Jarvas: um Chatbot Assistente via Aplicativos de Mensagens Instantâneas para Aplicações Científicas

Rafael Nogueira Rodrigues[1,3], Lucas Soares[1,3], Yang da Fontoura Rodrigues[1,3], Luiz Felipe Laviola[1,2,3], Diego Kreutz[1,2,3], Rodrigo Brandão Mansilha[1,2,3],

[1]Laboratório de Estudos Avançados em Computação (LEA)

[2]Programa de Pós-Graduação em Engenharia de Software (PPGES)

[3]Universidade Federal do Pampa (UNIPAMPA)

{rafaelnogueira, lucasfs2, yangrodrigues}.aluno@unipampa.edu.br,
luiz@laviola.dev, {diegokreutz, mansilha}@unipampa.edu.br

Resumo. *A crescente complexidade dos projetos de ciência de dados tem um impacto direto na comunicação das equipes de desenvolvimento. Por exemplo, o tempo de execução e treinamento de redes neurais profundas pode variar amplamente, de minutos a dias. Com base em resultados parciais, muitos desses processos poderiam ser interrompidos antecipadamente. Nesse contexto, propomos Jarvas: um chatbot criado para auxiliar equipes de desenvolvimento no acompanhamento de processos científicos computacionais via aplicativos de mensagem instantânea. Jarvas se integra ao fluxo de trabalho dos desenvolvedores, fornecendo atualizações em tempo real sobre o progresso dos processos, reduzindo a necessidade de monitoramento manual constante. Apresentamos uma implementação pública como prova de conceito e demonstramos sua utilidade por meio de um estudo de caso.*

Abstract. *The increasing complexity of data science projects has a direct impact on communication within development teams. For example, the execution and training times of deep neural networks can vary widely, ranging from minutes to days. Based on partial results, many of these processes could be stopped early. In this context, we propose Jarvas: a chatbot designed to assist development teams in monitoring computational scientific processes via instant messaging apps. Jarvas integrates into the developers' workflow, providing real-time updates on process progress, thus reducing the need for constant manual monitoring. We present a publicly available implementation as a proof of concept and demonstrate its usefulness through a case study.*

1. Introdução

A evolução e o aumento da complexidade nos projetos de software voltados à ciência de dados têm intensificado a necessidade de ferramentas que melhorem a comunicação e a gestão de processos entre os envolvidos. No desenvolvimento de soluções computacionais, como o treinamento de modelos de Inteligência Artificial (IA) e o processamento de grandes volumes de dados (*big data*), o monitoramento contínuo

do progresso das aplicações é um desafio constante, especialmente quando esses processos são demorados e exigem acompanhamento em tempo real. Nesse contexto, o uso de um chatbot [Adamopoulou and Moussiades 2020] para notificação e monitoramento pode ser uma solução eficaz para auxiliar na gestão de projetos.

Um exemplo relevante é o projeto Malware DataLab[1], fomentado pela Rede Nacional de Ensino e Pesquisa (RNP[2]), no âmbito do Programa Hackers do Bem[3]. O Malware DataLab tem como objetivo capacitar os chamados "hackers do bem" em tecnologias de IA generativa aplicadas ao combate à proliferação de *malwares* Android, como o MalSynGen [Nogueira et al. 2024a, Nogueira et al. 2024b]. Em testes iniciais da MalSynGen com estudantes, observamos o desafio de manter o engajamento durante o processo de ensino-aprendizagem, especialmente ao lidar com campanhas de experimentos e treinamentos de IA generativa, que podem durar desde alguns minutos até vários dias. Diante disso, estamos buscando maneiras de tornar o processo de ensino, pesquisa e extensão mais dinâmico e envolvente.

