v7.n11.2026

REVISTA CIENTÍFICA - RECIMA21 ISSN 2675-6218

# AUTODEV: ORQUESTRAÇÃO AUTÔNOMA E PERSISTENTE PARA DESENVOLVIMENTO DE *SOFTWARE*

## *AUTODEV: AUTONOMOUS AND PERSISTENT ORCHESTRATION FOR SOFTWARE DEVELOPMENT*

## *AUTODEV: ORQUESTACIÓN AUTÓNOMA Y PERSISTENTE PARA EL DESARROLLO DE SOFTWARE*

Danillo Gabriel Vieira de Matos[1], Diego César de Oliveira Chaves[1], Eduardo Pateis Joaquim[1], Guilherme Félix Francisco[1], Lucas Alberto dos Santos Carvalho[1], Nathan Justo da Costa Lindo[1], Victor Stefano Garcia[1], Walter da Silva Menezes[1], Nelson Augusto Oliveira de Aguiar[2]

e7119507
https://doi.org/10.47820/recima21.v7i11.9507

PUBLICADO: 11/2026

**RESUMO**

Ferramentas de agentes de inteligência artificial (IA) para desenvolvimento de *software* são contratadas por assinatura, com cotas renovadas em janelas de tempo: a cota não utilizada expira e é renovada no início das janelas seguintes, e a execução é interrompida quando o limite é atingido. Este trabalho apresenta o projeto, a arquitetura e uma avaliação descritiva e exploratória do AutoDev, extensão gratuita para o Visual Studio Code que orquestra os agentes de linha de comando já contratados pelo usuário, com encadeamento de provedores, espera pela renovação da cota, pontos de restauração (*checkpoints*) no Git e limites de contexto. A pesquisa, aplicada e predominantemente quantitativa e descritiva, seguiu a *Design Science Research*, com desenvolvimento em *Scrum*. Foram analisadas 27 execuções em oito projetos, entre 26/08 e 27/09/2026, com 142,5 horas de operação e 213 *commits*, além de um estudo de caso de cerca de 108 horas, tratado como evidência separada. Quanto à hipótese de conversão de cota (H1), os registros não permitem medir a fração da cota utilizada por provedor e por janela. A continuidade diante de limites de cota (H2a) foi consistente com a hipótese: 31 de 35 episódios foram superados sem intervenção humana (88,6%), valor exploratório, e não medida geral de confiabilidade. A continuidade diante de interrupções do ambiente (H2b) não foi sustentada: uma reinicialização do Windows não foi recuperada. Os resultados indicam viabilidade parcial da continuidade operacional, sem demonstrar ganho de produtividade ou de qualidade do código. Os dados e *scripts* de análise estão disponíveis no Zenodo.

**PALAVRAS-CHAVE:** Agentes de Inteligência Artificial. Orquestração Persistente. Aproveitamento de Cota Ociosa. Tolerância a Falhas. Engenharia de Software. VS Code.

***ABSTRACT***

*Artificial intelligence (AI) coding agent tools are sold by subscription, with quotas renewed in time windows: unused quotas expire and are renewed at the beginning of subsequent usage windows, and execution stops when the limit is reached. This paper presents the design, architecture, and a descriptive and exploratory evaluation of AutoDev, a free Visual Studio Code extension that orchestrates the command-line agents the user already subscribes to, with provider chaining, waiting for quota renewal, Git checkpoints, and context limits. The research is applied and predominantly quantitative and descriptive; it followed Design Science Research, with development under Scrum.*

[1] Graduando em Engenharia da Computação pela Universidade São Judas Tadeu (USJT).
[2] Doutor em Ciências, com aplicação de Inteligência Artificial, pela Universidade de São Paulo (USP), docente desde 2006, professor orientador na USJT.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

*Twenty-seven runs across eight projects, between 26 August and 27 September 2026, were analyzed, totaling 142.5 hours of operation and 213 commits, plus a case study of about 108 hours, treated as separate evidence. As for the quota conversion hypothesis (H1), the logs do not allow measuring the share of quota used per provider and per window. Continuity under quota limits (H2a) was consistent with the hypothesis: 31 of 35 episodes were overcome without human intervention (88.6%), an exploratory figure rather than a general measure of reliability. Continuity under environment interruptions (H2b) was not supported: a Windows restart was not recovered. The results indicate partial feasibility of operational continuity, without demonstrating gains in productivity or code quality. The data and analysis scripts are available on Zenodo.*

***KEYWORDS:*** *Artificial Intelligence Agents. Persistent Orchestration. Idle Quota Use. Fault Tolerance. Software Engineering. VS Code.*

***RESUMEN***
*Las herramientas de agentes de inteligencia artificial (IA) para el desarrollo de software se contratan por suscripción, con cuotas renovadas en ventanas de tiempo: la cuota no utilizada expira y se renueva al inicio de las ventanas siguientes, y la ejecución se detiene cuando se alcanza el límite. Se presentan el diseño, la arquitectura y una evaluación descriptiva y exploratoria de AutoDev, extensión gratuita para Visual Studio Code que orquesta los agentes de línea de comandos ya contratados por el usuario, con encadenamiento de proveedores, espera por la renovación de la cuota, puntos de control en Git y límites de contexto. La investigación, aplicada y predominantemente cuantitativa y descriptiva, siguió la Design Science Research, con desarrollo en Scrum. Se analizaron 27 ejecuciones en ocho proyectos, entre el 26/08 y el 27/09/2026, con 142,5 horas de operación y 213 commits, además de un estudio de caso de unas 108 horas, tratado como evidencia separada. Sobre la hipótesis de conversión de cuota (H1), los registros no permiten medir la fracción de la cuota usada por proveedor y por ventana. La continuidad ante límites de cuota (H2a) fue consistente con la hipótesis: 31 de 35 episodios se superaron sin intervención humana (88,6%), valor exploratorio y no medida general de confiabilidad. La continuidad ante interrupciones del entorno (H2b) no se sostuvo: un reinicio de Windows no se recuperó. Los resultados indican viabilidad parcial de la continuidad operacional, sin demostrar ganancias de productividad o de calidad del código. Los datos y scripts están disponibles en Zenodo.*

***PALABRAS CLAVE:*** *Agentes de Inteligencia Artificial. Orquestación Persistente. Aprovechamiento de Cuota Inactiva. Tolerancia a Fallos. Ingeniería de Software. VS Code.*

## INTRODUÇÃO

O avanço das ferramentas de assistência à codificação baseadas em inteligência artificial (IA, do inglês *Artificial Intelligence*) alterou a rotina de desenvolvedores. Serviços como Claude Code (Anthropic), Codex (OpenAI) e Antigravity (Google) deixaram de ser apenas recursos de autocompletar texto e passaram a atuar como agentes que leem o código do projeto, editam arquivos e executam comandos no terminal (ANTHROPIC, 2026a; OPENAI, 2026a; GOOGLE, 2026). Na edição de 2025 da pesquisa anual do Stack Overflow, 84% dos respondentes afirmaram usar ou planejar usar ferramentas de IA em seu processo de desenvolvimento (STACK OVERFLOW, 2025).



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

O uso cotidiano dessas ferramentas, entretanto, envolve duas limitações práticas. A primeira decorre do modelo de assinatura, cujos planos oferecem uma cota de uso medida em janelas de tempo. No Claude Code, o limite é renovado a cada cinco horas e há limites semanais (ANTHROPIC, 2026c); no Codex, a documentação estima a quantidade de mensagens locais por período de cinco horas nos planos Plus e Business e informa que limites semanais também podem se aplicar (OPENAI, 2026b). Como a cota é medida por janela, e não por tempo de uso, ela pode ser esgotada em poucos minutos de trabalho intenso; por outro lado, a cota não utilizada nas janelas em que ninguém aciona a ferramenta, como noites e fins de semana, expira e é renovada no início da janela seguinte. O aproveitamento da capacidade contratada fica, assim, limitado pela disponibilidade humana para acionar o agente.

A segunda limitação manifesta-se em tarefas mais longas: ao atingir o limite de requisições do provedor, a ferramenta interrompe a execução e emite um erro de limite de taxa (*rate limit*). O trabalho permanece paralisado até que uma pessoa perceba a parada, aguarde a renovação ou configure manualmente outro serviço. Somam-se a esse obstáculo os encerramentos abruptos de terminal, as falhas de conexão e o crescimento do contexto acumulado (*token bloat*), no qual o histórico da sessão é reenviado a cada turno e eleva o consumo de *tokens* (SALIM *et al.*, 2026).

A justificativa deste trabalho reside no fato de que a necessidade de monitoramento contínuo dos agentes constitui uma limitação operacional à execução autônoma de tarefas de desenvolvimento. Uma camada que permita definir metas ao fim do expediente e manter a execução durante a noite, alternando entre provedores quando uma cota se esgota e registrando cada entrega no Git, apresenta potencial para ampliar o período de execução automatizada, sem necessariamente exigir novas assinaturas.

Diante desse problema, o objetivo geral deste trabalho consiste em desenvolver, modelar e avaliar empiricamente o AutoDev, extensão para o Visual Studio Code (VS Code) que opera como camada de orquestração persistente sobre agentes de IA de desenvolvimento de *software*, permitindo delegar objetivos em linguagem natural, utilizar cotas fora do horário de trabalho, contornar limites de uso por meio da alternância automática entre provedores e registrar pontos de restauração no Git sem supervisão contínua. Para atingir esse objetivo, definiram-se os seguintes objetivos específicos:

a) modelar uma arquitetura em quatro camadas, que isole a apresentação, o núcleo de orquestração, os provedores de IA e a infraestrutura de persistência;

b) desenvolver uma extensão para o VS Code que permita configurar equipes de agentes sem custos adicionais de licenciamento;



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

c) implementar um mecanismo de recuperação automática (*failover*) no qual, ao se esgotar a cota de um provedor, outro provedor assuma a tarefa ou o sistema aguarde a renovação;

d) estruturar um mecanismo de salvamento e reversão via Git (/restore) que busque preservar a consistência entre o código versionado e a lista de tarefas;

e) implementar limites de memória e de *tokens* para conter o crescimento do histórico em execuções prolongadas;

f) avaliar o comportamento do sistema em condições reais de uso, por meio dos registros de execuções em projetos distintos.

