comunicação restrita, e o adaptamos a duas particularidades do nosso problema. Primeiro, as questões são autocontidas e textuais, o que torna

[2] `https://github.com/VictorFCos/enem-matematica-unimodal`

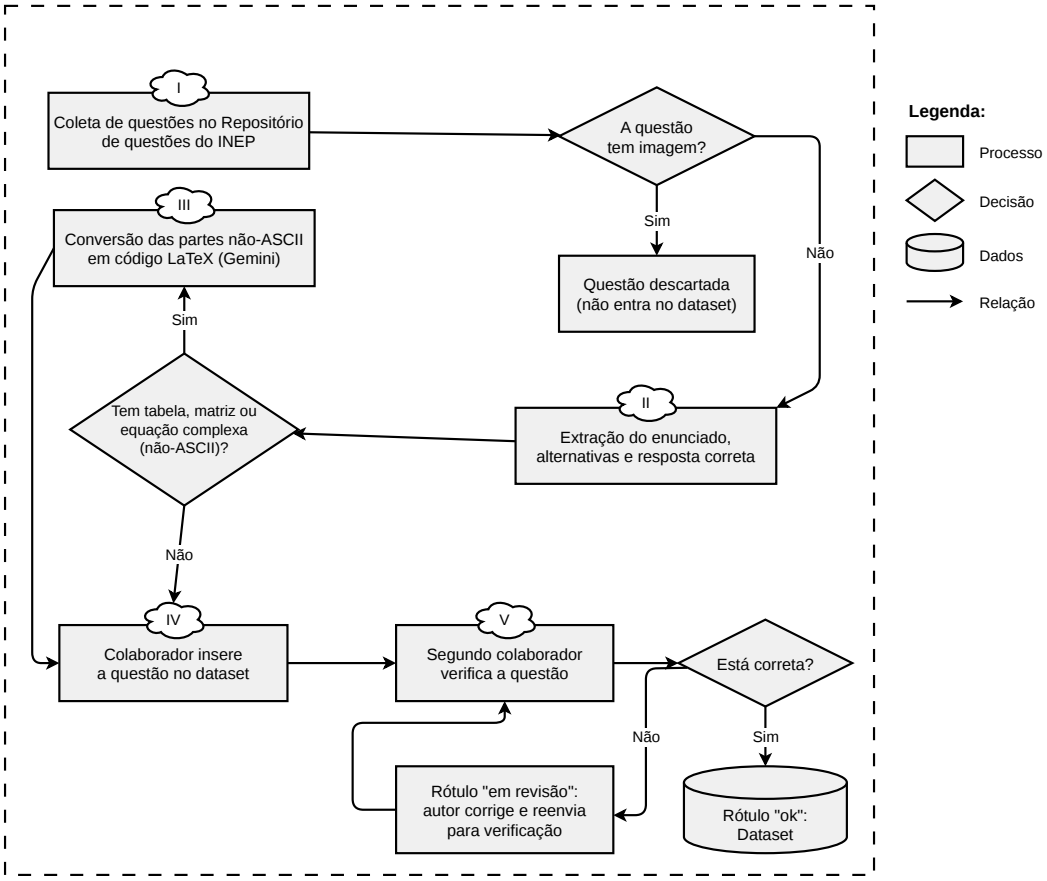


Figura 1. Fluxograma do processo de criação do *dataset*

desnecessários os sub-agentes de busca e de inspeção de arquivos do desenho original e isola um único sub-agente relevante: o executor de código (doravante, *Coder*). Segundo, a resposta é sempre uma alternativa de múltipla escolha (A–E), o que permite avaliar a acurácia de ponta a ponta de forma não ambígua.

O estudo varia dois fatores de forma controlada. O primeiro é a arquitetura: comparamos uma linha de base de invócação direta da LLM com dois sistemas multiagente que separam planejamento de execução: Modo 1 (delegação pura) e Modo 2 (resolver e verificar). O segundo fator é a escala do modelo: as configurações são avaliadas com um modelo de 20 bilhões e um de 120 bilhões de parâmetros da mesma família. Mantendo fixos o *prompt*, o nível de raciocínio, o conjunto de questões e a infraestrutura de execução, atribuímos as diferenças de desempenho à arquitetura e à escala, e não a fatores de implementação.