Neste trabalho, apresentamos o Jarvas, um chatbot desenvolvido para auxiliar no monitoramento de processos avançados de IA que exigem alto poder computacional. Integrado ao fluxo de trabalho dos desenvolvedores, o Jarvas oferece atualizações automáticas e em tempo real sobre o andamento das aplicações, reduzindo a necessidade de supervisão manual frequente. Utilizando aplicativos de mensagens instantâneas, o sistema envia notificações aos usuários durante a execução de processos prolongados, oferecendo uma forma de acompanhamento mais intuitiva (preferida pelas novas gerações) e pervasiva (imediatamente disponível para milhões de usuários). Além das atualizações automáticas, o Jarvas permite interação reversa, em que o usuário pode consultar a situação das operações por meio de redes sociais ou aplicativos de mensagens, sem depender de notificações automáticas. A disponibilização de uma API REST e de uma biblioteca Python amplia seu uso em diferentes contextos, tornando o Jarvas uma solução flexível para monitoramento de processos complexos.

A viabilidade técnica da solução é demonstrada por meio de uma implementação disponibilizada publicamente, cujas funcionalidades foram verificadas através de testes sistemáticos. Para comprovar a utilidade da solução, apresentamos um estudo de caso que aborda o uso do Jarvas no projeto Malware DataLab.

Até onde sabemos, Jarvas é o primeiro chatbot focado em auxiliar o desenvolvimento de soluções que envolvem ciência de dados, como o treinamento de Redes Neurais Profundas. As soluções mais próximas discutem o uso de tecnologias de comunicação anteriores, como email (ScriptPulse) [Rodrigues 2023], ou são voltadas para aprendizado em outras áreas [Colace et al. 2018], ou ainda abrangem chatbots mais genéricos [Luo et al. 2022]. A aplicação Tinybio[4] também pode ser considerada uma solução relacionada, já que é projetada para auxiliar pesquisadores na análise de dados científicos e execução de tarefas complexas. No entanto, enquanto o foco do Tinybio é apoiar a análise de dados e correção automática de códigos, o

[1] `https://malwaredatalab.github.io/`
[2] `https://www.rnp.br/`
[3] `https://hackersdobem.org.br/`
[4] `https://www.tinybio.cloud`

Jarvas é voltado para monitorar a execução de aplicações em tempo real.

No restante deste trabalho, apresentamos a arquitetura da Jarvas na Seção 2, seguida por uma implementação como prova de conceito na Seção 3 e um estudo de caso detalhado na Seção 4. Por fim, apresentamos as considerações finais e as perspectivas para trabalhos futuros na Seção 5.

2. Arquitetura do Jarvas

O objetivo geral do Jarvas é facilitar o monitoramento de processos computacionais complexos, especialmente aqueles de longa duração, como é comum em processos sofisticados de IA (e.g., arquiteturas de Redes Neurais Artificiais). Para atingir esse objetivo, projetamos uma arquitetura estruturada em três componentes principais, conforme ilustrado na Figura 1. A seguir, discutimos em detalhes os elementos arquiteturais, incluindo os papéis, os fluxos de informação e os componentes de software.

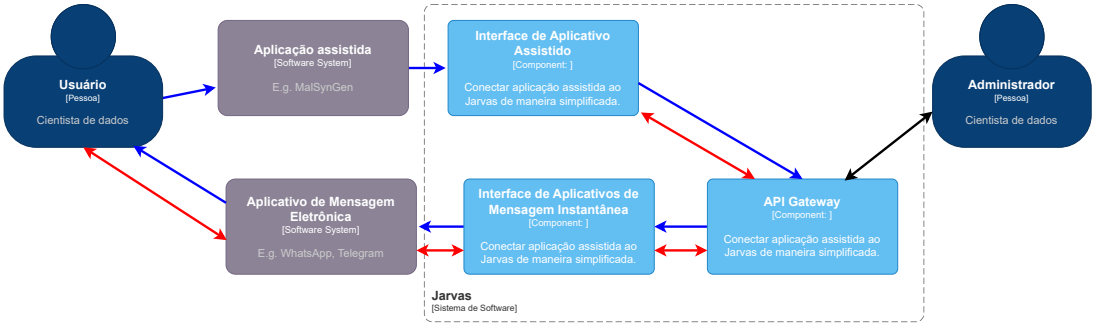

Figure 1. Arquitetura do Jarvas.

Definimos dois papéis principais: administrador e usuário. O **Administrador** é responsável pela configuração do ambiente de execução do Jarvas, incluindo aspectos relacionados à segurança, e esse papel geralmente é desempenhado pelo líder do grupo de desenvolvedores. O **Usuário** é aquele que acompanha o processo computacional com o auxílio do Jarvas, podendo ser qualquer membro da equipe, incluindo o líder.

