# Uma Abordagem Orientada a Dados de Proveniência para Rastreabilidade de *Workflows* Agênticos

**Valesca Moura de Sousa**[1], **Sandro Rama Fiorini**[2], **Daniel de Oliveira**[1]

[1]Instituto de Computação – Universidade Federal do Fluminense (UFF)

[2]Instituto PUC-Behring de Inteligência Artificial

valescamoura@id.uff.br, srfiorini@puc-rio.com.br, danielcmo@ic.uff.br

***Abstract.*** *With the increasing use of AI agents in workflow specification and automation, LLMs are increasingly responsible for making decisions about tasks and data dependencies. However, due to their susceptibility to hallucinations, these models may compromise the validity of results, making it essential to track and analyze their decision-making processes. Capturing such information is challenging, as these processes are often not explicit. In this paper, we propose a provenance capture approach for agentic workflows to enable the analysis of agents' decisions and their impact on the generated workflows. This approach is based on instrumenting agentic frameworks and representations in accordance with the W3C PROV standard, providing greater granularity than existing approaches.*

***Resumo.*** *Com o crescente uso de agentes de IA na especificação e automação de workflows, LLMs passaram a ser responsáveis por decidir sobre tarefas e dependências de dados. Entretanto, por serem suscetíveis a alucinações, esses modelos podem comprometer a validade dos resultados, tornando essenciais o rastreamento e a análise de suas decisões. A captura desses dados é desafiadora, pois o processo decisório frequentemente não é explícito. Neste artigo, propomos uma abordagem de captura de proveniência para workflows agênticos, a fim de permitir a análise das decisões dos agentes e do impacto dessas decisões no workflow gerado. Essa captura é baseada na instrumentação de frameworks de agentes e na representação conforme o padrão W3C PROV, oferecendo maior granularidade do que os trabalhos existentes.*

## 1. Introdução

O uso da abstração de *workflows* para representar etapas de simulações computacionais consolidou-se como padrão *de fato* na academia e na indústria [de Oliveira et al. 2019]. Diversos sistemas, *e.g.*, o Pegasus [Deelman et al. 2021] e o Parsl [Babuji et al. 2019], foram propostos para apoiar usuários na especificação, execução, monitoramento e análise de *workflows*. Nesses sistemas, o usuário define e implementa as tarefas e as dependências na linguagem do próprio sistema. Por exemplo, no Parsl, *workflows* são descritos em Python por meio de *decorators* que indicam paralelismo, enquanto no Pegasus as dependências entre tarefas são explicitadas em arquivos XML. Essa especificação não é trivial, pois a mesma tarefa pode ter múltiplos programas candidatos, e a escolha da alternativa mais adequada depende de diversos fatores. Além disso, a curva de aprendizado para especificar *workflows* nesses sistemas pode ser elevada.

Com o advento dos Modelos de Linguagem de Larga Escala (LLMs) [Zhao et al. 2023] e sua ampla adoção em agentes de IA [Wang et al. 2024], o processo de

especificação de *workflows* vem se transformando. Esses agentes são capazes de inferir automaticamente tarefas, dependências e parâmetros, dando origem aos chamados *Workflows Agênticos* [Schick et al. 2023]. Em geral, tais decisões são guiadas por descrições em linguagem natural fornecidas pelos usuários, incluindo os objetivos do *workflow*, os dados de entrada e as saídas esperadas. O uso desses agentes traz vantagens como: *(i)* decomposição de problemas complexos em subtarefas, *(ii)* coordenação de múltiplas ações e *(iii)* integração de diferentes ferramentas e serviços, *e.g.*, APIs, em um único processo de execução.

Entretanto, o uso de agentes de IA na especificação de *workflows* ainda não dispõe de uma solução definitiva. Esses agentes podem alucinar, uma vez que se baseiam em LLMs em sua camada subjacente [Huang et al. 2025], especialmente devido à qualidade e à formulação dos *prompts* de entrada. No contexto de *workflows* agênticos, esse problema pode ser ainda mais acentuado, pois múltiplas decisões são encadeadas até a definição final da especificação do *workflow*. As alucinações podem ocorrer por diferentes razões, *e.g.*, *prompts* mal elaborados, descrições incompletas, *etc*.

Para mitigá-las, diferentes estratégias podem ser adotadas, *e.g.*, abordagens *human-in-the-loop* [Gómez-Carmona et al. 2024]. No entanto, essas estratégias dependem do acesso ao histórico de decisões e à derivação dos dados que levaram o agente à especificação do *workflow*, *i.e.*, de seus dados de *Proveniência* [Freire et al. 2008]. O principal desafio é que, atualmente, esses dados de proveniência não estão devidamente estruturados nos *frameworks* e nos sistemas. Embora algumas informações sejam coletadas, elas geralmente estão dispersas em múltiplos arquivos de *log*, enquanto outras sequer são registradas. Esse cenário compromete a auditabilidade e a capacidade analítica dos *workflows* agênticos, além de dificultar a adoção de estratégias complementares para mitigação de alucinações.