4.1. Modelos e configuração de raciocínio

Utilizamos os modelos abertos *gpt-oss-20b* e *gpt-oss-120b* [OpenAI 2025], servidos pelo GroqCloud [Groq, Inc. 2026a], uma plataforma de inferência como serviço que expõe os modelos por meio de uma interface de programação de aplicações (API) compatível com o formato OpenAI. O nível de raciocínio explícito é controlado por um parâmetro de esforço de raciocínio que mapeia diretamente para as configurações estudadas por

[Żywot et al. 2026]: sem pensamento, pensamento apenas no orquestrador, pensamento apenas nos sub-agentes ou em todos os componentes. Adotamos como configuração principal o pensamento apenas no orquestrador, que os autores identificam como a posição mais benéfica do raciocínio explícito em sistemas multiagente: pensar no orquestrador melhora a decomposição da tarefa, ao passo que pensamento irrestrito nos sub-agentes tende a desestabilizar a orquestração de ferramentas. Todas as gerações utilizam temperatura zero, para garantir determinismo e reprodutibilidade das comparações. As duas configurações comparadas neste estudo compartilham a mesma base de componentes. Como mostra a Figura 2, essa base reúne um orquestrador que coordena, um agente executor de código (*Coder*) que opera em contexto isolado, uma memória estruturada com o estado da tarefa e um ambiente *sandbox* para execução de código Python. A linha de base *Direct* não envolve agentes auxiliares e é descrita apenas em texto. As duas configurações multiagente diferem entre si apenas no *protocolo de comunicação* entre orquestrador e Coder, conforme sintetizado na Tabela 1.

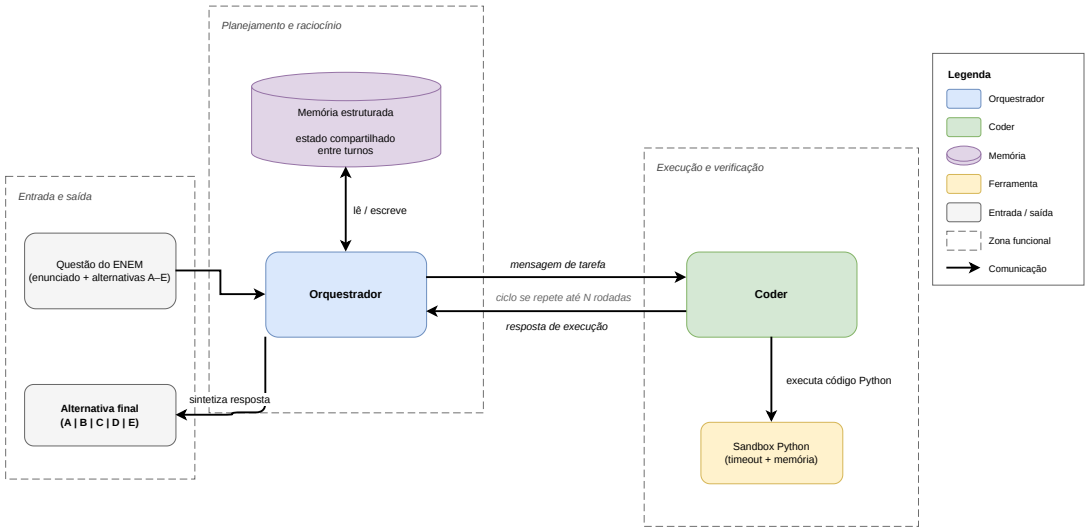


Figura 2. Arquitetura compartilhada pelas duas configurações multiagente.

A separação visual entre as zonas tracejadas na Figura 2 reflete a divisão de responsabilidades: a zona de *planejamento e raciocínio* concentra o orquestrador e o estado compartilhado; a zona de *execução e verificação* concentra o *Coder* e o *sandbox* de execução; e a zona de *entrada e saída* engloba a questão recebida e a alternativa final emitida. As subseções seguintes descrevem, na ordem, a linha de base *Direct* e as duas configurações multiagente.

4.2. Arquiteturas comparadas

4.2.1. *Direct* (linha de base)

A linha de base é uma invócação direta da LLM, sem agência adicional: para cada questão, emite-se uma única chamada à API do GroqCloud contendo o enunciado e as alternativas, com instruções para que o modelo raciocine passo a passo no estilo *chain-of-thought* [Wei et al. 2022] e indique a alternativa correta. Não há ferramentas externas, múltiplos turnos nem componentes auxiliares; o desempenho aqui reflete a capacidade do

Tabela 1. Protocolo de comunicação entre orquestrador e *Coder* nas duas configurações multiagente. As linhas marcam cada decisão de protocolo que distingue os modos.