Na Figura 1, as setas indicam os fluxos de informação entre os papéis e os componentes arquiteturais, sendo representados por cores distintas (preto, azul e vermelho). Para o **Administrador**, previmos um único fluxo: o de configuração do ambiente (setas pretas). Para o **Usuário**, identificamos dois tipos de fluxo: **Passivo** (setas azuis) e **Ativo** (setas vermelhas). No fluxo **Passivo**, o usuário recebe atualizações geradas automaticamente pelo Jarvas, de acordo com as configurações estabelecidas antes do início da execução. No fluxo **Ativo**, o usuário pode interagir diretamente com o sistema, enviando solicitações durante a execução do processo.

2.1. Aplicações de Mensagem Instantânea

A *Interface de Aplicativos de Mensagem Instantânea* permite a comunicação entre o Jarvas e seus usuários por meio de plataformas como o WhatsApp. Por esse canal, o usuário pode receber atualizações automáticas em tempo real sobre o status das

operações (fluxo **Passivo**) e também consultar manualmente o progresso de um processo quando necessário (fluxo **Ativo**).

Esse componente arquitetural funciona como um intermediário para APIs externas, como a API do WhatsApp e a API do Telegram, facilitando a integração com outros componentes do sistema. Além disso, simplifica futuras manutenções e expansões para novos aplicativos de mensagens instantâneas, tornando o sistema mais flexível e escalável.

2.2. Aplicação Assistida

O componente **Interface de Aplicativo Assistido** é responsável pela comunicação com a aplicação científica que se deseja monitorar, como a MalSynGen. Para que qualquer aplicação possa ser integrada e assistida pelo Jarvas, desenvolvemos uma biblioteca de programação. Com essa biblioteca, o usuário pode conectar sua aplicação diretamente ao fluxo de trabalho do Jarvas, permitindo a comunicação com os outros componentes do sistema de forma transparente. Na prática, a biblioteca define uma série de métodos que encapsulam as interações com a API Gateway do Jarvas, discutida na próxima seção.

2.3. API Gateway

A **API Gateway** centraliza toda a lógica do Jarvas, sendo responsável pelo gerenciamento de dados, controle dos processos em execução e orquestração das interações entre os diversos componentes do sistema. Por exemplo, a **API Gateway** coordena o envio de atualizações em tempo real sobre o progresso das aplicações monitoradas, utilizando dados fornecidos pela biblioteca integrada à aplicação externa do usuário. A Gateway também gerencia solicitações de *status* das aplicações e gera relatórios de progresso.

A **API Web** segue o padrão RESTful, ou seja, todas as funcionalidades — como gestão de usuários, autenticação e comunicação — são oferecidas através de um conjunto padronizado de *endpoints*, conforme descrito abaixo:

- **Endpoint User**. Este *endpoint* é responsável pela gestão dos usuários cadastrados no sistema, seguindo o padrão CRUD (Create, Read, Update, Delete). Ele oferece rotas que permitem a manipulação de dados de usuários de forma organizada e estruturada.
- **Endpoint App**. Dedicado à gestão das aplicações monitoradas pelo Jarvas, este *endpoint* também adota o padrão CRUD, permitindo que os desenvolvedores criem, consultem, atualizem e excluam aplicações. Além disso, o **App** inclui rotas específicas para o envio de notificações. Como o Jarvas foi projetado para notificar os usuários em tempo real sobre o andamento das aplicações, essas rotas são essenciais para enviar mensagens instantâneas às redes sociais do usuário, como o WhatsApp, informando sobre o progresso ou conclusão de tarefas.
- **Endpoint Auth**. Este *endpoint* gerencia o processo de autenticação dos usuários no sistema, garantindo que apenas usuários autorizados tenham acesso às funcionalidades do Jarvas. Ele também oferece funcionalidades para a geração e renovação de *tokens*, que verificam a validade dos fluxos de comunicação e mantêm a segurança da interação.