A questão central desta pesquisa é: em que medida uma camada de orquestração sobre agentes de IA contratados por assinatura consegue manter a execução de tarefas de desenvolvimento sem intervenção humana diante de limites de cota e de interrupções do ambiente? O trabalho apresenta duas contribuições distintas: a primeira, de natureza arquitetural, é o projeto e a implementação do artefato; a segunda, de natureza empírica, é uma avaliação descritiva e exploratória de sua continuidade operacional, sem grupo de comparação e sem mensuração de produtividade ou de qualidade do código produzido.

## 1. REFERENCIAL TEÓRICO

### 1.1. Agentes autônomos de inteligência artificial no desenvolvimento de *software*

A incorporação de modelos generativos ao ciclo de vida do *software* passou por fases distintas. Inicialmente restritos ao autocompletar de código, esses sistemas passaram a operar em ciclos de agente capazes de editar arquivos, compilar, executar testes e realizar operações no Git. Um exemplo é o AutoDev: *Automated AI-Driven Development* (TUFANO *et al.*, 2024), *framework* em que agentes planejam e executam tarefas de engenharia de *software* dentro de contêineres Docker, com permissões definidas pelo usuário. Apesar do mesmo nome, trata-se de outra ferramenta, com outro objetivo: o AutoDev apresentado neste trabalho não executa agentes próprios, mas orquestra as ferramentas de linha de comando já contratadas pelo usuário, como o Claude Code e o Codex (ANTHROPIC, 2026a; OPENAI, 2026a), com foco no aproveitamento de cota e na continuidade da execução.

A autonomia desses agentes, contudo, é limitada. Ao analisar execuções de sistemas multiagentes baseados em modelos de linguagem, Cemri *et al.* (2025) identificaram 14 modos de falha, entre eles a repetição de etapas, o desconhecimento das condições de parada e o desvio da tarefa. Esses modos de falha indicam que a conclusão declarada por um agente não



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

equivale à verificação de que a tarefa foi cumprida: a qualidade do código produzido depende de mecanismos externos de avaliação, como testes automatizados, revisão por outro agente ou por pessoas e análise estática, cuja cobertura determina o que é efetivamente verificado. Há, ainda, o risco associado à execução de comandos sem supervisão, pois um agente com acesso ao terminal pode alterar ou excluir arquivos, instalar dependências e modificar o histórico do repositório; por esse motivo, Tufano *et al.* (2024) executam os agentes em contêineres isolados e com permissões explícitas. Por fim, os serviços contratados impõem restrições operacionais que não dependem do usuário, como os limites por janela de tempo e os limites semanais (ANTHROPIC, 2026c; OPENAI, 2026b).

### 1.2. Economia de cota e ociosidade em modelos de assinatura

Os modelos comerciais das interfaces de programação e das ferramentas de Interface de Linha de Comando (CLI, do inglês *Command-Line Interface*) de IA estruturam-se em torno da contabilização de *tokens*, unidades de texto processadas, cuja quantificação Salim *et al.* (2026) denominam *tokenomics*. Enquanto o acesso por Interface de Programação de Aplicações (API, do inglês *Application Programming Interface*) cobra por *token* de entrada e de saída, as ferramentas voltadas ao desenvolvedor final empregam assinaturas com capacidade vinculada a janelas de tempo, como limites renovados a cada cinco horas ou semanalmente.

Essa tarifação por janela cria um compromisso: a cota não utilizada em uma janela expira na renovação, e uma rotina automatizada que esgota o limite fica parada até a janela seguinte. Na literatura, a combinação de vários provedores aparece sobretudo como estratégia de redução de custo: o FrugalGPT, por exemplo, aprende em que sequência consultar diferentes APIs de modelos de linguagem para cada entrada, reduzindo o custo de inferência sem perda de desempenho (CHEN; ZAHARIA; ZOU, 2023). Esses trabalhos, contudo, tratam do custo por chamada de API, e não da continuidade da execução diante de limites de cota de assinatura, problema abordado pelo AutoDev.

### 1.3. Tolerância a falhas e recuperação por *checkpoint*

Candea e Fox (2003) introduziram o conceito de *crash-only software*, segundo o qual sistemas resilientes devem tratar quedas inesperadas, congelamentos de processo e reinicializações como eventos operacionais normais, e não como exceções. Para isso, o estado do sistema deve ser persistido externamente, de modo que a recuperação dispense procedimentos complexos de encerramento (*shutdown*). A recuperação por *checkpoint*, em que o estado é gravado periodicamente em armazenamento estável e o processo é retomado a partir



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

do último ponto gravado após uma falha, é estudada há décadas em sistemas distribuídos (ELNOZAHY *et al.*, 2002).

Em sistemas multiagentes para *software*, essa abordagem requer que o orquestrador separe a geração de código pelos modelos do controle do ciclo de vida das tarefas: se um subprocesso de agente falhar, travar ou for encerrado pelo sistema operacional, o núcleo deve reconstituir o estado a partir do disco e retomar o trabalho do último *checkpoint*. A persistência do estado é, porém, condição necessária, mas não suficiente: a retomada também depende de que o próprio orquestrador volte a ser executado após uma falha do ambiente.

### 1.4. Economia de contexto e degradação da janela de atenção

Os Modelos de Linguagem em Grande Escala (LLM, do inglês *Large Language Models*) possuem limites quanto à quantidade de *tokens* admitidos na janela de contexto (*context window*). Embora modelos contemporâneos suportem janelas extensas, o desempenho na recuperação de informações cai quando a informação relevante está no meio de contextos longos (LIU *et al.*, 2024). Além disso, como o histórico acumulado é reenviado a cada turno, o custo por chamada cresce com o tamanho da sessão: em 30 tarefas executadas pelo *framework* ChatDev com um modelo GPT-5, os *tokens* de entrada responderam, em média, por 53,9% do consumo (SALIM *et al.*, 2026). Em automações contínuas, uma sessão mantida aberta indefinidamente acumula registros (*logs*) de testes, código obsoleto e mensagens de depuração; assim, o orquestrador deve limitar o tamanho de cada componente do contexto e renovar periodicamente a sessão, de modo a conter o custo por iteração.

### 1.5. Síntese comparativa

O *Quadro 1* sistematiza a comparação entre o AutoDev e as abordagens discutidas. A última coluna distingue o que foi efetivamente observado na avaliação deste trabalho daquilo que foi apenas implementado.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

**Quadro 1.** Comparação entre abordagens relacionadas e o AutoDev

| Característica | AutoDev de Tufano *et al.* (2024) | FrugalGPT (Chen; Zaharia; Zou, 2023) | CLIs de agentes individuais | AutoDev (este trabalho) e evidência obtida |
|---|---|---|---|---|
| Objetivo principal | Execução autônoma de desenvolvimento | Redução do custo de inferência | Assistência à codificação | Execução contínua e aproveitamento de cota |
| Agentes utilizados | Agentes próprios do *framework* | Chamadas a APIs de modelos | O próprio CLI | CLIs já contratados pelo usuário |
| Múltiplos provedores | Não é foco | Sim, em cascata, por custo | Não | Sim (*failover*); 11 trocas no *corpus* |
| Retomada após limite de cota | Não tratada | Não tratada (cobrança por chamada) | Manual | Parcial: 31 de 35 episódios (H2a) |
| Retomada após falha do ambiente | Não tratada | Não tratada | Manual | Não sustentada (H2b) |
| Contenção de comandos de risco | Contêineres Docker e permissões | Não se aplica | Permissões definidas pelo CLI | Portões de aprovação; falha observada (F3) |
| Qualidade do código | Testes no ciclo do agente | Não se aplica | A cargo do usuário | Revisão entre agentes; qualidade não medida |

Fonte: Elaborado pelos autores (2026), baseado em Tufano *et al.* (2024) e em Chen, Zaharia e Zou (2023).

## 2. METODOLOGIA

### 2.1. Caracterização e método de pesquisa

A pesquisa possui natureza aplicada, pois se orienta à solução de um problema prático da engenharia de *software*: a cota contratada de IA que não é utilizada fora do horário de trabalho e a paralisação de execuções autônomas por limites de uso (SILVA; MENEZES, 2005). Quanto à abordagem, classifica-se como predominantemente quantitativa e descritiva, associada ao desenvolvimento de um artefato tecnológico: as hipóteses são avaliadas por indicadores calculados a partir dos registros de execução, sem procedimentos sistemáticos de análise qualitativa. A especificação de requisitos e a modelagem arquitetural em Linguagem de Modelagem Unificada (UML, do inglês *Unified Modeling Language*) integram o desenvolvimento do artefato, e não a análise dos dados.

Como método, adotou-se a Pesquisa em Ciência do Design (DSR, do inglês *Design Science Research*), voltada ao desenvolvimento e à avaliação de artefatos projetados para intervir em problemas reais (DRESCH; LACERDA; ANTUNES JÚNIOR, 2015). O processo seguiu as seis etapas propostas por Peffers *et al.* (2007):



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

a) identificação do problema e motivação: a cota não utilizada fora do horário de trabalho e as paradas por limite de uso;

b) definição dos objetivos da solução: persistência de estado, alternância de provedores, versionamento no Git e contenção do contexto;

c) projeto e desenvolvimento: arquitetura em quatro camadas, implementada em TypeScript sobre Node.js e integrada ao VS Code;

d) demonstração: execução em projetos de teste de complexidade crescente;

e) avaliação: análise dos registros de execuções reais, com indicadores de *commits*, uso de cota, recuperação de falhas e distribuição horária;

f) comunicação: divulgação dos resultados na forma de artigo científico.

O desenvolvimento foi organizado com o *framework* ágil *Scrum*, em cinco *sprints* ao longo do semestre letivo de 2026. O *backlog* do produto foi decomposto em histórias de usuário derivadas dos casos de uso modelados, e o acompanhamento das tarefas e das retrospectivas foi feito na plataforma Jira.