Neste artigo, propomos uma abordagem orientada a dados de proveniência para possibilitar a análise de *workflows* agênticos, permitindo ao usuário compreender claramente quais decisões foram tomadas pelo agente de IA e a partir de quais *prompts* de entrada. Esse nível de compreensão auxilia na identificação de problemas e na adoção de medidas corretivas e preventivas. Além disso, em determinados contextos de aplicação, a rastreabilidade de decisões é mandatória, inclusive por razões legais. A abordagem proposta, denominada `ANCHOR` (*ANalysis of agentiC workflows via HistOry and pRovenance*), realiza a captura de dados por meio da instrumentação de *frameworks* agênticos, *e.g.*, BeeAI [BeeAI 2026], permitindo a extração de dados de proveniência com semântica específica para *workflows* gerados por LLMs, o que favorece análises mais contextualizadas. Diferentemente do estado da arte, a `ANCHOR` captura dados com maior granularidade. A `ANCHOR` foi avaliada por meio da especificação de um *workflow* de Biologia Computacional e submetida a consultas analíticas, com resultados promissores.

## 2. Conceitos Importantes e Trabalhos Relacionados

**Dados de Proveniência.** Os dados de proveniência [Freire et al. 2008] representam o histórico de derivação dos dados, permitindo compreender sua origem, transformações e ações que influenciaram sua geração ou modificação. Esse tipo de registro é fundamental para garantir a reprodutibilidade, a auditabilidade e a rastreabilidade, especialmente em ambientes complexos com aplicações de IA. Os dados de proveniência podem ser categorizados em dois grupos [Freire et al. 2008]: *(i)* proveniência *Prospectiva* (*p-prov*), que descreve a especificação de *workflows* ainda não executados, e *(ii)* proveniência *Retrospectiva* (*r-prov*), que captura metadados de eventos já ocorridos, registrando o histórico efetivo de derivação

dos dados. Essa distinção permite comparar o comportamento esperado com o observado.

Ao longo das últimas décadas, diversos modelos de proveniência foram propostos. Contudo, o padrão de referência é o W3C PROV [Missier et al. 2013]. O PROV organiza a proveniência em três conceitos centrais: *Entidades*, *Atividades* e *Agentes*. *Entidades* são objetos ou dados com estado persistente em determinado instante; *Atividades* são processos ou ações que geram, consomem ou modificam entidades; e *Agentes* são os responsáveis por iniciar, controlar ou supervisionar tais atividades. As relações entre esses elementos, *e.g.*, *wasGeneratedBy*, *used*, *wasDerivedFrom*, *wasAssociatedWith*, *wasInformedBy*, *actedOnBehalfOf* e *wasAttributedTo*, definem a estrutura semântica do grafo de proveniência, conectando entidades, atividades e agentes para representar o fluxo de derivação dos dados.

***Workflows* Agênticos.** Os *Workflows Agênticos* [Schick et al. 2023] são aqueles em que um ou mais agentes de IA colaboram na especificação, planejamento, execução e validação de tarefas, bem como na produção de dados e de artefatos intermediários e finais. Diferentemente de modelos tradicionais de IA, agentes de IA combinam *(i)* capacidades de raciocínio, como a decomposição de tarefas, o planejamento e a tomada de decisão, e *(ii)* capacidade de atuar no ambiente por meio da invocação de programas e de fontes de dados. Essa integração permite a execução iterativa e orientada a objetivos, sensível ao contexto. Ao contrário dos *workflows* tradicionais, tipicamente definidos como um fluxo "fixo" de tarefas, *workflows* agênticos apresentam comportamento dinâmico e adaptativo. A definição de tarefas e o fluxo de controle podem emergir durante a execução, com base em decisões locais dos agentes e no estado dos dados. Essa característica é particularmente relevante em *workflows* de integração de dados, em que as fontes podem evoluir.

Os *workflows agênticos* podem ser modelados como grafos de execução distribuídos, nos quais os nós representam agentes e as arestas, fluxos de dados e dependências de controle. Cada agente pode encapsular funções como a execução de consultas e a validação de resultados. A interação entre agentes pode ocorrer de forma síncrona ou assíncrona, frequentemente mediada por filas de mensagens ou por mecanismos de memória persistente. Uma característica fundamental desses *workflows* é a presença de ciclos de retroalimentação, em que resultados intermediários são avaliados e utilizados para refinar as decisões subsequentes. Em cenários multiagente, podem emergir estratégias de coordenação como delegação de tarefas, negociação e composição incremental de resultados.