Aspecto do protocolo	Modo 1 (Delegação pura)	Modo 2 (Resolver-e-verificar)
Papel do orquestrador	Planeja e delega; não resolve a questão internamente	Resolve por raciocínio e reconcilia o resultado
Mensagem ao *Coder*	Subtarefa autocontida (sem o enunciado)	Alternativa candidata e tarefa de verificação
Resposta do *Coder*	Resumo do resultado computado	Resultado independente da execução
Critério de saída	Quando o orquestrador julga ter informação suficiente	Quando o resultado computado confirma a alternativa candidata
Disparador do ciclo	Necessidade de nova subtarefa	Discordância entre candidata e resultado

modelo por si só, e fornece o controle contra o qual o ganho das arquiteturas multiagente é medido.

4.2.2. Modo 1

No sistema multiagente (*MAS*) em Modo 1, o orquestrador é responsável exclusivamente por planejamento e coordenação e não executa código. Ao receber a questão, ele produz uma análise inicial e, quando um cálculo é necessário, emite uma subtarefa autocontida ao *Coder*. O *Coder* opera em contexto isolado: recebe apenas a descrição da subtarefa, não vê o enunciado da questão, escreve código Python, executa-o em ambiente *sandbox* com limites de tempo e memória e devolve um resumo do resultado. O orquestrador então reconstrói o estado da tarefa a partir da memória estruturada e decide entre emitir nova subtarefa ou sintetizar a resposta final. Este desenho separa estritamente planejamento de execução e replica a arquitetura multiagente canônica do trabalho de origem.

4.2.3. MAS Modo 2

O Modo 2 introduz uma variante destinada a atacar um modo de falha conhecido em agentes que raciocinam explicitamente em tarefas matemáticas. Quando o raciocínio é habilitado, o agente tende a resolver a questão internamente e a apresentar uma resposta final sem nunca recorrer à ferramenta de código; nesse percurso, erros aritméticos no traço de raciocínio deixam de ser detectados, levando a respostas incorretas que poderiam ter sido evitadas por uma checagem trivial em código. [Żywot et al. 2026] relatam manifestações desse padrão em GAIA, observando que o pensamento irrestrito frequentemente leva os agentes a pular passos de verificação e a finalizar de forma prematura. O Modo 2 impõe uma verificação por construção: dissocia, na arquitetura, quem resolve de quem verifica, de modo que a checagem externa por execução de código seja parte do protocolo e não uma decisão do agente.

O protocolo possui três fases. Na fase de resolução, o orquestrador resolve a questão por raciocínio e emite, de forma estruturada, dois itens: a alternativa candidata e uma tarefa de cálculo autocontida cuja execução independente confirmaria essa candidata. Na fase de verificação, o *Coder* implementa e executa esse cálculo em Python, em ambiente isolado, e devolve o resultado. Na fase de reconciliação, o orquestrador compara o resultado computado com a candidata e decide confirmar ou revisar a resposta; em caso de revisão, o ciclo se repete por até um número máximo de rodadas. O princípio central é separar, no nível da arquitetura, o raciocínio do orquestrador da computação do *Coder*, de forma que a execução externa funcione como uma checagem cruzada dos erros aritméticos do traço de raciocínio. Esse mecanismo se relaciona à linha de programas como verificadores para raciocínio matemático [Toh et al. 2024, Zhou et al. 2023], mas a incorpora a uma arquitetura multiagente com contexto isolado, em vez de um agente único do tipo *sample-then-verify*.

4.2.4. Memória estruturada

Nos modos multiagente, em vez de reenviar o histórico completo entre componentes, o sistema mantém uma memória estruturada compartilhada organizada em quatro categorias: análise da questão, passos anteriores, resultados de ferramentas e subobjetivos. Os componentes escrevem nessa memória, mas apenas o orquestrador a lê integralmente; a cada turno, ele reconstrói o estado atual da tarefa a partir dela. Esse mecanismo permite raciocínio coerente em múltiplos passos sem que os agentes mantenham contexto próprio entre turnos.

4.3. Conjunto de dados

Empregamos os 301 itens textuais descritos na Seção 3. Para a avaliação automática, um analisador segmenta as alternativas A–E e tolera variações de formatação, como marcadores em maiúsculas ou minúsculas e alternativas agrupadas em uma única linha.