### 2.2. Hipóteses de pesquisa

O estudo é descritivo e exploratório. As hipóteses foram formuladas de modo que os próprios registros de execução possam contrariá-las, e cada uma é acompanhada do resultado que a contrariaria. Os resultados são relatados como consistentes ou não consistentes com as hipóteses, sem pretensão de confirmação estatística:

a) **H1 (conversão de cota disponível):** em execução autônoma, o AutoDev mantém os agentes em atividade até o limite de uso imposto pelo provedor e retoma o trabalho após a renovação, independentemente do horário. Indicadores: proporção de execuções que atingiram o limite; utilização semanal da cota informada pelo CLI; retomada após a renovação; *tokens* consumidos. A H1 seria contrariada se as execuções terminassem, em sua maioria, sem atingir os limites, ou se a utilização da cota permanecesse distante do teto;

b) **H2a (continuidade diante de limites de cota):** quando um provedor atinge o limite de uso, o AutoDev retoma o trabalho sem intervenção humana, por troca de provedor ou por espera programada. Indicador: proporção de episódios de cota esgotada resolvidos sem intervenção. A H2a seria contrariada se a maior parte dos episódios exigisse intervenção;

c) **H2b (continuidade diante de interrupções do ambiente):** após queda de processo, suspensão ou reinicialização do computador, o AutoDev retoma a execução a partir



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

do último estado persistido, sem intervenção humana. Indicador: desfecho de cada interrupção registrada. A H2b seria contrariada por qualquer interrupção após a qual a execução não fosse retomada sem intervenção.

### 2.3. Coleta e tratamento dos dados

A avaliação utiliza duas fontes de evidência, analisadas separadamente. A primeira é o *corpus* multiprojeto, formado pelos registros de execução de uma das contas da equipe entre 26 de agosto e 27 de setembro de 2026 (seção 3.4). A segunda é o estudo de caso de uma execução contínua de cerca de 108 horas, realizada em outra conta, em outro computador e com versões posteriores da extensão (seção 3.5). Em razão dessas diferenças, os resultados do estudo de caso não são somados aos do *corpus* nem tratados como diretamente equivalentes a eles.

**Registro de cota.** O subsistema *QuotaLedger* grava um registro por evento, em formato JSON por linha (*quota.jsonl*), de três tipos: (a) chamada a um agente, com horário de início, duração, provedor, êxito e, quando o CLI informa, *tokens* consumidos e utilização da cota; (b) limite, com o provedor bloqueado e o horário de liberação informado; e (c) trabalho, com o identificador (*hash*) do *commit* gerado. Apenas o CLI do Claude Code informa a utilização da cota, como percentual da janela semanal da conta inteira, incluindo o uso manual do titular; os CLIs do Codex e do Antigravity informam somente o bloqueio. Por isso, a utilização da cota é medida apenas para o Claude Code; para os demais provedores, contam-se os bloqueios.

**Registro de eventos e perda parcial.** O AutoDev também grava um registro detalhado de eventos (*events.jsonl*), mas descarta o histórico quando o arquivo ultrapassa 10 MB. Por isso, os eventos de 26 de agosto a cerca de 15 de setembro se perderam. Essa perda tem três consequências: os desfechos das paradas desse intervalo foram reconstruídos apenas pelo registro de cota; a regra de classificação dos episódios só pôde ser conferida no intervalo de 16 a 27 de setembro, no qual todas as paradas por limite do registro de eventos constam também do registro de cota; e as interrupções do ambiente no *corpus* não puderam ser identificadas, razão pela qual a H2b é avaliada apenas no estudo de caso.

**Protocolo.** Em cada execução, a equipe configurou o repositório no VS Code com os CLIs já autenticados, definiu as metas e a cadeia de provedores e acionou o modo autônomo. As execuções foram iniciadas, em geral, ao fim do expediente: 24 das 27 começaram fora do horário comercial (segunda a sexta-feira, das 8h às 18h, horário de Brasília), e 90,4% do tempo de operação ocorreu fora desse horário.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

**Critérios de inclusão e exclusão.** Uma execução é uma sequência de registros do mesmo projeto em que o intervalo entre registros consecutivos não excede 6 horas. Foram incluídas as execuções com duração mínima de 15 minutos e ao menos cinco chamadas a agentes; as demais foram excluídas por corresponderem a testes rápidos de funcionamento da extensão. Das 36 execuções identificadas, 9 foram excluídas e 27 compõem o *corpus*, em oito projetos. A contagem de episódios de cota (seção 3.4) utiliza todos os 63 registros de limite do período, sem exclusão de execuções. Horas ativas são a soma das durações das chamadas aos agentes; horas de operação, a soma das durações das execuções.

**Projetos.** Os oito projetos são: o repositório do próprio AutoDev (*autodev*), em TypeScript, com cerca de 680 arquivos e 62 mil linhas, e duas *worktrees* dele, tratadas como projetos distintos (*nightly-2026-08-28*, de 28/08 a 17/09, e *nightly-2026-09-27*); o site e a documentação do AutoDev (*autodev_client*), em TypeScript, React e MDX, com cerca de 260 arquivos; um *bot* do Discord que atua como assistente de voz com a API GPT-Live da OpenAI (*discord-live-mvp*), com cerca de 130 arquivos e 13 mil linhas; um jogo cooperativo em primeira pessoa na Unreal Engine 5.8 (*nosuchplace*), com cerca de 23 mil linhas de C++ e 560 arquivos binários de recursos (*assets*); e dois projetos de teste do painel (*autodev-sandbox-v9* e *autodev-sandbox-v10*), que não geraram *commits*. Dos 213 *commits*, 141 (66%) foram feitos no repositório do próprio AutoDev, o que reduz a diversidade efetiva das observações.

**Versões.** No período, o AutoDev foi usado em *builds* de desenvolvimento até 17/09 e na versão 0.1.0, distribuída como pacote .vsix, a partir de 18/09; a primeira versão publicada no Visual Studio Marketplace (0.2.0) é de 28/09, posterior ao *corpus*. O projeto *nightly-2026-08-28* utilizou uma versão inicial do motor de orquestração, com etapas de planejamento definidas pelo próprio AutoDev, removida em 27/09. Foram utilizados o Claude Code 2.1.246 a 2.1.283 (modelos Claude Sonnet 5, Claude Opus 5, Claude Opus 5.5, Claude Haiku 4.5 e Claude Fable 5.1), o Codex CLI 0.146 a 0.157 (modelos GPT-5.6 e GPT-6) e o Antigravity CLI 1.2.5, cujo modelo não é registrado. Os indicadores do *corpus* são apresentados de forma agregada para as duas versões; as diferenças entre versões são indicadas quando os registros permitem associá-las a datas, como nas paradas descritas na seção 3.4.

**Chamadas acionadas pelo AutoDev.** O AutoDev não usa modelo próprio nem chama APIs de IA diretamente: toda geração de texto ou código, inclusive as rodadas acionadas pelo orquestrador, como a auditoria de encerramento, é feita pelos CLIs do usuário e consome a mesma cota da assinatura. O registro de cota não as separa das demais.

**Episódios de cota esgotada.** Registros de limite do mesmo projeto e provedor com menos de 2 minutos de intervalo foram tratados como duplicatas. Um episódio começa no



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

primeiro registro de limite e termina na primeira chamada bem-sucedida seguinte no mesmo projeto; um novo limite após essa chamada abre outro episódio. O desfecho é classificado pela chamada que encerra o episódio: troca de provedor, se feita por outro provedor em menos de 1 hora; espera programada, se ocorrer até 1 hora após o horário de liberação informado; retomada manual ou tardia, se ocorrer depois disso; e sem retomada, se não houver chamada seguinte. Os dois primeiros desfechos contam como superação autônoma. Como análise de sensibilidade, relata-se também o resultado pela regra usada na versão anterior deste trabalho, que agrupava num só episódio os limites ocorridos antes do horário de liberação acrescido de 1 hora.

**Tratamento estatístico.** A amostra é intencional e não probabilística: os projetos foram escolhidos e operados pela própria equipe, em ambiente Windows, o que abre espaço para viés do experimentador. Não há linha de base, nem com uso manual dos CLIs, nem com o AutoDev sem troca de provedor; os resultados são, portanto, descritivos. Os episódios não são independentes, pois se agrupam em execuções e em projetos, e mais de um episódio pode decorrer da mesma causa. O intervalo de confiança de Wilson de 95% é apresentado apenas como referência de ordem de grandeza: por pressupor observações independentes, ele subestima a incerteza, e o intervalo real seria mais amplo. Os indicadores são gerados por um *script* em Python, que usa apenas a biblioteca padrão, a partir do registro anonimizado depositado no Zenodo (MATOS *et al.*, 2026).

## 3. RESULTADOS E DISCUSSÃO

### 3.1. O artefato AutoDev

O AutoDev é uma extensão gratuita: não comercializa créditos de IA nem cobra mensalidades, e utiliza as assinaturas que o desenvolvedor já possui, como Claude Pro (Anthropic), ChatGPT Plus (OpenAI) ou Google Antigravity. A instalação requer o Node.js, o Git e os CLIs oficiais dos provedores desejados (claude, codex ou agy), autenticados previamente pelo próprio usuário no terminal. A arquitetura foi projetada para utilizar autenticações previamente estabelecidas nos respectivos agentes, sem requerer armazenamento próprio de credenciais: o AutoDev executa o utilitário local já autenticado.