Ao contrário dos *workflows* determinísticos, o resultado de um *workflow* agêntico depende das interações e decisões dos agentes, o que introduz não-determinismo [Pinhanez et al. 2025]. Tal característica decorre da natureza probabilística dos mecanismos de raciocínio, sujeitos a imprecisões ou alucinações. Consequentemente, a reprodutibilidade e a interpretabilidade tornam-se desafios centrais. Nesse cenário, dados de proveniência desempenham papel fundamental, pois permitem registrar não apenas entradas e saídas, mas também o histórico de execução, incluindo decisões dos agentes, consultas, transformações e interações entre componentes. Esses metadados são essenciais para auditoria, depuração, explicabilidade e análise de qualidade, além de possibilitarem a reprodução, total ou parcial, das execuções. Assim, adaptar e estender mecanismos de captura de proveniência torna-se essencial para o suporte a *workflows* agênticos.

**Trabalhos Relacionados.** A captura de proveniência tornou-se central para *workflows* agênticos, a fim de garantir transparência e auditabilidade. É necessário compreender não apenas como os dados são criados e transformados, mas também por que determinadas

decisões são tomadas, dado o caráter não determinístico dos agentes e sua suscetibilidade a alucinações. Esse requisito não é atendido por abordagens tradicionais de captura de proveniência em *workflows*, como o noWorkflow [Pimentel et al. 2017] e a DfAnalyzer [Silva et al. 2018]. Existem, porém, trabalhos que buscam abordar a captura de proveniência em *workflows* agênticos. O Flowcept [Souza et al. 2023] fornece captura de proveniência e telemetria, integrando-se ao código por meio de adaptadores. A ferramenta registra execução de tarefas, caminhos de derivação, *prompts*, telemetria e chamadas de ferramentas, facilitando a análise posterior do *workflow*. O Flowcept segue o padrão W3C PROV [Souza et al. 2025], mas não contempla explicitamente interações multiagente, como delegação, cooperação e hierarquias.

[Sirqueira et al. 2018] exploram o uso de proveniência em sistemas multiagentes sob uma perspectiva baseada na arquitetura *Belief-Desire-Intention*, o que permite reconstruir o comportamento do sistema e analisar suas decisões. A abordagem segue a recomendação W3C PROV e possibilita a análise de interações e execuções. No entanto, esse e outros trabalhos [Friedman et al. 2020, Davis et al. 2017] concentram-se em sistemas multiagentes tradicionais, sem contemplar *workflows* agênticos baseados em LLMs e que possuam delegação e cooperação. [Chang and Echizen 2025] propõem um mecanismo de rastreamento de proveniência baseado na incorporação de informações no próprio conteúdo gerado, permitindo reconstruir *a posteriori* a sequência de agentes responsáveis pelos dados. Contudo, o modelo limita-se a cadeias lineares de geração e não contempla aspectos estruturais de *workflows*, nem o contexto de decisão, o uso de ferramentas ou interações complexas.

## 3. A Abordagem ANCHOR

Esta seção apresenta a abordagem ANCHOR, voltada à captura de dados de proveniência em *workflows* agênticos, destacando seu modelo de proveniência e sua arquitetura.

**Representação dos Dados de Proveniência na ANCHOR.** A modelagem de proveniência em *workflows* agênticos impõe desafios que abordagens tradicionais não contemplam adequadamente. Nesses ambientes multiagentes, a execução não segue uma sequência fixa de tarefas, mas envolve decisões, interações, delegações e uso contextual de capacidades. Assim, o modelo proposto não se restringe a *p-prov* ou *r-prov*, incorporando também o contexto e o processo decisório dos agentes. Esse tipo de proveniência, denominado *a-prov* (proveniência agêntica), oferece uma representação do comportamento do sistema.

O esquema de dados da ANCHOR, apresentado na Figura 1, adota um modelo centrado em interações (*Interaction-Centric Model* [Plötzky et al. 2025]), no qual a interação é a unidade básica de representação. Diferentemente de abordagens centradas em tarefas, aqui o encadeamento causal entre eventos é explicitamente modelado. Nesse contexto, *Interaction* é a entidade central, conectando os elementos do *workflow* agêntico e registrando eventos com semântica própria, incluindo agente executor, intenção, tarefa associada e relações causais com interações anteriores (*e.g.*, *caused_by_interaction_id*).

*Entity* representa os elementos que executam ou participam das interações, como agentes, usuários, ferramentas ou componentes de sistema, incluindo relações hierárquicas como supervisão e delegação. *Capability* modela recursos e ferramentas disponíveis aos agentes e seu uso efetivo ao longo das interações. *Intention* explicita o objetivo associado a cada interação, permitindo capturar o processo decisório do agente e reconstruir a lógica subjacente ao *workflow*. *TaskExecution* representa a execução de tarefas, equivalente às

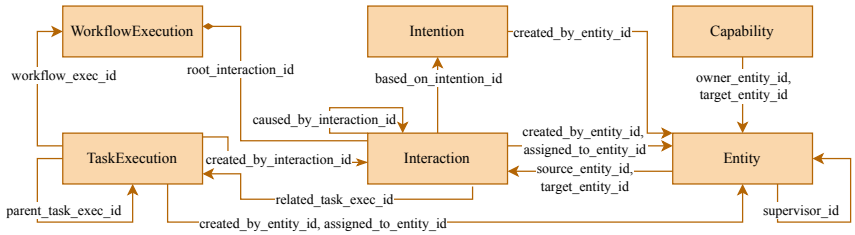


**Figura 1. Esquema do Banco de Dados de Proveniência da ANCHOR.**

noções de *p-prov* e *r-prov*, registrando entradas, saídas e métricas de desempenho.

