

YUMA: Geração de Workflows Multiagentes a partir de Linguagem Natural

André Neves, Daniel Mantovani, Caíque Campelo, Gabriel Souza, Débora Souza, Beatriz Furtado,
Leandro Balby, Eliane Araújo
Universidade Federal de Campina Grande
Campina Grande, Brasil
{andre.neves,daniel.soares,caique.campelo,gabriel.souza,debora.souza,anna.beatriz.furtado}@ccc.ufcg.edu.br
{lbmarinho,eliane}@computacao.ufcg.edu.br

ABSTRACT

Large Language Models (LLMs) have enabled the translation of natural language into functional systems. However, creating multi-agent systems based on these models still requires advanced technical knowledge. In this work, we present YUMA, a multi-agent tool that simulates software engineering roles — Requirements Engineer and Software Architect — to collaboratively generate multi-agent systems. This approach translates users' informal requirements into deployable workflows, with human supervision at each step, ensuring transparency and validation. The tool coordinates requirements elicitation, architecture definition, and system generation, reducing the technical barrier to creating solutions. Finally, we present the tool's architecture, the interaction flow, and a usage example, demonstrating how this approach democratizes access to multi-agent technologies, making them accessible and transparent to non-specialized users.

KEYWORDS

Sistemas Multiagentes, Geração Automática de Sistemas Multiagentes, Orquestração de Agentes, Vibe Coding

1 INTRODUÇÃO

Nos últimos anos, diversos estudos destacaram o potencial dos Grandes Modelos de Linguagem (LLMs) para gerar código a partir de descrições em linguagem natural, especialmente em contextos pontuais como trechos de código ou funções isoladas [1–3]. Esses avanços despertaram interesse em integrar tais modelos ao ciclo tradicional da engenharia de software, desde a elicitação de requisitos até a validação [4, 5]. Quando dotados da capacidade de interagir com ferramentas externas, esses modelos tornam-se agentes capazes de compreender contextos, decidir ações futuras e executar interações complexas com sistemas externos. Incorporados a arquiteturas multiagentes alimentadas por LLMs (Large Language Model-powered Multi-Agent Systems, ou LMA), esses agentes potencializam ainda mais as oportunidades de automação e colaboração no desenvolvimento de sistemas, pois combinam a capacidade dos LLMs de compreender e gerar linguagem natural com a coordenação especializada dos sistemas multiagentes, ampliando significativamente o potencial de colaboração inteligente em tarefas complexas. [6].

In: XXIV Workshop de Ferramentas e Aplicações (WFA 2025). Anais Estendidos do XXXI Simpósio Brasileiro de Sistemas Multimídia e Web (WFA'2025). Rio de Janeiro/RJ, Brasil. Porto Alegre: Brazilian Computer Society, 2025.

© 2025 SBC – Sociedade Brasileira de Computação.

ISSN 2596-1683

Apesar do progresso, criar e operar agentes baseados em LLMs ainda depende de desenvolvedores que conheçam bem o código-fonte, integrem APIs e dominem a engenharia de prompts, além de frameworks como o LangChain, LangGraph e CrewAI. Tentativas de simplificar esse processo em plataformas low/no-code — por exemplo, OpenHands [7], MetaGPT [8] e AutoGen [9] — esbarram em orquestrações pouco claras, fluxos difíceis de inspecionar e pouca abertura para que o usuário intervenha durante a execução. Outros projetos, como CodeAgent [10] e AutoDev [11], continuam voltados a tarefas muito específicas ou exigem um nível alto de conhecimento técnico, o que limita seu alcance e adoção.

Falta, contudo, uma solução que transforme pedidos em linguagem simples, feitos por quem não é da área, em sistemas multiagentes completos, permitindo que o usuário acompanhe e valide cada passo por meio de um processo human-in-the-loop. Embora algumas ferramentas LMA (Large Language Model-powered Multi-Agent Systems) já existam, elas ainda são escassas e pouco maduras: geralmente focam em tarefas pontuais, exigem conhecimento técnico avançado ou oferecem pouca flexibilidade na construção colaborativa dos sistemas. LMA, refere-se a arquiteturas multiagentes nas quais agentes autônomos colaboram e se comunicam utilizando grandes modelos de linguagem para raciocínio, coordenação e execução de tarefas complexas. As ferramentas disponíveis apenas geram trechos de código — caso do HULA [12] — ou exigem que o próprio usuário construa toda a arquitetura, como ocorre no MAHILo¹. Essa distância entre a intenção inicial e um sistema executável ainda limita a adoção prática de soluções LMA, sobretudo quando a colaboração ágil entre pessoas e agentes é indispensável.