4.4. Protocolo experimental

Cada configuração foi executada sobre os mesmos 301 itens, com temperatura zero e o mesmo nível de raciocínio. A métrica primária é a acurácia de ponta a ponta, isto é, a proporção de alternativas preditas que coincidem com o gabarito oficial. As comparações isolam o efeito da arquitetura entre as configurações de 20B e o efeito da escala entre os modelos de 20B e 120B.

4.5. Análise de eficiência e custo

Além da acurácia, registramos, por questão, os *tokens* de entrada e de saída, o número de chamadas ao modelo, o número de chamadas de ferramenta e a latência. O GroqCloud, como provedor de inferência, cobra separadamente os *tokens* de entrada e de saída, com preços publicados por milhão de *tokens* para cada modelo [Groq, Inc. 2026b]. Assim, o custo monetário de uma configuração sobre o conjunto inteiro é dado por

$$C \;=\; T_{\text{in}} \cdot p_{\text{in}} + T_{\text{out}} \cdot p_{\text{out}}, \tag{1}$$

em que T_{in} e T_{out} são os totais de *tokens* observados sobre as 301 questões e p_{in}, p_{out} são os preços por *token* do modelo correspondente. Para que o custo reflita apenas o trabalho

útil na comparação entre arquiteturas, contabilizamos somente os *tokens* consumidos em execuções limpas, isto é, sem repetições de chamada decorrentes de falhas de formato no provedor. Reportamos custo total por configuração, custo por questão e custo por questão correta, este último enquanto medida direta do compromisso entre acurácia e eficiência.

5. Resultados e Discussão

A Tabela 2 resume o desempenho das quatro configurações sobre as 301 questões, contrapondo acurácia e custo. Para cada configuração reportamos a acurácia geral, a média de *tokens* por questão, o custo total de processar o conjunto e o custo por questão correta.

Tabela 2. Acurácia e custo das quatro configurações sobre as 301 questões de matemática do ENEM. Custos em dólares (US$), segundo os preços do GroqCloud [Groq, Inc. 2026b]

Configuração	Modelo	Acurácia	*Tokens*/questão	Custo total	Custo/acerto
Direct (CoT)	20B	91,7%	676	0,032	0,000117
MAS Modo 1	20B	95,7%	1814	0,083	0,000288
MAS Modo 2	20B	97,7%	2235	0,111	0,000379
Direct (CoT)	120B	97,7%	898	0,092	0,000314

Sob escala fixa de 20 bilhões de parâmetros, as arquiteturas multiagente superam a linha de base *Direct*, com o Modo 2 obtendo o maior ganho, de 91,7% para 97,7% de acurácia. Mais relevante, o Modo 2 sobre o modelo de 20B *iguala* a acurácia da invócação direta sobre o modelo de 120B, um modelo seis vezes maior. O resultado corrobora, em domínio e idioma distintos dos do trabalho de origem [Żywot et al. 2026], a hipótese de que a coordenação entre componentes especializados extrai de um modelo pequeno um desempenho comparável ao de um modelo substancialmente maior.

A leitura conjunta com o custo, contudo, qualifica esse ganho. O Modo 2 consome cerca de três vezes mais *tokens* por questão que a linha de base de mesmo porte e, em custo absoluto, supera até a invócação direta do modelo de 120B. A vantagem do sistema multiagente pequeno, portanto, não reside no custo monetário por inferência, mas na possibilidade de operar com um modelo de menor porte: alcançando a mesma acurácia do modelo de 120B com um modelo de 20B, o Modo 2 viabiliza execução local e os benefícios de latência, privacidade e independência de infraestrutura que dela decorrem.

6. Limitações e Reprodutibilidade

Reconhecemos três limitações que delimitam o alcance das conclusões deste estudo. (I) O conjunto avaliado é restrito a questões autocontidas e puramente textuais; itens que exigiam compreensão de gráficos ou figuras foram descartados. Como demonstrado por [Pires et al. 2023], até mesmo modelos de fronteira apresentam desempenho substancialmente inferior em itens multimodais do ENEM. Portanto, as acurácias reportadas devem ser interpretadas como um limite superior circunscrito ao subconjunto textual da prova, e não como uma estimativa direta do desempenho esperado no exame completo. (II) A validade externa das métricas de eficiência e a análise de custos estão atreladas a uma infraestrutura específica. O estudo avaliou apenas a família de modelos *gpt-oss* servida pelo GroqCloud; logo, ganhos exatos de desempenho e custos monetários absolutos estão sujeitos à volatilidade de preços do provedor e a particularidades dessa arquitetura de