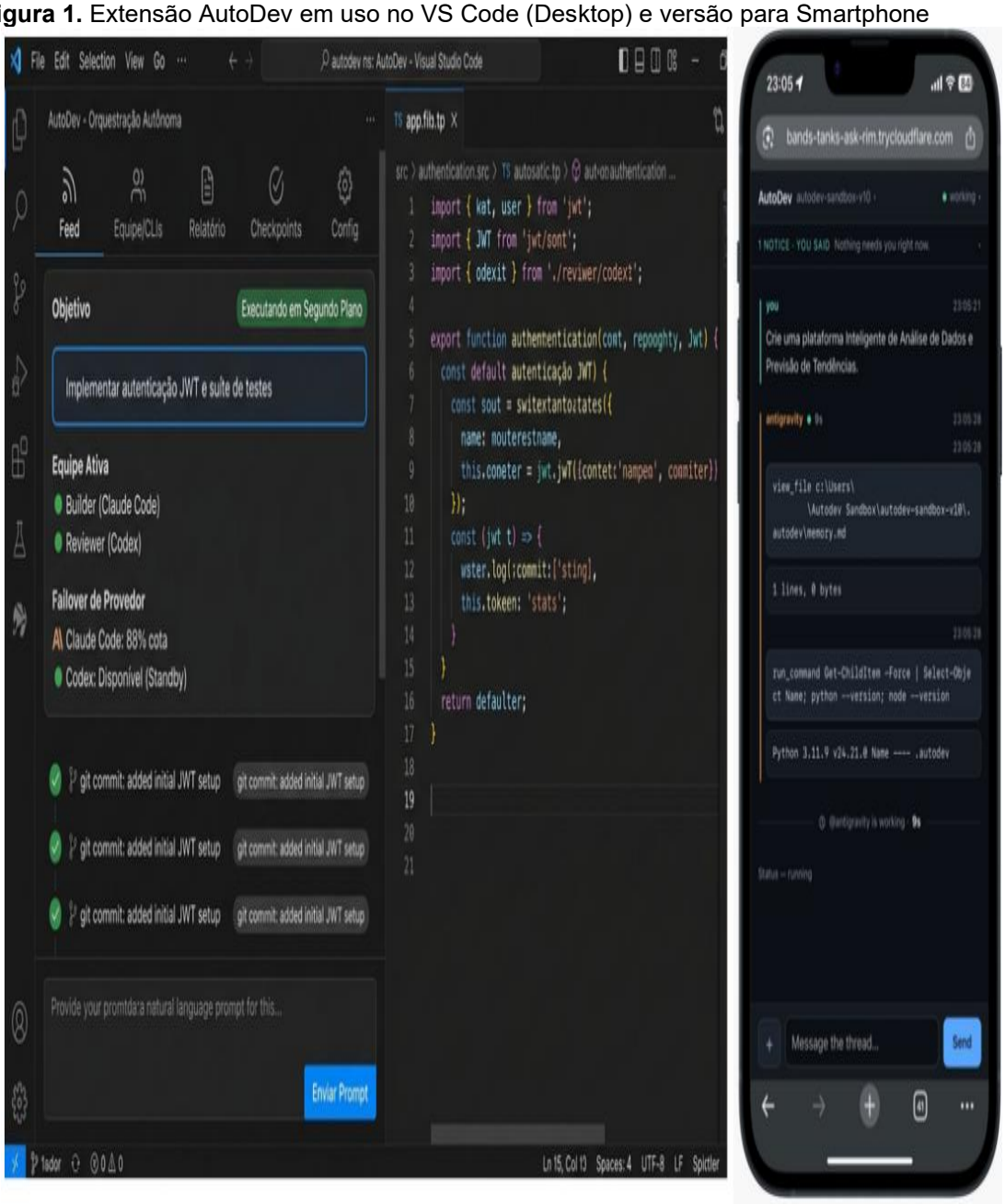
O uso ocorre no próprio VS Code. No painel da extensão, o desenvolvedor autoriza explicitamente o diretório do projeto, o que cria a pasta *.autodev/*; escolhe um modelo de equipe, entre o Modo Solo, no qual um único agente implementa e testa, o Modo Construtor e Revisor (*Builder & Reviewer*), no qual um agente programa e outro inspeciona o código antes da entrega, e o Modo Completo, com planejador, programador e validador; descreve o objetivo em linguagem



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

natural no campo de entrada (*Composer*), por exemplo, “Implementar autenticação JWT, rotas protegidas e testes unitários com Jest”; e inicia a execução, que prossegue em segundo plano, decompondo o objetivo em tarefas. A Figura 1 evidencia o uso no desktop e no smartphone.

**Figura 1.** Extensão AutoDev em uso no VS Code (Desktop) e versão para Smartphone




Fonte: Elaborado pelos autores (2026).



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

O painel reúne seis visões: linha do tempo da execução (*Timeline*); equipe e provedores, com a ordem de substituição e o estado da cota; relatório da execução (*Report*); memória e *checkpoints*, com a restauração (/restore); pedidos de decisão (*Needs You*), para aprovar ou rejeitar alterações de risco; e configurações, com limites de custo e de iterações. O acompanhamento remoto por *smartphone* utiliza um túnel da Cloudflare, com pareamento por código de seis dígitos, e permite visualizar os *commits* e responder aos pedidos de aprovação. Essas funcionalidades de interface não foram avaliadas experimentalmente neste estudo.

**3.2. Arquitetura e requisitos funcionais**

O usuário intervém principalmente na definição da meta; a partir do início da automação, o núcleo do AutoDev desdobra a meta em tarefas, submete os comandos aos agentes locais de IA, avalia o código produzido, registra pontos de restauração no Git e informa o andamento pelo acompanhamento remoto. A arquitetura organiza-se em quatro camadas desacopladas (*Figura 2*), de modo que novos provedores ou novas interfaces possam ser incluídos sem alteração do núcleo orquestrador. A Figura 2 representa as *builds* de desenvolvimento usadas até 17/09/2026; a interface de terminal interativa (TUI, do inglês *Terminal User Interface*) da camada de apresentação foi removida nessa data e não consta da versão 0.1.0. As camadas são:

a) camada de apresentação: CLI acionada pelo comando autodev e extensão do VS Code, que exibe a execução em tempo real e permite intervenção humana sob demanda;

b) camada de orquestração (AutoDev Core): *ExecutionLoop*, que controla os limites globais de tempo, custo e iterações; *Orchestrator*, que conduz o ciclo planejar–executar–validar; *PlannerAgent*, responsável pelo desdobramento do plano de trabalho (*roadmap*); e *RoleRouter*, que despacha as chamadas conforme a disponibilidade dos provedores;

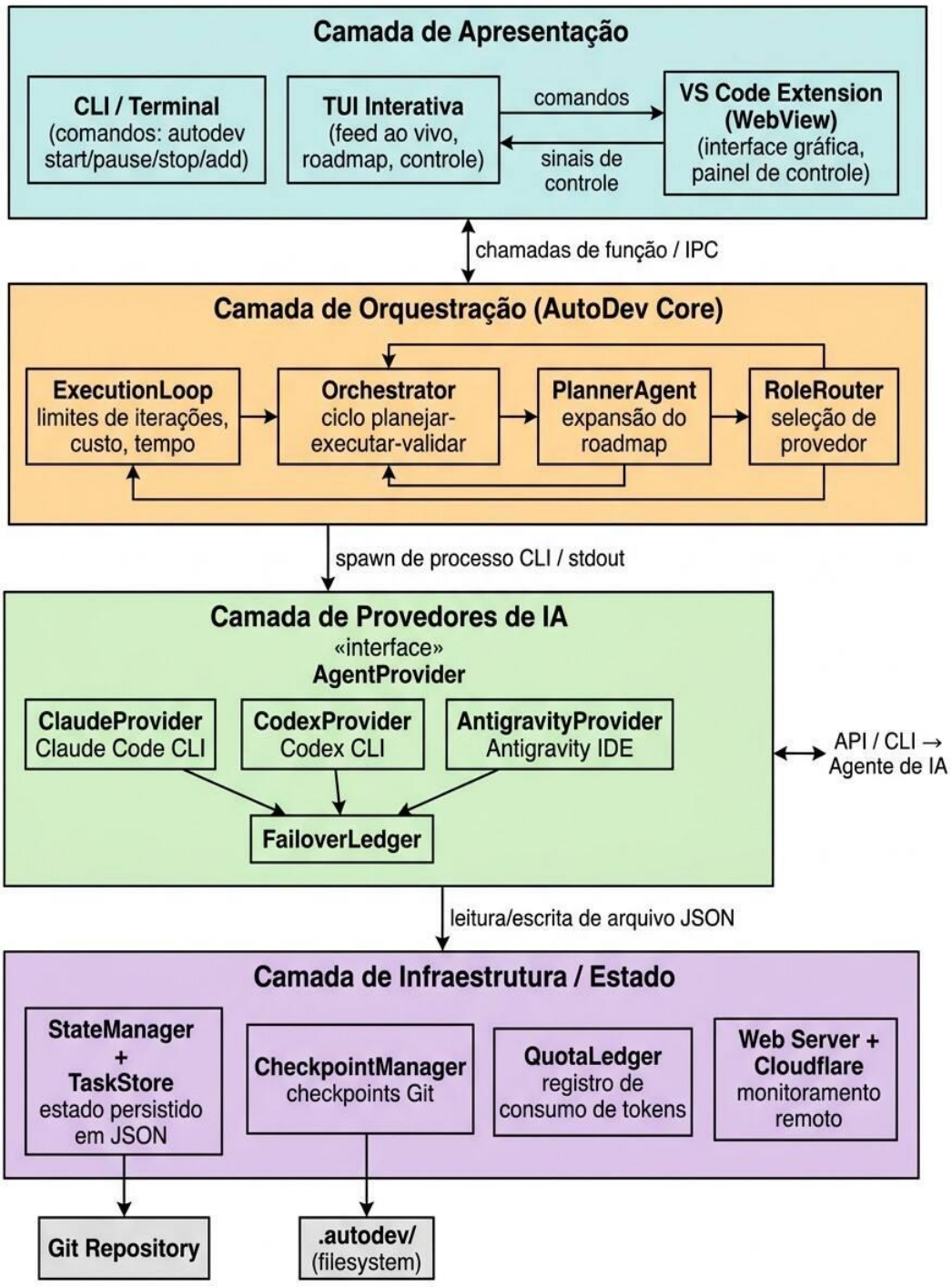
c) camada de provedores de IA: abstração sobre as ferramentas instaladas (*ClaudeProvider*, *CodexProvider* e *AntigravityProvider*), que executa cada agente como subprocesso isolado e monitora suas saídas;

d) camada de infraestrutura e persistência: *StateManager* e *TaskStore* (arquivos JSON no diretório *.autodev/*), *CheckpointManager* (*branches* e *commits* no Git), *QuotaLedger* (registro de uso) e túnel da Cloudflare para o acompanhamento remoto.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

**Figura 2.** Diagrama de camadas do AutoDev




Fonte: Elaborado pelos autores (2026).