O esquema também inclui artefatos de dados, configurações de modelos e métricas de execução, relacionando-os às tarefas que os produzem ou consomem, o que permite capturar fluxos de dados e controle de forma integrada. A entidade *WorkflowExecution* agrega informações globais da execução. Em conjunto, o modelo explicita estados intermediários de raciocínio (*Intention*), o uso de capacidades e relações de coordenação entre agentes, como delegação e supervisão, tornando explícitas informações frequentemente implícitas.

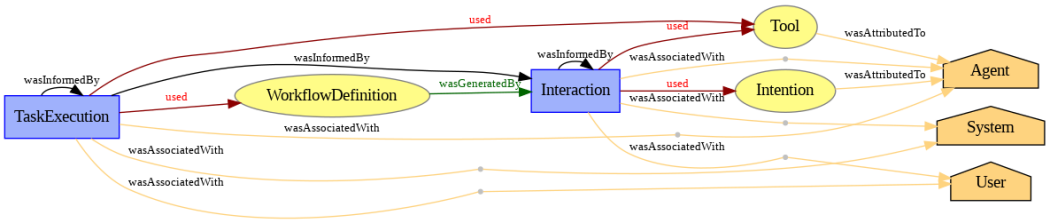


**Figura 2. Mapeamento do Esquema de Proveniência da ANCHOR com o PROV.**

Para viabilizar a interoperabilidade, o esquema é mapeado para o W3C PROV (Figura 2). Nesse mapeamento, interações são representadas como *prov:Activity*; intenções, ferramentas, configurações e artefatos como *prov:Entity*; e agentes como *prov:Agent*. Relações como uso, geração, associação, atribuição e delegação são expressas por predicados PROV (*e.g.*, *prov:used*, *prov:wasGeneratedBy*, *prov:wasAssociatedWith*). Isso permite representar dependências causais e fluxos de dados de forma padronizada, preservando os principais aspectos estruturais do modelo. Embora promova interoperabilidade, esse mapeamento implica perda parcial de expressividade, pois certos conceitos não possuem correspondência direta no PROV, como *Capability*. Além disso, relações semanticamente distintas podem ser mapeadas para o mesmo predicado, exigindo qualificadores adicionais para preservar distinções.

Dessa forma, ao importar uma representação em PROV para o seu esquema de proveniência, os sistemas podem interpretar *prov:Entity*, *prov:Activity* e *prov:Agent* de maneiras distintas, especialmente em cenários de *workflows* agênticos, em que ainda não há consenso quanto à modelagem conceitual do domínio. Como consequência, estruturas semanticamente equivalentes podem ser representadas de formas distintas ou, inversamente, estruturas semelhantes podem carregar significados distintos. Ainda assim, o mapeamento preserva as principais dependências causais e estruturais do modelo, sendo adequado para fins de padronização e interoperabilidade em conformidade com a recomendação W3C PROV.

**Arquitetura da ANCHOR.** Para popular o banco de dados de proveniência, é necessário capturar as informações definidas no esquema da Figura 1. Na ANCHOR, essa captura é rea-

lizada por meio de instrumentação do código e do uso de bibliotecas de telemetria. Em sua versão atual, a abordagem utiliza o *OpenInference* [Arize AI 2024], que estende o *OpenTelemetry* [Cloud Native Computing Foundation 2024] para coletar dados de execução, incluindo chamadas a modelos, uso de ferramentas e métricas associadas. Embora esses mecanismos representem um avanço na coleta de dados, as informações são majoritariamente estruturadas como *spans*, isto é, registros de operações individuais dentro de um *trace* (execução de um *workflow* agêntico), contendo contexto e metadados. Como consequência, elementos como intenções, estados intermediários de raciocínio (*thoughts*), capacidades disponíveis e relações causais permanecem implícitos nos *spans*.

A ANCHOR segue a arquitetura apresentada na Figura 3 e é composta por quatro componentes principais: (i) Capturador de Telemetria, (ii) Estruturador de Proveniência, (iii) Banco de Dados de Proveniência e (iv) Mapeador PROV. A abordagem pode ser integrada a múltiplos *frameworks* de agentes, *e.g.*, BeeAI [BeeAI 2026], Agno [Agno 2026] e CrewAI [CrewAI 2026], desde que estes sejam instrumentados para enviar dados ao *OpenInference*. Em sua versão atual, a ANCHOR está integrada ao BeeAI.