pesos. Contudo, as proporções de consumo de *tokens* tendem a permanecer válidas por derivarem da configuração arquitetural dos agentes. (III) A reprodutibilidade estrita é um desafio inerente ao uso de APIs de inferência. Apesar do uso de temperatura zero para maximizar o determinismo, flutuações em *backends* de nuvem como roteamento dinâmico e otimizações subjacentes de *hardware* podem introduzir variações pontuais nos *tokens* gerados. Não obstante, a agregação dos resultados sobre as 301 questões torna as tendências de ganho arquitetural e as conclusões do estudo robustas a essa variabilidade.

7. Conclusão

Este trabalho investigou se uma arquitetura multiagente leve pode melhorar o raciocínio matemático de modelos de linguagem pequenos em questões textuais do ENEM. Para isso, construímos um conjunto de 301 itens autocontidos de Matemática, com enunciados, alternativas e gabaritos validados, e comparamos uma linha de base de invocação direta com duas configurações multiagente baseadas em orquestração centralizada e execução de código. A principal diferença do Modo 2 foi tornar a verificação externa parte obrigatória do protocolo, separando o agente que raciocina do agente que calcula.

Os resultados indicam que a organização arquitetural pode compensar parte da ausência de escala. Em particular, o Modo 2 com modelo de 20 bilhões de parâmetros alcançou 97,7% de acurácia, igualando a invocação direta do modelo de 120 bilhões e superando a linha de base direta de 20 bilhões, que obteve 91,7%. Esse ganho, contudo, não ocorreu sem custo: as configurações multiagente consumiram mais *tokens* e apresentaram maior custo por questão correta. Assim, a principal contribuição do sistema não está em reduzir o custo monetário da inferência, mas em mostrar que modelos menores podem atingir desempenho de modelos maiores quando apoiados por decomposição de tarefas e verificação computacional.

Como trabalhos futuros, pretendemos ampliar o conjunto para questões multimodais, analisar erros por tipo de conteúdo matemático e avaliar a arquitetura em outras famílias de modelos e infraestruturas de inferência. Essas extensões podem esclarecer em que condições sistemas multiagente com modelos pequenos oferecem uma alternativa prática, reprodutível e privada para tarefas educacionais de raciocínio matemático.

Referências

Almeida, T. S., Laitz, T., Bonás, G. K., and Nogueira, R. (2023). BLUEX: A benchmark based on Brazilian leading universities entrance exams. *arXiv preprint arXiv:2307.05410*.

Belcak, P., Heinrich, G., Diao, S., Fu, Y., Dong, X., Muralidharan, S., Lin, Y. C., and Molchanov, P. (2025). Small language models are the future of agentic AI. arXiv preprint arXiv:2506.02153.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., and Hoefler, T. (2024). Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. (2023). Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. (2022). Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Huang, M., Duan, N., and Chen, W. (2024). ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*.

Groq, Inc. (2026a). GroqCloud: Inference-as-a-service platform. `https://console.groq.com/docs`. Acessado em 2026.

Groq, Inc. (2026b). GroqCloud pricing. `https://groq.com/pricing`. Acessado em 2026.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. (2021). Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, volume 34.

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., and Zhou, D. (2024). Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*.

Nunes, D., Primi, R., Pires, R., Lotufo, R., and Nogueira, R. (2023). Evaluating GPT-3.5 and GPT-4 models on Brazilian university admission exams. arXiv preprint arXiv:2303.17003.

OpenAI (2025). GPT-OSS-20B and GPT-OSS-120B: Open-weight reasoning models. *arXiv preprint arXiv:2508.12461*.

Pires, R., de Souza, T. S., Ribeiro, P. H., and Lotufo, R. (2023). Evaluating GPT-4's vision capabilities on Brazilian university admission exams. arXiv preprint arXiv:2311.14169.

Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652.

Toh, V. et al. (2024). Not all votes count! programs as verifiers improve self-consistency of language models for math reasoning. *arXiv preprint arXiv:2410.12608*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Zhou, A., Wang, K., Lu, Z., Shi, W., Luo, S., Qin, Z., Lu, S., Jia, A., Song, L., Zhan, M., and Li, H. (2023). Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*.

Żywot, A., Chen, X., and de Rijke, M. (2026). Can small agent collaboration beat a single big LLM? *arXiv preprint arXiv:2601.11327*.