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Os requisitos funcionais (RF) levantados na fase de especificação, com o benefício esperado e a prioridade de cada um, constam do *Quadro 2*.

**Quadro 2.** Requisitos funcionais do AutoDev

| ID | Requisito | Benefício esperado | Prioridade |
|---|---|---|---|
| RF-01 | Execução por linguagem natural | Definir metas sem *prompts* complexos | Essencial |
| RF-02 | Pausar e retomar a execução | Retomar do último estado salvo | Essencial |
| RF-03 | Alternar entre provedores de IA | Prosseguir em outro provedor | Essencial |
| RF-04 | Persistir o estado em disco | Retomar após falha do ambiente | Essencial |
| RF-05 | Registrar *checkpoints* no Git | Um *commit* auditável por tarefa | Essencial |
| RF-06 | Restaurar o repositório | Reverter o código e reabrir tarefas | Essencial |
| RF-07 | Registrar métricas de cota | Exibir uso, limites e custo | Importante |
| RF-08 | Oferecer acesso remoto | Aprovar decisões pelo *smartphone* | Desejável |
| RF-09 | Oferecer interface no VS Code | Visualizar a execução no editor | Desejável |

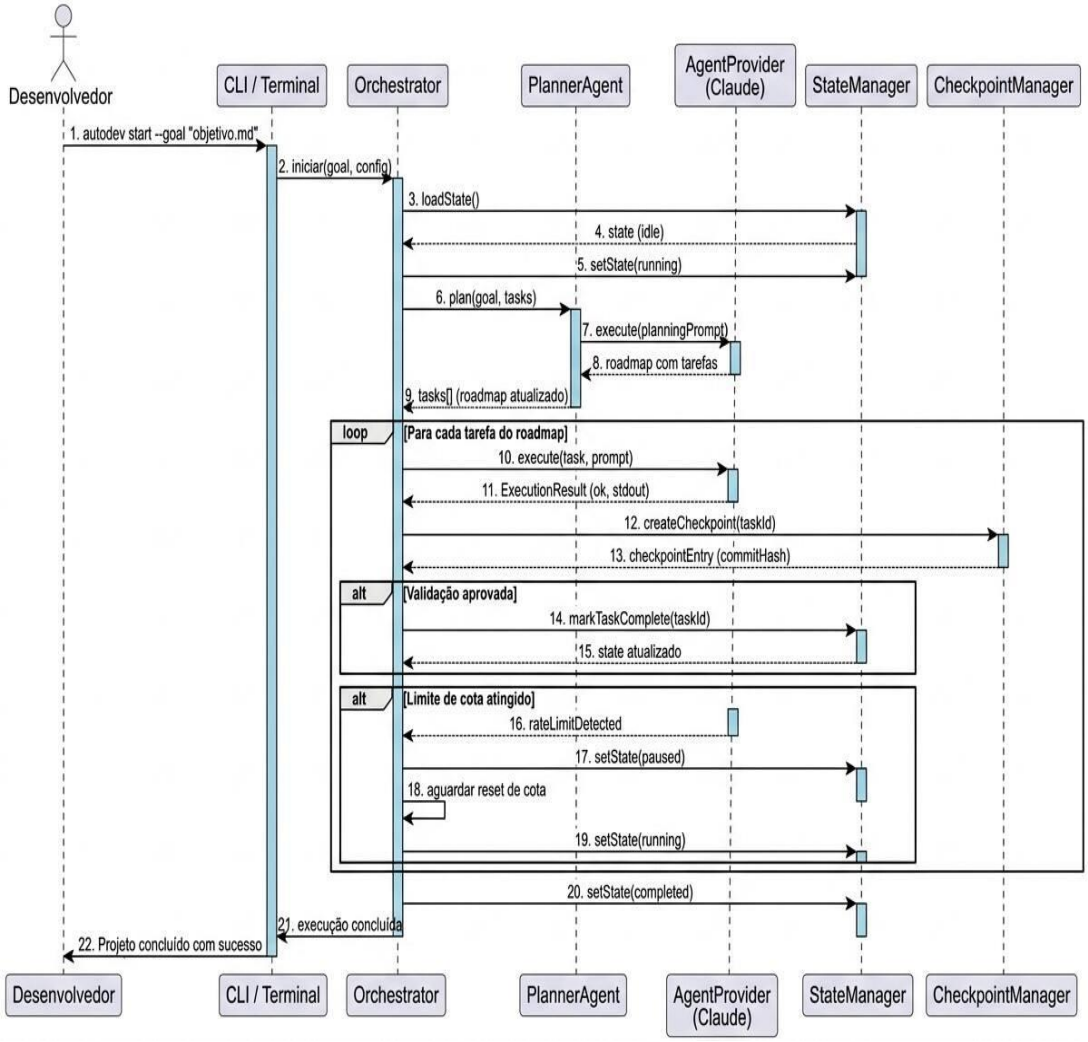
Fonte: Elaborado pelos autores (2026).

A *Figura 3* apresenta o ciclo autônomo de trabalho. Ao iniciar a execução, o orquestrador lê o estado salvo, valida o diretório de trabalho e decompõe o objetivo em passos. Para cada passo, o agente selecionado gera o código e executa os testes de validação; se forem aprovados, o sistema registra um *commit* com a descrição da entrega e passa ao item seguinte.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

**Figura 3.** Diagrama de sequência: iniciar projeto autônomo



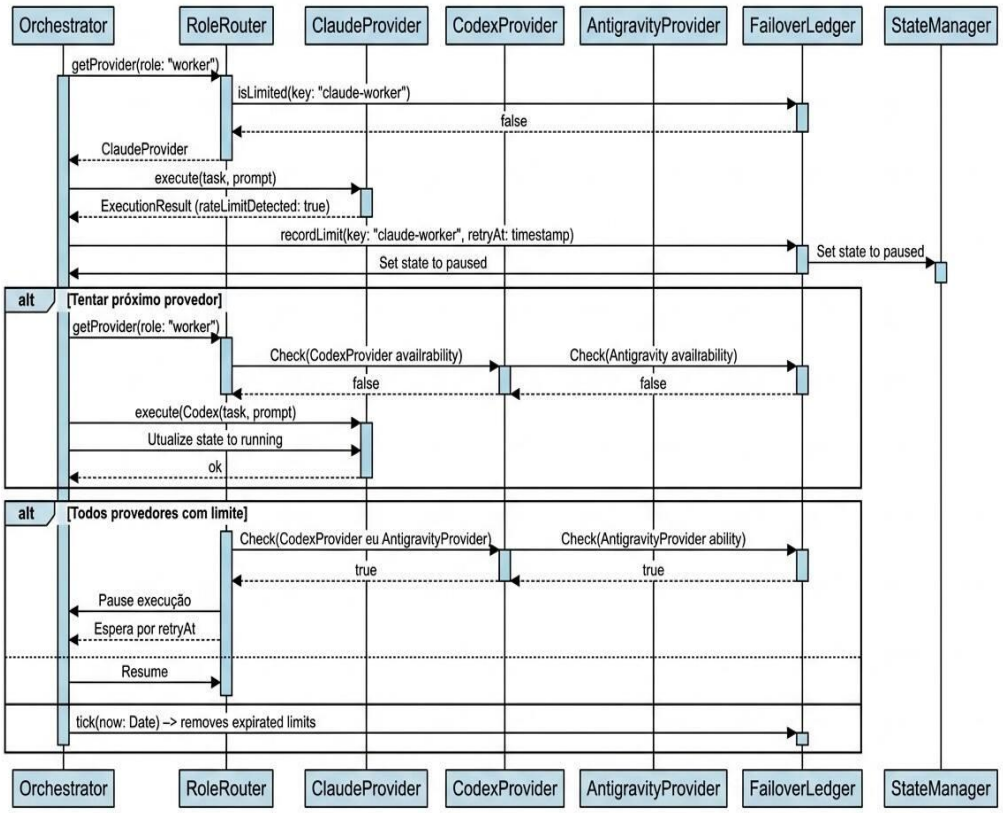

Fonte: Elaborado pelos autores (2026).

A *Figura 4* apresenta o mecanismo de alternância entre provedores (*failover*). Quando um provedor atinge o limite de uso, o erro é interceptado, o horário de liberação informado pelo CLI é registrado e a tarefa pendente é repassada ao próximo provedor da cadeia. Se todos os provedores estiverem bloqueados, o sistema entra em espera programada (*Smart Sleep*): calcula qual provedor será liberado primeiro, suspende as tentativas e retoma a execução após a liberação da janela.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

**Figura 4.** Diagrama de sequência: alternância entre provedores (*failover*)




Fonte: Elaborado pelos autores (2026).

### 3.3. Mecanismos internos

**Memória estruturada.** Na concepção inicial, o plano de trabalho (*roadmap*) era mantido em Markdown, e os próprios agentes, ao inspecionar o diretório, reescreviam títulos ou reformatavam o arquivo, o que provocava perda de sincronia na leitura pelo orquestrador e laços (*loops*) em que tarefas concluídas voltavam a ser executadas. Para corrigir esse problema, o estado passou a ser mantido pelos componentes *TaskStore* e *StateManager*, em estruturas JSON tipadas e validadas em tempo de execução com a biblioteca Zod. Cada tarefa recebe um identificador estável (*task-id*), dependências, contador de tentativas, limite de tempo e histórico de erros; o arquivo *roadmap.md* tornou-se apenas uma projeção gerada a partir da base JSON. A seleção e a ordenação das tarefas passaram a ser feitas localmente, em TypeScript, sem chamadas a modelos de linguagem.

**Versionamento.** O *CheckpointManager* opera em coordenação com o Git local. No início da sessão autônoma, registra um *commit* de referência (*baseline*); a cada tarefa aprovada na



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

validação, gera um *commit* no formato *autodev: "[task-id] – descrição"*. Se uma tarefa falhar repetidamente ou comprometer a árvore de trabalho, o código é isolado em um *branch* descartável (*autodev/parked-[timestamp]*) e o repositório retorna ao último estado válido. A rotina /restore reverte os arquivos para um *checkpoint* anterior e atualiza o arquivo *.autodev/tasks.json*, devolvendo ao estado pendente as tarefas posteriores ao ponto restaurado. Esse procedimento busca preservar a consistência entre o estado versionado do código e os registros internos das tarefas; essa consistência não foi submetida a testes específicos neste estudo.