Neste trabalho, apresentamos a ferramenta YUMA, que proporciona uma abordagem semiautomática baseada em interações de chat entre o YUMA e o desenvolvedor para converter descrições em linguagem natural em um sistema multiagentes executável. O usuário dialoga com agentes especializados que, de forma supervisionada, levantam e refinam requisitos de maneira iterativa até chegar a uma arquitetura que satisfaça as necessidades do desenvolvedor. A partir desse modelo, o sistema é gerado automaticamente e importado para uma plataforma low-code, onde pode ser validado, testado e implantado. A abordagem combina os princípios das metodologias Gaia [13] que fornece modelos de papéis e protocolos para definir responsabilidades e interações dos agentes, e de MASE [14], cuja abordagem orientada a metas apoia a análise e o refinamento da arquitetura multiagente até a geração

¹<https://github.com/wjayesh/mahilo>

do sistema executável, assegurando validação humana em todas as etapas – da concepção inicial ao funcionamento final do sistema.

Assim, a principal contribuição deste trabalho é disponibilizar uma ferramenta que aproxima a engenharia de sistemas multiagentes da linguagem natural, permitindo que usuários sem conhecimento técnico avançado construam, inspecionem e validem arquiteturas LMA de forma colaborativa e transparente.

2 A FERRAMENTA YUMA

Esta seção descreve a arquitetura e principais funcionalidades da plataforma YUMA. O nome YUMA deriva de *Your User-guided Multi-agent System*, refletindo seu objetivo central: permitir que usuários sem formação técnica consigam especificar e implantar workflows multiagentes impulsados por LMA. Diferentemente de frameworks como LangChain e soluções low/no-code existentes – que ainda dependem de desenvolvedores especializados –, a plataforma YUMA foi concebida com o objetivo de integrar linguagem natural ao processo de construção desses workflows multiagentes. Dessa forma, busca-se reduzir a barreira de entrada para a criação de workflows inteligentes, aproximando a intenção expressa em linguagem natural da geração automatizada de sistemas multiagentes funcionais.

2.1 Arquitetura da Ferramenta

A organização modular da solução busca garantir flexibilidade, escalabilidade e facilidade de manutenção. Conforme ilustrado na Figura 1, a ferramenta é estruturada em módulos integrados, que conduzem o processo desde a definição inicial do sistema até sua execução em uma plataforma de destino. Cada módulo possui responsabilidades bem definidas, detalhadas na subseção a seguir.

2.1.1 Módulos Funcionais. A arquitetura da ferramenta YUMA está organizada em quatro módulos encadeados, cada um responsável por uma etapa distinta da geração de sistemas multiagentes. Essa organização funcional garante clareza na divisão de responsabilidades entre agentes e facilita a extensibilidade do sistema.

O primeiro módulo é responsável por levantar e refinar os requisitos iniciais do sistema a partir de entradas de alto nível fornecidas pelo usuário. Essa atividade é conduzida pelo agente *Engenheiro de Requisitos*, que interage com o usuário de forma supervisionada por meio de um diálogo orientado. Durante essa interação, são esclarecidos os objetivos do sistema, as funcionalidades desejadas, as tecnologias pretendidas e as restrições envolvidas. Ao final do processo, obtém-se uma especificação textual detalhada com as intenções do usuário e requisitos definidos.

No módulo de proposta de arquitetura, o agente *Arquiteto de Sistemas Multiagentes* é responsável por propor uma organização inicial do sistema com base nos requisitos definidos na etapa anterior. Para isso, ele interage com o usuário a fim de validar e aperfeiçoar a proposta, garantindo que a arquitetura esteja alinhada com as necessidades identificadas. O resultado dessa etapa é a apresentação de uma estrutura arquitetural preliminar, acompanhada de uma descrição dos principais componentes, suas interações e respectivas responsabilidades.

Na terceira etapa, o sistema passa à geração do workflow, isto é, a construção do sistema multiagente com base na arquitetura previamente validada. Essa fase é executada de forma autônoma por agentes especializados, como o *Gerador de Nós* e o *Gerador de*

Arestas, que são responsáveis por estruturar o workflow do sistema. Esses agentes definem os nós – representando unidades funcionais como Agentes, LLMs, APIs ou interfaces com o usuário – e estabelecem as conexões entre eles, de acordo com as interações previstas na arquitetura. O resultado é a criação de dois artefatos principais: um diagrama de sequência, que ilustra o fluxo de comunicação entre os componentes, e um arquivo *YAML*, que descreve formalmente a topologia do grafo, os componentes envolvidos e suas interações.