- **Endpoint SocialMedia**. Esse *endpoint* possibilita a interação inversa entre o usuário e o Jarvas via aplicativos de mensagens instantâneas, uma subcategoria das redes sociais. Enquanto o foco principal do sistema é notificar o usuário sobre o progresso das aplicações, o **Endpoint SocialMedia** permite que o usuário interaja com o Jarvas por meio de mensagens recebidas via API de uma rede social.

3. Prova de Conceito

Demonstramos a viabilidade técnica da arquitetura proposta por meio de uma implementação funcional, apresentada como prova de conceito e disponibilizada publicamente[5]. Nesta seção, detalhamos as tecnologias selecionadas (Subseção 3.1), o ambiente de testes utilizado (Subseção 3.2) e os testes funcionais realizados (Subseção 3.3).

3.1. Tecnologias

O sistema foi desenvolvido em Python v3, com o uso do *Poetry* para gerenciar dependências e ambientes virtuais. Para criar as rotas da API Web, utilizamos o *framework FastAPI*, combinado com a ferramenta *Swagger* para gerar a documentação das rotas, e a biblioteca *Pydantic* para validação e conversão dos dados recebidos pela API.

Na camada de persistência, optamos por implementar um banco de dados evolutivo utilizando o conceito de *migrations*, aliado a uma combinação de tecnologias. Como o Jarvas não demanda grandes volumes de dados, escolhemos o *SQLite* como sistema de gerenciamento de banco de dados. Para gerenciar as migrações de banco de dados, usamos o *Alembic*, uma ferramenta leve que funciona em conjunto com o ORM *SQLAlchemy*.

Na implementação do sistema de segurança da API Jarvas, adotamos tecnologias amplamente reconhecidas. O protocolo *OAuth2* foi escolhido para gerenciar o processo de autenticação dos usuários. Utilizamos *JWT* (JSON Web Token) [Jones 2015] para gerenciar sessões, permitindo a geração de tokens que garantem acesso às rotas protegidas da aplicação. Para assegurar o armazenamento seguro das senhas dos usuários, utilizamos a função ide derivação de chaves *argon2*[6], que foi projetada para ser eficiente e resistir a ataques como *side-channel*[7].

3.2. Ambiente de Testes

O ambiente de testes foi implementado utilizando diversas tecnologias de teste automatizado em Python. Adotamos o *Pytest* como *framework* principal para realizar os testes automatizados, uma escolha que facilita a escrita de testes de maneira intuitiva e eficiente, promovendo um desenvolvimento ágil. Para simplificar a execução de comandos repetitivos dentro da aplicação Jarvas, utilizamos a *TaskAPI*, que permite reduzir comandos extensos a uma ou duas palavras, facilitando a automação de tarefas complexas, como a execução de testes e configurações.

[5] `https://github.com/RafaelNogueiraXD/jarvas`
[6] `https://www.npmjs.com/package/argon2`
[7] `https://en.wikipedia.org/wiki/Argon2`

Para gerar dados falsos e acelerar o processo de teste em diferentes cenários, empregamos a biblioteca *Factory*. Esses dados simulados permitem testar a aplicação em situações realistas sem a necessidade de utilizar informações reais dos usuários, garantindo mais flexibilidade e segurança no ambiente de testes.

3.3. Testes Funcionais

Para garantir que todas as rotas da API Jarvas funcionassem corretamente, desenvolvemos testes automatizados para cada uma delas. Esses testes verificaram, por exemplo, se um administrador podia alterar os dados de um usuário comum e se um usuário comum conseguia modificar informações de outro usuário sem a devida permissão.

Adicionalmente, implementamos testes de segurança para validar o comportamento do sistema em tentativas de geração de tokens com credenciais inválidas. O objetivo era assegurar que o sistema de autenticação fosse seguro, permitindo que apenas usuários autorizados realizassem operações sensíveis na aplicação.

A Figura 2 apresenta o resultado dos testes realizados. Como mostrado, nossa implementação passou em todos os testes, confirmando a viabilidade técnica da arquitetura proposta. As definições detalhadas dos testes estão disponíveis no repositório público do Jarvas [5].