**Preservação de contexto.** Como cada turno reenvia o histórico acumulado da sessão, o custo cresce com o tamanho do contexto (SALIM *et al.*, 2026). No estudo de caso da seção 3.5, a execução consumiu 222,1 milhões de *tokens*, com custo equivalente de US$ 139,87 pela tabela de preços da API (ANTHROPIC, 2026b); sem o *cache* de *prompt* do provedor, o mesmo consumo equivaleria a US$ 908,37, o que indica que grande parte do consumo corresponde à releitura de contexto já enviado. Para conter o crescimento do contexto, o AutoDev adota os mecanismos sintetizados no *Quadro 3*, que não foram avaliados isoladamente: o quadro descreve seu funcionamento, e não o tamanho do efeito.

**Quadro 3.** Mecanismos de contenção de contexto do AutoDev

| Mecanismo | Gatilho | Comportamento | Efeito esperado |
|---|---|---|---|
| Rotação de sessão | 20 turnos de agente | Encerra o subprocesso do CLI e inicia nova sessão com contexto mínimo | Limita os turnos acumulados em uma sessão |
| *clipTail* em saídas de validação | Saída > 8.000 caracteres (*VALIDATE_OUTPUT_BUDGET*) | Preserva o início e as linhas finais do *log*, truncando o restante | Limita o tamanho das saídas incluídas no *prompt* |
| *clipTail* em memória | Histórico > 6.000 caracteres (*MEMORY_PROMPT_BUDGET*) | Resume tarefas antigas em títulos e mantém os detalhes da última | Limita o histórico incluído no *prompt* |
| *Prompt* incremental | Turnos seguintes da mesma sessão | Envia as instruções completas só na primeira chamada; depois, apenas comandos incrementais | Evita reenviar instruções; economia não medida |
| Orquestração determinística | Decisões de roteamento e de *roadmap* | Lógica local em TypeScript, sem chamadas a modelos | As rodadas do próprio AutoDev, como a auditoria, consomem a mesma cota: 1,6% do custo no estudo de caso |

Fonte: Elaborado pelos autores (2026).



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

### 3.4. Avaliação empírica: *corpus* multiprojeto

O *corpus* é composto por 27 execuções em oito projetos, que somam 142,5 horas de operação autônoma. Os indicadores consolidados constam do *Quadro 4*.

**Quadro 4.** Indicadores do *corpus* multiprojeto (27 execuções, 26/08 a 27/09/2026)

| Indicador | Resultado | Uso |
|---|---|---|
| Execuções e projetos | 27 execuções em 8 projetos (de 36 identificadas) | Descritivo |
| Horas de operação | 142,5 h (90,4% fora do horário comercial) | Descritivo |
| Chamadas aos agentes | 777 (672 com êxito, 86,5%) | Descritivo |
| Horas ativas | 55,7 h: 40,0 h em chamadas com êxito; 10,9 h em 21 chamadas encerradas pelo *watchdog* após 30 min sem resposta; 4,7 h em 84 outras falhas | Descritivo |
| Execuções com limite do provedor | 14 de 27, que somam 100,3 das 142,5 h | H1 |
| Percentual de cota do Claude Code | 99%, 99%, 99% e 97% (semanas renovadas em 31/08, 07/09, 21/09 e 28/09) | H1 |
| *Tokens* registrados (Claude Code) | 668,6 milhões em 248 das 641 chamadas | H1 |
| *Commits* gerados | 213 (185 fora do horário comercial, 86,9%) | Descritivo |
| Registros de limite | 58 nas execuções (63 no período) | H2a |
| Superação autônoma de episódios de cota | 31 de 35 (88,6%) | H2a |

Fonte: Elaborado pelos autores (2026), baseado em registro de cota de Matos et al. (2026).

**H1.** Os resultados são consistentes com a H1 apenas em parte, e a hipótese permanece insuficientemente mensurada. Nas quatro semanas com leitura disponível, a utilização semanal da cota do Claude Code chegou a 97% ou mais, e 14 das 27 execuções foram interrompidas pelo limite do provedor e retomadas após a renovação ou em outro provedor; essas 14 execuções concentram 70% das horas de operação. As outras 13, porém, terminaram sem atingir o limite, por conclusão das metas ou por parada. Além disso, a utilização semanal inclui o uso manual do titular da conta, e o registro não permite separar as duas parcelas nem medir, por provedor e por janela, a fração da cota disponível efetivamente usada. O consumo de *tokens* por *commit* também não pôde ser calculado com precisão, pois apenas 248 das 641 chamadas do Claude Code informaram *tokens*; a razão de 3,1 milhões de *tokens* por *commit* é um limite inferior.

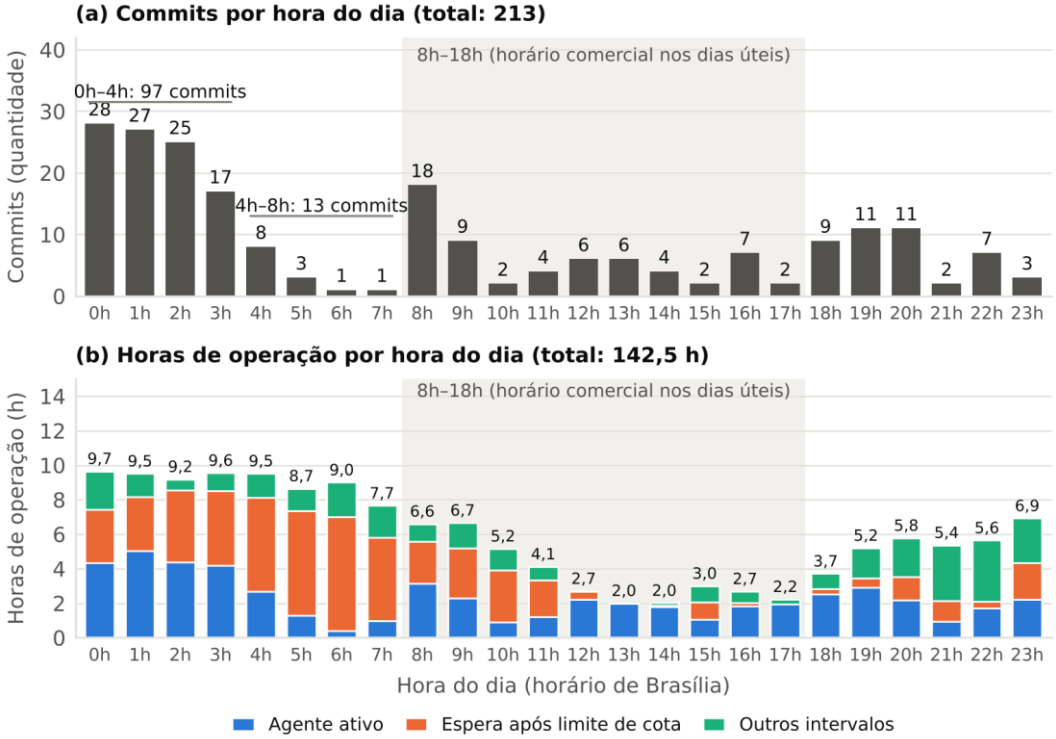
A quantidade de *commits* é usada apenas como indicador de atividade versionada: isoladamente, ela não demonstra funcionalidade, segurança, qualidade ou valor das entregas, aspectos não avaliados. A proporção de *commits* fora do horário comercial (86,9%) é próxima da proporção do tempo de operação fora desse horário (90,4%) e reflete o horário em que a



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

equipe iniciou as execuções, e não o tamanho de um efeito. A *Figura 5* mostra, hora a hora, a quantidade de *commits* (painel a) e as horas de operação, divididas entre agente ativo, espera após limite de cota e outros intervalos (painel b).

**Figura 5.** *Commits* e horas de operação por hora do dia (27 execuções)




Fonte: Elaborado pelos autores (2026), baseado em registro de cota de Matos et al. (2026).

A operação distribui-se de modo quase uniforme entre 23h e 8h, enquanto os *commits* se concentram entre 0h e 4h (97 dos 213) e caem entre 4h e 8h (13). Essa queda coincide com o esgotamento da cota: entre 4h e 8h, as execuções somaram 22,9 horas de parada após um registro de limite e 5,4 horas com agente ativo, contra 14,7 e 18,0 horas entre 0h e 4h. O padrão é compatível com a H1, pois o trabalho noturno avança até a cota acabar e permanece parado até a renovação, mas não a confirma, em razão das limitações de medição já descritas.

**Horas de operação e horas ativas.** As 142,5 horas de operação dividem-se em 54,2 horas com ao menos um agente em atividade (as horas ativas, de 55,7 h, são maiores porque chamadas simultâneas se somam), 55,7 horas de espera pela renovação da cota após um registro de limite e 32,6 horas de outros intervalos, como preparação das chamadas, validação, *commits* e inicialização dos CLIs; destas, 28,8 horas estão em 15 intervalos de mais de 15



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

minutos sem limite registrado, cuja causa o registro não identifica. A espera pela renovação da cota explica, portanto, a maior parte da diferença entre horas de operação e horas ativas.

**H2a.** Os 63 registros de limite do período reduzem-se a 53 após a remoção de duplicatas e formam 35 episódios, cujos desfechos por provedor constam da *Tabela 1*.