Por fim, o módulo de deploy é responsável por exportar o arquivo *YAML* gerado para a plataforma de execução, atualmente integrada ao ambiente no-code *Dify*². Apesar do suporte nativo ao *Dify*, essa etapa foi projetada de forma modular e independente do núcleo funcional do sistema, o que possibilita sua substituição por outras plataformas low/no-code ou ambientes de execução compatíveis, sem comprometer o funcionamento geral da solução.

2.1.2 Modelagem com LangGraph e Interações Técnicas. A implementação da ferramenta YUMA criada usando a biblioteca LangGraph, escolhida por permitir a modelagem de agentes como nós em grafos dirigidos, inspirando-se na arquitetura modular do OpenHands. A separação de responsabilidades é feita por meio de subgrafos especializados: um voltado à interação com o usuário, incluindo nós como requirements-engineer, architecture-agent e human-node; e outro responsável pela geração autônoma do workflow.

Cada agente opera de forma reativa, processando o estado atual e respondendo por meio de modelos LLM com suporte a chamada de ferramentas, como os disponibilizados pelas APIs da OpenAI e DeepInfra. Os prompts são personalizados por funções parametrizadas e mantêm o contexto das interações anteriores para garantir continuidade e coerência no raciocínio. As respostas dos agentes são usadas para gerar automaticamente os arquivos que definem o workflow, incluindo representações em *YAML* e diagramas de sequência.

A exportação para o *Dify* é suportada nativamente, mas tratada como um componente desacoplado do núcleo do sistema, o que permite substituí-la por outras plataformas de deploy ou visualização. Essa arquitetura flexível, sustentada por subgrafos independentes e componentes reutilizáveis, facilita a evolução da ferramenta, a substituição de agentes e a integração com diferentes modelos ou ambientes de execução.

3 EXEMPLO DE USO

O vídeo de demonstração³ do sistema apresenta todas as funcionalidades descritas nessa seção. Para ilustrar o uso da ferramenta, consideramos um cenário comum no contexto acadêmico: a realização de uma Revisão Sistemática da Literatura (RSL), atividade recorrente entre alunos de pós-graduação e pesquisadores envolvidos na produção científica. A RSL é fundamental para mapear o estado da arte sobre um determinado tema, mas costuma ser um processo trabalhoso, exigindo a busca em múltiplas bases de dados, além da triagem e classificação criteriosa de artigos. Com o objetivo de simplificar e acelerar esse processo, utilizamos a ferramenta

²<https://dify.ai>

³https://drive.google.com/file/d/1u8qPDcgi_8eKoeSKEb73GsAInkFHGcfN/view?usp=sharing

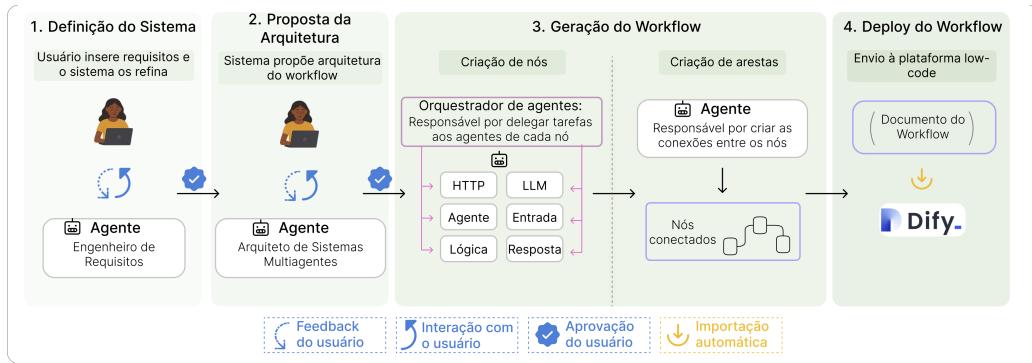


Figure 1: Visão geral da arquitetura da ferramenta YUMA, composta por quatro módulos principais: (1) Definição do Sistema, (2) Proposta da Arquitetura, (3) Geração do Workflow e (4) Deploy do Workflow.



Figure 2: Workflow de sistema de Revisão Sistemática da Literatura gerado automaticamente pelo sistema YUMA.

proposta para gerar automaticamente um sistema baseado em linguagem natural, capaz de auxiliar na execução da RSL de forma mais intuitiva, transparente e eficiente.