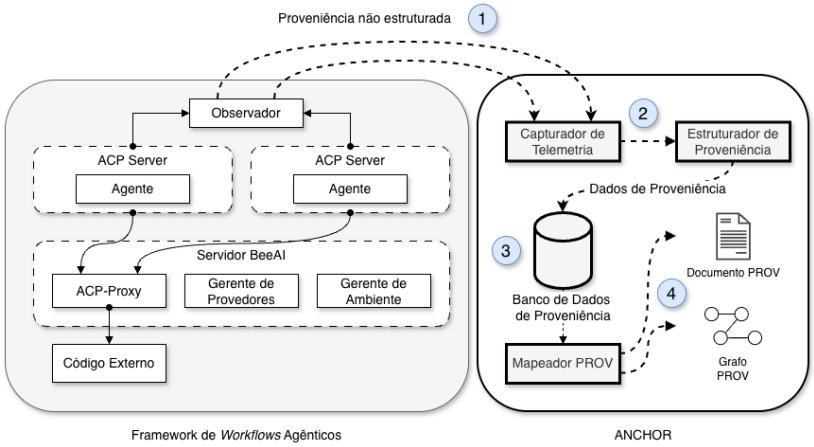


**Figura 3. Arquitetura do ANCHOR acoplada ao BeeAI.**

A Figura 3 ilustra a ANCHOR integrada ao BeeAI (cujos detalhes de funcionamento estão fora do escopo do artigo). O BeeAI é composto por agentes que se comunicam entre si e por um componente *Observador*, responsável por coletar dados de telemetria. A partir dele, o *Capturador de Telemetria* da ANCHOR inicia sua execução (passo ①). Esse componente importa *spans* e identifica informações relevantes para o povoamento da base de proveniência. Como a ANCHOR utiliza *OpenTelemetry*, o *Capturador de Telemetria* extrai informações no formato JSON gerado pela ferramenta, composto por pares chave-valor ainda não estruturados. A Figura 4 mostra um fragmento no qual se observa um *trace* associado ao agente *Orchestrator*, que utilizou o modelo *gpt-4*, bem como o *prompt* e a resposta gerada. Essas informações permitem derivar dados de proveniência, embora ainda não estruturados conforme o esquema da ANCHOR.

Uma vez extraídos os dados dos *spans*, o componente *Estruturador de Proveniência* é acionado (passo ②), mapeando-os para as entidades do esquema da Figura 1. Esse processo utiliza classes Pydantic como camada intermediária para transformar dados brutos em representações estruturadas como *Interaction*, *Entity* e *Intention*. Assim, o Pydantic atua como ponte entre *logs* de baixo nível e o modelo conceitual da ANCHOR, com validação e

```
{
  "trace_id": "9c2a1f3b7d5e4a1f8b6c3d2e9a7f4c11",
  "span_id": "a1b2c3d4e5f67890",
  "name": "llm.generate",
  "start_time": "2026-03-28T10:15:30Z",
  "end_time": "2026-03-28T10:15:42Z",
  "attributes": {
    "agent.name": "Orchestrator",
    "llm.model": "gpt-4",
    "llm.prompt": "Generate a workflow for processing genomic data using multiple agents.",
    "llm.response": "Step 1: Ingest data...Step 2: Preprocess sequences..."
  }
}
```


**Figura 4. Fragmento do JSON importado do *OpenTelemetry***

tipagem forte. Os dados estruturados são então armazenados no *Banco de Dados de Proveniência* (passo ③), implementado com TinyDB e espelhado em SQLite, ambos embutidos e sem necessidade de servidor. Essa duplicidade aumenta a flexibilidade na elaboração de consulta ao fornecer dois modelos para responder a consultas.

A partir do banco de dados, o *Mapeador PROV* realiza a conversão para o padrão W3C PROV (passo ④), utilizando a biblioteca `prov` do Python[1]. O resultado é armazenado em um banco de dados do Neo4j, permitindo consultas orientadas a grafos e a exportação em formatos como JSON. Além disso, os dois modelos de armazenamento são complementares: o esquema da `ANCHOR` (tanto no TinyDB quanto no SQLite) suporta consultas semânticas ricas (intenções, hierarquias e interações), enquanto o PROV (implementado no Neo4J) favorece a interoperabilidade. Essa separação equilibra expressividade e padronização. O código-fonte da `ANCHOR` é aberto e pode ser obtido em `https://github.com/UFFeScience/anchor`.

## 4. Avaliação Experimental

Esta seção apresenta a avaliação experimental da `ANCHOR`, incluindo uma análise do *overhead* introduzido e uma avaliação qualitativa que demonstra seu suporte a consultas analíticas típicas de usuários. Os experimentos utilizam como estudo de caso um *workflow* de mineração de árvores filogenéticas da Biologia Computacional.