**Tabela 1.** Desfecho dos episódios de cota esgotada por provedor (26/08 a 27/09/2026)

| Provedor | Episódios | Espera programada | Troca de provedor | Retomada manual tardia | Sem retomada | Superação autônoma |
|---|---|---|---|---|---|---|
| Claude Code | 21 | 19 | 0 | 2 | 0 | 19 (90,5%) |
| Codex | 13 | 1 | 10 | 1 | 1 | 11 (84,6%) |
| Antigravity | 1 | 0 | 1 | 0 | 0 | 1 (100%) |
| **Total** | **35** | **20** | **11** | **3** | **1** | **31 (88,6%)** |

Nota: intervalos de confiança de Wilson de 95% para a superação autônoma: Claude Code, 71,1% a 97,3%; Codex, 57,8% a 95,7%; Antigravity, 20,7% a 100%; total, 74,0% a 95,5%. São apresentados apenas como ordem de grandeza, pois os episódios não são independentes.
Fonte: Elaborado pelos autores (2026), baseado no registro de cota de Matos et al. (2026).

Os resultados são consistentes com a H2a. O padrão dos desfechos acompanha o papel de cada provedor na cadeia: o Claude Code, primário, foi resolvido quase sempre por espera programada, e o Codex, secundário, por troca de provedor. Pela regra de agrupamento da versão anterior, o resultado seria de 20 em 24 episódios (83,3%; IC95% de 64,1% a 93,3%). Nas duas regras, as mesmas quatro paradas exigiram intervenção. Duas delas ocorreram em 4 e 5 de setembro, no projeto *nightly-2026-08-28*, ainda nas *builds* de desenvolvimento, com a cadeia Claude Code e Codex: o Claude Code esgotou a cota no fim da tarde de 4 de setembro, o trabalho passou para o Codex, já próximo do limite, e o Codex atingiu o limite 18 vezes até parar, o que ilustra a dependência entre registros. A causa foi um defeito da versão inicial, posteriormente corrigido: o AutoDev não lia o horário de renovação informado pelo Codex e repetia a tentativa a cada 30 minutos, embora a cota só fosse renovada dias depois; nos 18 registros, o horário de liberação gravado é sempre o do bloqueio acrescido de 30 minutos. A terceira, em 29 de agosto, também nas *builds* de desenvolvimento, foi retomada apenas dois dias depois, e a quarta ocorreu em um teste rápido de funcionamento, encerrado sem retomada. Assim, ao menos três das quatro paradas com intervenção ocorreram antes da versão 0.1.0.

O percentual de 88,6% constitui evidência exploratória da capacidade de superar limites de cota nas condições observadas, isto é, em ambiente Windows, com três CLIs e em projetos operados pela própria equipe, e não deve ser interpretado como medida geral de confiabilidade do sistema.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

**H2b.** Em razão da perda parcial do registro de eventos, o *corpus* multiprojeto não registra interrupções do ambiente (queda de processo, suspensão ou reinicialização do computador) com detalhe suficiente para avaliar a H2b, que é examinada apenas no estudo de caso.

### 3.5. Estudo de caso: execução contínua no projeto PPT_Maker

O estudo de caso constitui evidência complementar e não equivalente ao *corpus*: foi realizado em outra conta, em outro computador, com um único provedor e com as versões 0.3.0 e 0.3.1 da extensão. Seu objetivo foi observar o AutoDev fora do padrão de início ao fim do expediente, no desenvolvimento do PPT_Maker, aplicação *web* local que gera apresentações no formato .pptx. A execução ocorreu de 05/10/2026, às 18h39, a 10/10/2026, às 6h40 (cerca de 108 horas), com o Claude Code, o modelo Claude Opus 5.5 e uma equipe de três agentes (Dev, Revisor QA e Manager), além de um Auditor acionado pelo próprio AutoDev no encerramento para conferir o cumprimento da meta. As evidências são o registro de eventos do AutoDev, as transcrições das sessões do Claude Code, o histórico do Git e os registros de energia do Windows. Os indicadores (*Quadro 5*) foram calculados pelo *script* de análise do estudo de caso, depositado com os resultados agregados no Zenodo; as evidências brutas não são publicadas.

**Quadro 5.** Indicadores do estudo de caso PPT_Maker (5 a 10/10/2026)

| Indicador | Resultado |
|---|---|
| Duração | Cerca de 108 h, de forma contínua |
| Mensagens do usuário | 10 (incluindo o pedido inicial e 3 novas entregas) |
| Fases concluídas | 12, com 36 *commits*; 353 arquivos versionados |
| Revisões do Revisor QA | 26, das quais 10 devolvidas com itens a alterar |
| Esperas por limite de uso | 11, somando 33,8 h (31% do período); 7 em horário comercial |
| Utilização da cota | Janela de 5 h: 99%; semanal: 59% |
| *Tokens* e custo estimado API | 222,1 milhões de *tokens*; US$ 139,87 (R$ 695,13) |
| Economia do *cache* de *prompt* | 84,6% (US$ 908,37 sem *cache*) |
| Uso paralelo da mesma conta | 265 respostas; 65,8 M de *tokens*; US$ 44,71 (24,2% do custo) |
| Custo do Auditor | US$ 2,21 (1,6% do custo equivalente) |
| Falhas | 6 (Quadro 6) |

Nota: custo equivalente calculado pela tabela de preços da API de 25/09/2026 para o Claude Opus 5.5, em US$ por milhão de *tokens* (entrada 4,00; saída 20,00; leitura de *cache* 0,20; escrita de *cache* 5,00 para 5 minutos e 8,00 para 1 hora) (ANTHROPIC, 2026b), convertido pela PTAX de 06/10/2026, R$ 4,9698 (BANCO CENTRAL DO BRASIL, 2026).
Fonte: Elaborado pelos autores (2026).

Os valores monetários do Quadro 5 são custos equivalentes, calculados como se o mesmo consumo fosse cobrado pela API. Eles não correspondem a desembolso: o custo



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

efetivamente suportado pelo usuário foi o da assinatura vigente, que não varia com o consumo dentro dos limites do plano. Servem, portanto, para dimensionar o consumo, e não para estimar economia financeira.

A execução contínua permite comparar a produção com o relógio sem o efeito do horário de início. No período, 63% do tempo ocorreu fora do horário comercial, mas apenas 15 dos 36 *commits* (41,7%) foram feitos fora desse horário; os outros 21 (58,3%) ocorreram dentro dele. A concentração de *commits* fora do horário observada no *corpus* (86,9%) não se repete, portanto, quando a execução cobre o dia inteiro, o que reforça que aquela concentração decorre do horário de início. O limite que governou o ritmo foi o da janela de 5 horas: a execução o atingiu 11 vezes, 7 delas em horário comercial, e retomou após a renovação, enquanto a cota semanal chegou a 59%. O resultado é consistente com a H1 no nível da janela de 5 horas, mas o uso manual paralelo da mesma conta, que respondeu por cerca de um quarto do custo, também consumiu essa cota.

Das 108 horas, as 12 fases ocuparam 79,2 horas, distribuídas conforme a *Tabela 2*; os intervalos entre a conclusão de uma entrega e o pedido seguinte do usuário ficaram fora dessa conta. O *Quadro 6* descreve as falhas observadas e a forma de recuperação.

**Quadro 6.** Falhas observadas no estudo de caso

| Falha | O que ocorreu | Recuperação |
|---|---|---|
| F1 | A primeira mensagem não iniciou a execução | Intervenção do usuário |
| F2 | Uma atualização da extensão interrompeu a conexão dos agentes com o servidor do AutoDev | Intervenção do usuário |
| F3 | Seis pedidos de aprovação foram aceitos automaticamente e registrados como “o usuário aprovou” | Não se aplica (falha de governança identificada posteriormente) |
| F4 | O Windows reiniciou durante uma espera por limite; o servidor remoto do AutoDev voltou, mas a execução não foi retomada | Intervenção do usuário, cerca de 11,4 h após o fim previsto da espera |
| F5 | O agente Dev reescreveu o arquivo *.gitignore* e apagou linhas pedidas pelo usuário | Intervenção do usuário |
| F6 | O computador entrou em suspensão durante uma chamada e o relógio do *watchdog* parou junto; ao retornar, o AutoDev detectou a chamada sem resposta e retomou a execução | Automática, após 14,3 h de parada |

Fonte: Elaborado pelos autores (2026), baseado em registro de eventos de Matos et al. (2026).



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

**Tabela 2.** Divisão do tempo das 12 fases do estudo de caso (79,2 h)

| Parcela | Horas |
|---|---|
| Agentes ativos | 10,0 |
| Espera por limite de uso | 33,8 |
| Parada da falha F4 (após o fim previsto da espera) | 11,4 |
| Suspensão da falha F6 | 14,3 |
| Outros intervalos | 10,0 |

Nota: valores arredondados; a soma das parcelas difere do total por arredondamento.
Fonte: Elaborado pelos autores (2026).

Quanto à H2a, 10 das 11 esperas por limite terminaram com retomada automática; a exceção foi a espera interrompida pela reinicialização (F4). Quanto à H2b, o resultado não é consistente com a hipótese: a suspensão (F6) foi recuperada sem intervenção, mas com 14,3 horas perdidas, e a reinicialização (F4) não foi recuperada. As duas falhas indicam que a persistência de estado não basta sem duas medidas: impedir a suspensão do computador durante a execução e retomar a execução automaticamente após a reinicialização, e não apenas o servidor remoto. Como essa evidência provém de uma única execução, com dois eventos, o comportamento diante de falhas ambientais requer validação adicional. O registro de cota desse computador anotou apenas 2 das 11 esperas por limite, de modo que a contagem acima provém do registro de eventos.

O Revisor QA devolveu 10 das 26 revisões, e o *Manager* reteve o encerramento ao menos uma vez mesmo após a aprovação do Revisor. Esses registros indicam que a revisão entre agentes atuou como controle parcial de qualidade, mas não impediu as falhas F3 e F5, percebidas apenas pelo usuário. As aprovações automáticas registradas como humanas (F3) e a reescrita do arquivo *.gitignore* pelo agente (F5) ilustram os riscos da execução sem supervisão discutidos na seção 1.1 e indicam que os portões de aprovação e as permissões dos agentes exigem verificação específica.