Antes de iniciar a construção do sistema, é necessário preparar o ambiente de execução da ferramenta, que atualmente é disponibilizada como uma interface de linha de comando (CLI). Para seu funcionamento adequado, é preciso configurar um arquivo .env com as variáveis de ambiente contendo as chaves de acesso ao modelo de linguagem selecionado (neste exemplo, LLaMA), bem como as credenciais necessárias para a integração com as APIs DoDify.

Com o ambiente configurado, o sistema YUMA pode ser inicializado e o usuário entrará na interface de chat do sistema. Na etapa inicial, o usuário conversa com o engenheiro de requisitos:

Usuário: Preciso de um sistema multiagente que auxilie na revisão sistemática da literatura sobre o uso de sistemas multiagentes na educação. O sistema deve apoiar na formulação da query de busca, na definição das questões de pesquisa, no acesso a bases científicas (como IEEE, ACM, arXiv e Google Acadêmico via HTTP) e na classificação das publicações como relevantes ou não, com base no alinhamento ao tema. Ao final, deve retornar uma lista marcada com a relevância de cada resultado.

Após o usuário inserir qual sistema ele deseja criar, o agente Engenheiro de Requisitos pede que o usuário especifique melhor o seu sistema:

Engenheiro de Requisitos:

Para começar a definir o sistema multiagente para auxiliar na revisão sistemática da literatura sobre o uso de sistemas multiagentes no contexto educacional, gostaria de saber mais sobre como você imagina a interação entre os agentes e as bases de indexação científica. Algumas perguntas específicas para esclarecer os requisitos:

1. Critérios de Relevância: Quais são os critérios que você utilizará para classificar as pesquisas como relevantes ou irrelevantes?
2. Geração de Query de Busca: Como o sistema deve auxiliar na construção da query?
3. Acesso a Bases de Indexação: Há outras bases além das citadas? Há restrições de acesso?
4. Classificação de Relevância: Deve ser automática ou assistida?
5. Cenários de Uso: Descreva um caso de uso típico.

Essas informações ajudarão a esclarecer os requisitos e garantir que o sistema atenda às suas necessidades.

Usuário: O sistema deve apoiar revisões sistemáticas sobre sistemas multiagentes na educação, sugerindo queries de busca refináveis, acessando bases científicas via HTTP (com uso opcional de scraping), e classificando publicações com auxílio de modelos de linguagem. A relevância será definida com base em palavras-chave e escopo, e o usuário poderá revisar as classificações. O fluxo inclui geração de query, recuperação de artigos, classificação assistida e exportação dos resultados.

Após essa interação, o usuário pede ao engenheiro de requisitos que ele prossiga para o agente Arquiteto do Sistema. O arquiteto propõe então a seguinte arquitetura para o sistema:

Arquiteto do Sistema:

Nós do sistema:

- `usuario_interface`: Interface de usuário para entrada de palavras-chave e critérios de busca
- `busca_paralela`: Realiza buscas paralelas e simultâneas em diferentes bases de indexação científica
- `gerador_questoes`: Gera questões de pesquisa com base nas pesquisas relevantes identificadas
- `screeningAutomatico`: Realiza screening automático dos artigos com base no título e resumo
- `exibir_resultados`: Apresenta os resultados para o usuário

Interações:

- (1) `usuario_interface` → `busca_paralela`: Envia palavras-chave e critérios de busca para a busca paralela
- (2) `busca_paralela` → `gerador_questoes`: Fornece pesquisas relevantes para geração de questões de pesquisa
- (3) `gerador_questoes` → `screeningAutomatico`: Fornece artigos para screening automático
- (4) `screeningAutomatico` → `exibir_resultados`: Fornece resultado do screening automático para atualização da lista de pesquisas

O usuário pode refinar a arquitetura proposta conversando com o agente Arquiteto do Sistema ou prosseguir para geração inserindo o comando “Prossiga para a geração”. Ao prosseguir para geração, o aplicativo Dify é aberto no browser e o fluxograma é importado automaticamente nele. A Figure 2 apresenta o sistema gerado.

4 TIPO DE LICENÇA

O tipo de licença que protege toda a solução é a GNU General Public License (GPL v3)⁴. Essa licença garante a liberdade de uso, estudo, modificação e redistribuição da solução, seja para fins comerciais ou não comerciais, desde que quaisquer versões modificadas ou derivadas mantenham a mesma licença. Assim, o Yuma preserva o espírito de software livre e promove a colaboração aberta, assegurando que melhorias e adaptações feitas pela comunidade permaneçam acessíveis a todos.