**Estudo de Caso.** O *workflow* utilizado nos experimentos realiza a descoberta de subárvores frequentes em bases de dados de árvores filogenéticas. Uma árvore filogenética é uma estrutura em forma de árvore que representa as relações evolutivas de parentesco entre diferentes espécies, organismos ou grupos, com base em ancestrais comuns. Essa árvore pode ser gerada de diversas maneiras a partir do mesmo conjunto de dados de entrada, dependendo do método de inferência adotado. Nesse contexto, a identificação de padrões recorrentes em árvores filogenéticas torna-se relevante tanto para análises evolutivas quanto para a descoberta de relações entre espécies. O *workflow* original foi proposto por [Moraes et al. 2025] e é composto por sete etapas: *(i)* validação de sequências de DNA e RNA; *(ii)* alinhamento múltiplo de sequências; *(iii)* seleção do modelo evolutivo; *(iv)* construção da árvore filogenética; *(v)* geração de subárvores; *(vi)* mapeamento de subárvores; e *(vii)* cálculo de similaridade. Cada etapa executa um programa ou serviço distinto.

Neste artigo, adaptamos o *workflow* estático definido por [Moraes et al. 2025] para um contexto multiagente, originando uma versão agêntica composta pelos seguintes elemen-

[1]`https://pypi.org/project/prov/`

tos: *(i)* um agente *Orchestrator*, responsável por receber *prompts* do usuário e delegar tarefas aos demais agentes; *(ii)* o agente *FASTA Validator*, encarregado da verificação e correção de arquivos FASTA; *(iii)* o agente *MSA*, responsável pelo alinhamento múltiplo de sequências; *(iv)* o agente *Tree Constructor*, que realiza a construção de árvores filogenéticas; e *(v)* o agente *Tree Analyst*, responsável pela identificação de subárvores frequentes, englobando as três últimas etapas do *workflow* original. A execução do *workflow* é iniciada a partir de um conjunto de arquivos FASTA, contendo sequências de DNA, RNA e aminoácidos.

**Configuração do Experimento.** Os experimentos foram executados em um servidor local com processador Apple M2 e 16 GB de RAM. O *workflow* agêntico foi implementado no *framework* BeeAI, responsável pela orquestração dos agentes e execução das tarefas, com instrumentação nativa baseada em *OpenInference* para coleta de dados de execução. Os agentes utilizam o modelo Granite 3.3 (8B), executado localmente via *Ollama*, viabilizando a independência de serviços externos. Adicionalmente, foi utilizado um servidor baseado no *Model Context Protocol* (MCP), responsável por disponibilizar ferramentas aos agentes durante a execução. Todas as alternativas de programas do *workflow* original foram expostas aos agentes. Os dados de entrada consistem em sequências do vírus Zika, disponibilizadas por [Zadra et al. 2024]. O *prompt* dos agentes foi definido para permitir a seleção adequada de ferramentas na execução do *workflow* agêntico, conforme exemplificado a seguir:

> ***Prompt* do agente *Tree Analyst*.** *"You are responsible for analyzing phylogenetic tree ensembles and identifying frequent subtrees. You will receive a path containing previously generated phylogenetic trees. Use the available tool to extract frequent subtrees. Return the path containing the analysis results. Do not proceed if tree data is not available. Do not create directories or assume paths. Only use paths provided or returned by tools".*

**Análise do *Overhead*.** Para analisar o *overhead* associado à captura de dados de proveniência, foram realizadas 15 execuções do *workflow* agêntico. Na configuração *baseline*, sem captura de proveniência, o *workflow* apresentou tempo médio de execução de 169,456 s e desvio padrão de 36,320 s. Com a `ANCHOR`, observou-se um tempo médio de 172,785 s e um desvio padrão de 18,598 s. Os resultados indicam um *overhead* médio de aproximadamente 2% no tempo de execução. De modo geral, a incorporação da `ANCHOR` não impacta significativamente o desempenho do *workflow*, ao mesmo tempo em que adiciona mecanismos para prover auditabilidade e interpretação com base nos dados de proveniência. Essas características são especialmente relevantes em cenários não-determinísticos, como *workflows* agênticos, nos quais a rastreabilidade das decisões é essencial para análise, depuração e reprodutibilidade.

**Consultas.** A segunda etapa de avaliação da `ANCHOR` concentra-se na análise de sua capacidade de responder a consultas de proveniência relevantes para auditoria, interpretação e compreensão do comportamento de *workflows* agênticos. Esse tipo de análise é fundamental para garantir a rastreabilidade das interações, das decisões e dos dados produzidos ao longo da execução. As consultas consideradas nos experimentos são apresentadas na Tabela 1 e foram inspiradas no *First Provenance Challenge*[2], amplamente utilizado como referência para avaliação de sistemas de proveniência. No contexto deste trabalho, as consultas foram adaptadas para refletir características específicas de *workflows* agênticos, como a delegação de tarefas, a seleção dinâmica de ferramentas e a coordenação entre agentes. Para fins de

[2]https://openprovenance.org/provenance-challenge/FirstProvenanceChallenge.html

organização e análise, as consultas foram classificadas nas três categorias, de acordo com seu objetivo: *(i) p-prov*; *(ii) r-prov*; e *(iii) a-prov*.