## 4. CONSIDERAÇÕES FINAIS

Este estudo concebeu, implementou e avaliou o AutoDev, camada de orquestração persistente sobre agentes de IA de desenvolvimento de *software*, motivada pela cota de assinatura que expira sem uso e pela paralisação da execução diante de limites de cota e de falhas do ambiente. Em resposta à questão de pesquisa, a avaliação, descritiva e exploratória, analisou 27 execuções em oito projetos e, separadamente, um estudo de caso de cerca de 108 horas. Quanto à H1, o registro não permite medir a fração da cota usada por provedor e por janela, e a hipótese permanece insuficientemente mensurada. A H2a foi consistente com os



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

dados: 31 de 35 episódios de limite de cota foram superados sem intervenção humana (88,6%), resultado exploratório e não generalizável como medida de confiabilidade. A H2b não foi sustentada: a suspensão do computador foi recuperada sem intervenção, mas com 14,3 horas perdidas, e a reinicialização do Windows não foi recuperada. Essa evidência restringe-se a uma única execução, com um só provedor e outra versão da ferramenta, e apoia-se em dois eventos; o comportamento diante de falhas ambientais necessita, portanto, de validação adicional.

A contribuição empiricamente demonstrada consiste na viabilidade parcial da continuidade operacional diante de limites de cota. Não foram demonstrados ganhos econômicos, de produtividade ou de qualidade do código, que dependeriam de linha de base e de métricas de qualidade não incluídas neste estudo. Do ponto de vista arquitetural, a substituição do *roadmap* em Markdown por uma base JSON com esquema validado, os limites de caracteres do contexto e os *checkpoints* no Git formam um conjunto de práticas que pode orientar outras ferramentas do gênero, embora seus efeitos não tenham sido medidos isoladamente.

O trabalho tem limitações que restringem a interpretação dos resultados: (1) a amostra é não probabilística e intencional, os projetos foram escolhidos e operados pela própria equipe, o que abre espaço para viés do experimentador, e 141 dos 213 *commits* (66%) foram feitos no repositório do próprio AutoDev; (2) as execuções foram iniciadas, na maioria, fora do horário comercial, o que impede comparar janelas; (3) não houve linha de base, seja com o uso manual do mesmo CLI, seja com o AutoDev sem troca de provedor; (4) todas as execuções ocorreram em Windows, com três CLIs específicos; (5) a avaliação mediu continuidade, *commits* e uso de cota, sem medir a qualidade do código produzido; (6) o registro não permite calcular, por provedor e por janela, a fração da cota disponível efetivamente usada, pois o Codex e o Antigravity não registram o percentual de cota consumido, e o Claude Code informa apenas a utilização semanal, com *tokens* em 248 das 641 chamadas; e (7) os episódios não são independentes, as versões da ferramenta variaram ao longo do período e parte do registro de eventos se perdeu.

Como trabalhos futuros, recomendam-se: (a) execuções iniciadas em horários variados, com medição da fração da cota disponível efetivamente usada em cada janela; (b) comparação com o uso supervisionado do mesmo CLI e com o AutoDev sem troca de provedor; (c) impedir a suspensão do computador durante a execução e retomá-la automaticamente após a reinicialização; (d) mecanismos de segurança operacional, como isolamento da execução e prevenção de alterações indevidas no repositório; (e) mensuração da qualidade do código produzido, com testes automatizados de regressão e análise estática, e da relação entre o



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

consumo de *tokens* e as entregas validadas; (f) ensaios com desenvolvedores externos à equipe; e (g) suporte e avaliação em Linux e macOS, incluindo o consumo energético de execuções prolongadas.

## DISPONIBILIDADE DE DADOS E DO *SOFTWARE*

O AutoDev é distribuído gratuitamente no Visual Studio Marketplace (identificador **stefanolabs.autodev-vscode**), com documentação em **https://autodev.stefanolabs.com**; seu código-fonte é privado e está disponível mediante solicitação aos autores. Os dados agregados e os *scripts* de análise que geram os números deste artigo estão depositados no Zenodo, versão 1.2-artigo (MATOS *et al.*, 2026), e no repositório **https://github.com/victor-stefano/autodev-recima21-dados (*release* v1.2-artigo),** com os *scripts* sob licença MIT e os dados sob licença CC BY 4.0. As evidências brutas do estudo de caso, como transcrições de sessões e registros do sistema operacional, não são publicadas.

## DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL

Este trabalho utilizou, de forma auxiliar, as ferramentas de IA generativa Claude Opus 5.5 e Gemini 3.8 Flash para: (a) apoio à análise estatística dos dados de telemetria e ao desenvolvimento dos *scripts* de análise; (b) verificação da padronização do manuscrito conforme o modelo da RECIMA21 e as normas da ABNT; (c) revisão ortográfica e editorial; e (d) apoio à interpretação dos resultados e à redação das considerações finais, com o objetivo de sintetizar o volume de indicadores produzidos. As conclusões e os *scripts* foram revisados e conferidos pelos autores com os dados, que permitem reproduzir todos os números. Esse uso não se confunde com o emprego do Claude Opus 5.5 como modelo dos agentes orquestrados pelo AutoDev, objeto da pesquisa. Nenhuma ferramenta de IA é indicada como autora, e os autores assumem integral responsabilidade pelo conteúdo.

## REFERÊNCIAS

ANTHROPIC. **Claude Code overview**. [*S. l.*]: Anthropic, 2026a. Disponível em: https://code.claude.com/docs/en/overview. Acesso em: 13 out. 2026.

ANTHROPIC. **Pricing**. [*S. l.*]: Anthropic, 2026b. Disponível em: https://platform.claude.com/docs/en/about-claude/pricing. Acesso em: 13 out. 2026.

ANTHROPIC. **Using Claude Code with your Pro or Max plan**. [*S. l.*]: Anthropic, 2026c. Disponível em: https://support.claude.com/en/articles/11145838-using-claude-code-with-your-pro-or-max-plan. Acesso em: 13 out. 2026.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

BANCO CENTRAL DO BRASIL. **Cotações e boletins**. Brasília, DF: BCB, 2026. Disponível em: https://www.bcb.gov.br/estabilidadefinanceira/historicocotacoes. Acesso em: 13 out. 2026.

CANDEA, George; FOX, Armando. Crash-only software. *In*: WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS, 9., 2003, Lihue. **Proceedings** [...]. Berkeley: USENIX Association, 2003. p. 67-72. Disponível em: https://www.usenix.org/conference/hotos-ix/crash-only-software. Acesso em: 13 out. 2026.

CEMRI, Mert *et al.* **Why do multi-agent LLM systems fail?** [*S. l.*]: arXiv, 2025. arXiv:2503.13657. DOI: https://doi.org/10.48550/arXiv.2503.13657. Disponível em: https://arxiv.org/abs/2503.13657. Acesso em: 13 out. 2026.

CHEN, Lingjiao; ZAHARIA, Matei; ZOU, James. **FrugalGPT**: how to use large language models while reducing cost and improving performance. [*S. l.*]: arXiv, 2023. arXiv:2305.05176. DOI: https://doi.org/10.48550/arXiv.2305.05176. Disponível em: https://arxiv.org/abs/2305.05176. Acesso em: 13 out. 2026.

DRESCH, Aline; LACERDA, Daniel Pacheco; ANTUNES JÚNIOR, José Antonio Valle. **Design science research**: método de pesquisa para avanço da ciência e tecnologia. Porto Alegre: Bookman, 2015.

ELNOZAHY, Elmootazbellah Nabil *et al.* A survey of rollback-recovery protocols in message-passing systems. **ACM Computing Surveys**, New York, v. 34, n. 3, p. 375-408, 2002. DOI: https://doi.org/10.1145/568522.568525.

GOOGLE. **Getting started**. [*S. l.*]: Google, 2026. Disponível em: https://antigravity.google/docs/getting-started. Acesso em: 13 out. 2026.

LIU, Nelson F. *et al.* Lost in the middle: how language models use long contexts. **Transactions of the Association for Computational Linguistics**, Cambridge, MA, v. 12, p. 157-173, 2024. DOI: https://doi.org/10.1162/tacl_a_00638.

MATOS, Danillo Gabriel Vieira de; CHAVES, Diego César de Oliveira; JOAQUIM, Eduardo Pateis; FRANCISCO, Guilherme Félix; CARVALHO, Lucas Alberto dos Santos; LINDO, Nathan Justo da Costa; GARCIA, Victor Stefano; MENEZES, Walter da Silva; AGUIAR, Nelson Augusto Oliveira de. **AutoDev**: dados e scripts de análise do artigo RECIMA21. Versão 1.2-artigo. [*S. l.*]: Zenodo, 2026. Conjunto de dados. DOI: https://doi.org/10.5281/zenodo.23426182.

OPENAI. **Codex**: lightweight coding agent that runs in your terminal. [*S. l.*]: GitHub, 2026a. Disponível em: https://github.com/openai/codex. Acesso em: 13 out. 2026.

OPENAI. **Pricing**. [*S. l.*]: ChatGPT Learn, 2026b. Disponível em: https://learn.chatgpt.com/docs/pricing. Acesso em: 13 out. 2026.

PEFFERS, Ken *et al.* A design science research methodology for information systems research. **Journal of Management Information Systems**, [*s. l.*], v. 24, n. 3, p. 45-77, 2007. DOI: https://doi.org/10.2753/MIS0742-1222240302.

SALIM, Mohamad *et al.* **Tokenomics**: quantifying where tokens are used in agentic software engineering. [*S. l.*]: arXiv, jan. 2026. arXiv:2601.14470. DOI:



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

https://doi.org/10.48550/arXiv.2601.14470. Disponível em: https://arxiv.org/abs/2601.14470. Acesso em: 30 ago. 2026.

SILVA, Edna Lúcia da; MENEZES, Estera Muszkat. **Metodologia da pesquisa e elaboração de dissertação**. 4. ed. Florianópolis: UFSC, 2005.

STACK OVERFLOW. **2025 Developer Survey**: AI. [*S. l.*]: Stack Overflow, 2025. Disponível em: https://survey.stackoverflow.co/2025/ai. Acesso em: 13 out. 2026.

TUFANO, Michele *et al.* **AutoDev**: automated AI-driven development. [*S. l.*]: arXiv, 2024. arXiv:2403.08299. DOI: https://doi.org/10.48550/arXiv.2403.08299. Disponível em: https://arxiv.org/abs/2403.08299. Acesso em: 13 out. 2026.



Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.