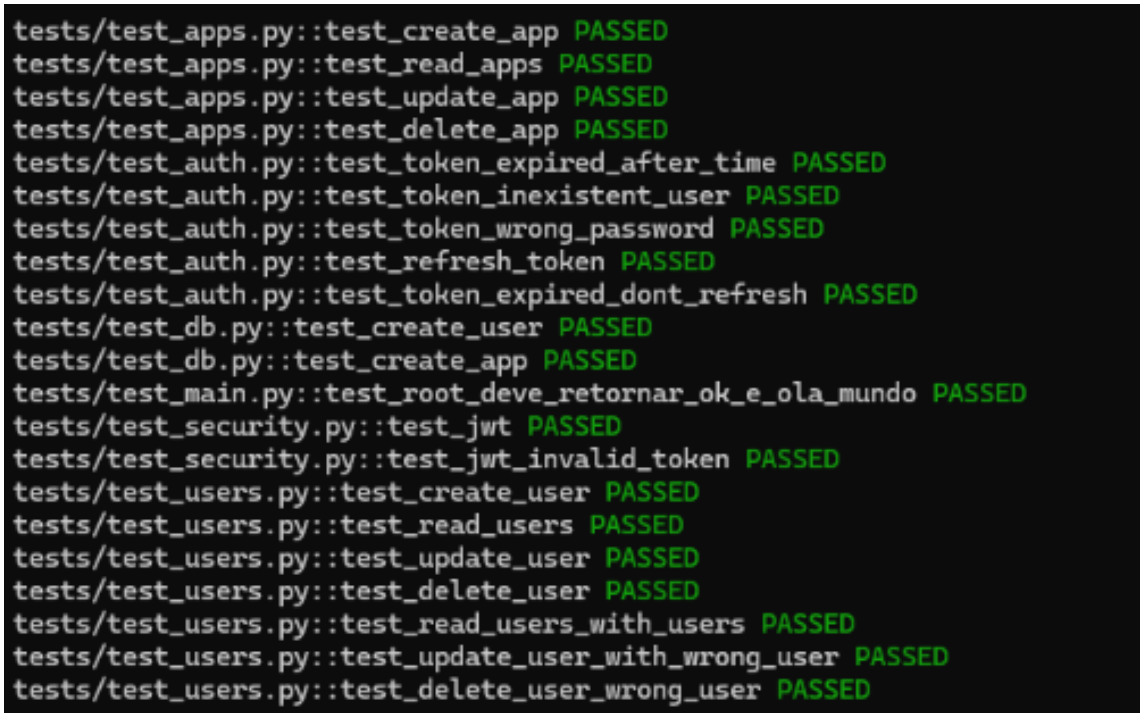

Figure 2. Todos os testes do sistema Jarvas

4. Estudo de Caso

Para atestar a *utilidade* do Jarvas, analisamos sua aplicação no projeto Malware DataLab. Esse projeto visa proporcionar um ambiente de ensino e aprendizagem de

Redes Neurais Artificiais (RNAs) aplicadas à geração de dados sintéticos sobre malwares Android. Esses dados sintéticos são valiosos para superar limitações impostas por métodos tradicionais de obtenção de dados rotulados sobre malwares Android (por exemplo, o uso de ferramentas como VirusTotal para rotulação). Esses dados são essenciais para treinar ferramentas de inteligência artificial voltadas à detecção de malwares Android.

Para verificar a flexibilidade da Jarvas, encapsulamos sua biblioteca de programação em uma nova biblioteca, com funcionalidades específicas para o projeto Malware DataLab. Ela foi projetada para facilitar a integração direta e simplificada ao fluxo de execução de ferramentas como o MalSynGen [Nogueira et al. 2024b], um gerador de dados sintéticos sobre malwares Android.

As funcionalidades específicas dessa biblioteca incluem a definição do início e fim do monitoramento, a marcação do início de uma nova fase (dobra), e a comunicação da conclusão de etapas. Além disso, incorporamos a opção de monitoramento do sistema, permitindo a coleta de informações sobre o uso de CPU, memória e disco durante a execução dos experimentos.

A Figura 3 apresenta um exemplo do uso do Jarvas no contexto do Malware DataLab. Essa implementação também está disponível no repositório público[8].