**Tabela 1. Consultas de proveniência em *workflows* agênticos.**

| ID | Consulta | Tipo |
|---|---|---|
| Q1 | Qual é a cadeia de interações e decisões que levou à execução da tarefa de validação de sequências e quais agentes participaram de cada interação? | a-prov + r-prov |
| Q2 | Qual agente tem maior probabilidade de ser selecionado como primeira delegação pelo agente *Orchestrator*? | a-prov + p-prov |
| Q3 | Dada a indisponibilidade de uma ferramenta de validação de sequências para a reexecução de um *workflow*, qual é a cadeia hierárquica de agentes que pode ser acionada para selecionar uma nova *capability*, considerando a hierarquia entre os agentes? | a-prov + r-prov |

**Discussão dos Resultados das Consultas.** Com o objetivo de avaliar a viabilidade da ANCHOR, executaram-se todas as consultas apresentadas na Tabela 1, cujos resultados são analisados a seguir. A consulta Q1 visa reconstruir a cadeia causal das interações associadas à execução de uma tarefa de um *workflow*. Para isso, parte-se das interações identificadas pelo atributo *related_task_exec_id* e percorre-se recursivamente a relação de causalidade definida pelo atributo *caused_by_interaction_id*, recuperando as interações antecessoras que contribuíram para o estado final do *workflow*. Cada interação é então enriquecida com informações sobre as entidades envolvidas e, quando disponível, com a intenção do agente responsável, permitindo não apenas identificar a sequência de eventos, mas também compreender o papel de cada entidade e seus objetivos ao longo da cadeia de decisões. A consulta Q1 pode ser implementada em SQL, conforme exemplificado na Figura 5a.

Uma vez submetida a consulta Q1, tendo como parâmetro a tarefa de validação de sequências no *workflow*, o resultado evidencia o fluxo completo de interações desde o início da execução até a etapa de análise. O processo inicia com uma interação do tipo *message* do usuário para o sistema, seguida da delegação da requisição ao agente orquestrador. Este realiza uma *model invocation* para interpretar a solicitação e, em seguida, uma *tool invocation* de raciocínio (*think*), gerando uma intenção estruturada que decompõe a tarefa em etapas como validação, alinhamento e análise. Com base nessa intenção, o orquestrador executa nova *model invocation* para decidir a delegação da próxima etapa ao agente validador, caracterizando um ciclo deliberativo de raciocínio e decisão. O agente validador, por sua vez, realiza uma *model invocation* e, em seguida, uma *tool invocation* específica para validação de sequências, correspondente à tarefa definida na consulta. De forma geral, o resultado da consulta apresentado na Figura 6 revela um padrão de orquestração hierárquica que combina raciocínio baseado em modelos e execução por meio de ferramentas, resultando em um *workflow* modular e interpretável.

A consulta Q2 considera as interações de delegação iniciadas pelo agente *Orchestrator* ao longo das execuções do *workflow*, identificando, em cada execução, a primeira delegação realizada com base na ordenação temporal das interações. A partir desse conjunto, agregam-se os destinos selecionados como primeiro ponto de delegação, o que permite estimar a frequência com que cada agente é escolhido nessa posição. Esse processo permite inferir a probabilidade de cada agente ser o primeiro destino de delegação. Um exemplo de consulta SQL que implementa essa lógica é apresentado na Figura 5b.

```
WITH RECURSIVE interaction_lineage AS (
    SELECT
        i.interaction_id, i.type, i.source_entity_id,
        i.target_entity_id, i.based_on_intention_id,
        i.caused_by_interaction_id, i.timestamp,
        i.related_task_exec_id
    FROM interactions i
    WHERE i.related_task_exec_id = 'd69a91e8-*'
    UNION ALL
    SELECT
        i2.interaction_id, i2.type, i2.source_entity_id,
        i2.target_entity_id, i2.based_on_intention_id,
        i2.caused_by_interaction_id, i2.timestamp,
        i2.related_task_exec_id
    FROM interactions i2
    JOIN interaction_lineage il
      ON il.caused_by_interaction_id = i2.interaction_id
)
SELECT
    il.type AS interaction_type, src.name AS source_entity,
    tgt.name AS target_entity, intent.goal AS intention_goal,
    il.related_task_exec_id AS task_exec_id
FROM interaction_lineage il
LEFT JOIN entities src ON il.source_entity_id = src.entity_id
LEFT JOIN entities tgt ON il.target_entity_id = tgt.entity_id
LEFT JOIN intentions intent ON
                  il.based_on_intention_id = intent.intention_id
ORDER BY il.timestamp;
```