5 CONCLUSÕES

A ferramenta YUMA, proposta neste trabalho, permite a geração automática de workflows multiagentes a partir de linguagem natural, permitindo que usuários não técnicos participem do processo de desenvolvimento por meio de interações com agentes especializados. A arquitetura modular da ferramenta, baseada em subgrafos com agentes reativos, viabiliza a elicitação, refinamento e validação supervisionada de requisitos até a geração de artefatos formais prontos para execução em plataformas low-code. Mesmo em estágio inicial, a ferramenta já se mostra promissora ao reduzir a distância entre intenção e implementação, especialmente em tarefas

complexas como revisões sistemáticas da literatura. Como trabalhos futuros, estão previstas melhorias na interface, ampliação do suporte a outros ambientes de execução e condução de estudos de usabilidade com usuários reais, com o objetivo de aprimorar a experiência e validar o impacto da solução em contextos práticos.

Limitações e Trabalhos Futuros. Embora este trabalho apresente um protótipo funcional em estágio inicial, sua principal contribuição é demonstrar a viabilidade da abordagem proposta. A implementação atual está restrita ao Dify e utiliza uma interface em linha de comando, escolhas motivadas pela simplicidade de prototipagem e validação das funcionalidades centrais. Reconhecemos também limitações na complexidade dos workflows gerados, na interpretação de requisitos muito ambíguos e na ausência de métricas de desempenho e escalabilidade. Como trabalhos futuros, pretendemos ampliar o suporte a outras plataformas de geração, oferecer interface web, e realizar estudos empíricos em cenários reais, considerando tempo de resposta, custo, consumo de recursos e comparações entre diferentes sistemas.

AGRADECIMENTOS

Este trabalho foi financiado por meio do projeto Agents4Good, uma colaboração entre a Kunumi e a unidade EMBRAPII UFCG-CEEI (Universidade Federal de Campina Grande).

REFERÊNCIAS

- [1] X. Du et al., “Evaluating large language models in class-level code generation,” in *Proc. IEEE/ACM 46th Int. Conf. on Software Engineering*, New York, NY, USA, ACM, 2024, pp. 1–13.
- [2] J. Jiang et al., “A survey on large language models for code generation,” *arXiv preprint arXiv:2406.00515*, 2024.
- [3] D. Souza et al., “Code Generation with Small Language Models: A Deep Evaluation on Codeforces,” *arXiv preprint arXiv:2504.07343*, 2025.
- [4] C. E. Jimenez et al., “Swe-bench: Can language models resolve real-world github issues?” *arXiv preprint arXiv:2310.06770*, 2023.
- [5] L. Belzner, T. Gabor, and M. Wirsing, “Large language model assisted software engineering: prospects, challenges, and a case study,” in *Int. Conf. on Bridging the Gap between AI and Reality*, Springer, 2023, pp. 355–374.
- [6] J. He, C. Treude, and D. Lo, “LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead,” *ACM Trans. Softw. Eng. Methodol.*, 2024, doi:10.1145/3712003.
- [7] X. Wang et al., “Openhands: An open platform for AI software developers as generalist agents,” in *The 13th Int. Conf. on Learning Representations*, 2024.
- [8] S. Hong et al., “MetaGPT: Meta programming for multi-agent collaborative framework,” *arXiv preprint arXiv:2308.00352*, vol. 3, no. 4, 2023, p. 6.
- [9] V. Dibis et al., “Autogen studio: A no-code developer tool for building and debugging multi-agent systems,” in *Proc. 2024 Conf. on Empirical Methods in Natural Language Processing: System Demonstrations*, Miami, FL, USA, ACL, 2024, pp. 72–79.
- [10] K. Zhang et al., “Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges,” *arXiv preprint arXiv:2401.07339*, 2024.
- [11] M. Tufano et al., “AutoDev: Automated AI-driven development,” *arXiv preprint arXiv:2403.08299*, 2024.
- [12] W. Takerngsaksri et al., “Human-In-the-Loop Software Development Agents,” *arXiv preprint arXiv:2411.12924*, 2024.
- [13] F. Zambonelli, N. R. Jennings, and M. Wooldridge, “Developing multiagent systems: The Gaia methodology,” *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 3, pp. 317–370, 2003.
- [14] S. A. DeLoach, M. F. Wood, and C. H. Sparkman, “Multiagent systems engineering,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 11, no. 03, pp. 231–258, 2001.

⁴<https://www.gnu.org/licenses/gpl-3.0.pt-br.html>