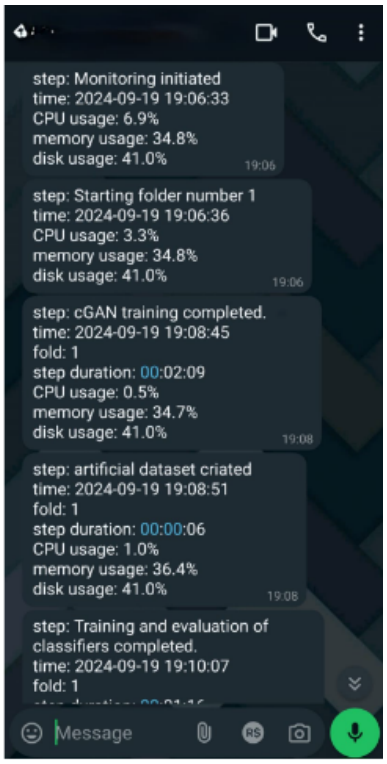

Figure 3. Funcionamento do Jarvas acoplando ao MalSynGen do projeto Malware DataLab.

[8]`https://github.com/SBSegSF24/MalSynGen`

5. Considerações Finais

Neste trabalho, apresentamos a Jarvas, um chatbot desenvolvido para auxiliar em processos científicos. A proposta é utilizar robôs de conversação como interface de comunicação, integrando-se aos aplicativos de mensagens instantâneas mais populares. Dessa forma, esperamos melhorar a interação entre equipes de desenvolvimento, especialmente aquelas envolvidas com ciência de dados. Como prova de conceito, apresentamos uma implementação pública, testada funcionalmente por meio de métodos automatizados. Além disso, um estudo de caso demonstra a utilidade da Jarvas no contexto de ensino de técnicas de ciência de dados aplicadas à cibersegurança.

Para trabalhos futuros, identificamos diversas oportunidades de melhoria. Planejamos adicionar novas funcionalidades, além de compatibilidade com outros aplicativos de mensagens instantâneas e redes sociais. Também pretendemos realizar testes para avaliar requisitos não funcionais, como aspectos de sistemas distribuídos e segurança. Outro objetivo é conduzir um estudo com usuários reais no âmbito do programa Hackers do Bem da RNP. Por fim, queremos explorar o impacto do uso de chatbots integrados a equipes de desenvolvimento, tanto profissionais quanto em formação, voltadas à ciência de dados, em contextos além da cibersegurança.

Agradecimentos. A pesquisa contou com apoio parcial da RNP (Programa Hackers do Bem - GT Malware DataLab), da CAPES (Código de Financiamento 001) e da FAPERGS, por meio dos editais 02/2022 (processo 22/2551-0000841-0), 08/2023 e 09/2023 e do termo de outorga 24/2551-0001368-7.

References

Adamopoulou, E. and Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006.

Colace, F., De Santo, M., Lombardi, M., Pascale, F., Pietrosanto, A., Lemma, S., et al. (2018). Chatbot for e-learning: A case of study. *International Journal of Mechanical Engineering and Robotics Research*, 7(5):528–533.

Jones, M. (2015). JSON web token (JWT). *Internet Engineering Task Force (IETF) RFC*, 7519.

Luo, B., Lau, R. Y., Li, C., and Si, Y.-W. (2022). A critical review of state-of-the-art chatbot designs and applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(1):e1434.

Nogueira, A., Paim, K., Bragança, H., Mansilha, R., and Kreutz, D. (2024a). Geração de dados sintéticos tabulares para detecção de malware android: um estudo de caso. In *Anais do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 808–814, Porto Alegre, RS, Brasil. SBC.

Nogueira, A., Paim, K., Bragança, H., Mansilha, R., and Kreutz, D. (2024b). Malsyngen: redes neurais artificiais na geração de dados tabulares sintéticos para detecção de malware. In *Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 129–136, Porto Alegre, RS, Brasil. SBC.

Rodrigues, R. N. (2023). Scriptpulse: Sistema de notificação para execução de experimentos computacionais. In *Salão Internacional de Ensino, Pesquisa e Extensão*, pages 129–136, Porto Alegre, RS, Brasil. SBC.