**(a) Q1**

```
WITH first_delegations AS (
    SELECT
        i.workflow_exec_id, i.target_entity_id,
        ROW_NUMBER() OVER (
            PARTITION BY i.workflow_exec_id
                         ORDER BY i.timestamp
        ) AS rn
    FROM interactions i
    WHERE
        i.source_entity_id = 'efa07de6-*'
        AND i.type = 'delegation'
        AND i.target_entity_id IS NOT NULL
)
SELECT
    e.name AS target_agent_name, e.type AS target_agent_type,
    COUNT(*) AS workflow_count
FROM first_delegations fd
JOIN entities e
  ON fd.target_entity_id = e.entity_id WHERE fd.rn = 1
GROUP BY e.entity_id, e.name, e.type
ORDER BY workflow_count DESC;
```

**(b) Q2**

```
WITH RECURSIVE escalation_chain AS (
    SELECT e.entity_id, e.name,e.type, e.supervisor_id, 0 AS level
        FROM capabilities c JOIN entities e ON c.owner_entity_id = e.entity_id WHERE c.target_entity_id = '3ded6677-*'
    UNION ALL
    SELECT sup.entity_id, sup.name, sup.type, sup.supervisor_id, ec.level + 1
        FROM escalation_chain ec JOIN entities sup ON ec.supervisor_id = sup.entity_id WHERE ec.supervisor_id IS NOT NULL
) SELECT DISTINCT entity_id, name, type, level FROM escalation_chain ORDER BY level ASC;
```

**(c) Q3**

**Figura 5. SQL para consultas Q1, Q2 e Q3.**

| interaction_type | source_entity | target_entity | intention_goal | task_exec_id |
|---|---|---|---|---|
| message | Workflow User | OpenInference System | | |
| delegation | OpenInference System | orchestrator_agent | | |
| model_invocation | orchestrator_agent | | | |
| tool_invocation | orchestrator_agent | think | To complete this task, I need to coordinate a phylogenetic analysis workflow. The steps are: validation, alignment, tree construction, and subtree analysis. | 0a3faa82-* |
| model_invocation | orchestrator_agent | | | |
| delegation | orchestrator_agent | validation_agent | | |
| model_invocation | validation_agent | | | |
| tool_invocation | validation_agent | sequence_validator | | d69a91e8-* |

**Figura 6. Resultado da Consulta Q1**

Uma vez submetida a consulta Q2 e fixando como parâmetro o agente *Orchestrator* do *workflow*, parametrizado como agente orquestrador, temos o resultado apresentado na Figura 7. Podemos perceber que o agente *validation_agent* é o único destino da primeira delegação, com frequência igual a 1 nas execuções observadas. No entanto, esse resultado está diretamente relacionado à composição da base analisada, que contém apenas um tipo de *workflow*, no qual a validação é sempre a primeira a ser executada. Dessa forma,

a distribuição observada não reflete necessariamente uma preferência geral do agente orquestrador, mas sim uma característica estrutural do *workflow* considerado. Ainda assim, a consulta demonstra a capacidade de identificar e quantificar padrões de delegação inicial, o que se torna mais informativo em cenários com maior diversidade de execuções.

| target_agent_name | target_agent_type | workflow_count |
|---|---|---|
| validation_agent | agent | 1 |

**Figura 7. Resultado da Consulta Q2**

Em *workflows* multiagentes, diferentes agentes podem ser responsáveis por capacidades específicas. Quando uma determinada ferramenta se torna indisponível, pode-se inicialmente consultar o agente associado a essa capacidade para identificar uma alternativa; caso não seja possível, a decisão é escalada ao seu supervisor, sucessivamente, até encontrar um agente capaz de resolver o problema. A consulta `Q3` identifica o agente associado à capacidade na respectiva tabela e percorre recursivamente a hierarquia definida na tabela de entidades, por meio de *supervisor_id*, construindo a cadeia hierárquica de agentes. Esse processo permite determinar quais agentes podem ser acionados em cenários de indisponibilidade. Um exemplo de consulta SQL que implementa essa lógica é apresentado na Figura 5c.

| entity_id | name | type | level |
|---|---|---|---|
| b1129289-d046-529a-93b1-f1d43ad8ea22 | validation_agent | agent | 0 |
| efa07de6-c929-532c-b3fe-a8ac377cde40 | orchestrator_agent | agent | 1 |

**Figura 8. Resultado da Consulta Q3**

Considerando a consulta `Q3` com a ferramenta de validação de sequências como parâmetro, nota-se, na Figura 8, que o resultado explicita a cadeia hierárquica de agentes associada à capacidade representada pela ferramenta, iniciando pelo agente diretamente responsável (*level 0*) e progredindo recursivamente até seus supervisores. Essa sequência evidencia o mecanismo de escalonamento definido no *workflow*, no qual, diante da indisponibilidade da ferramenta, agentes de níveis superiores da hierarquia podem ser acionados como alternativas para a execução da tarefa. Dessa forma, o resultado não apenas identifica o agente primariamente associado à capacidade, mas também revela a estrutura organizacional que sustenta a substituição progressiva de responsabilidades, o que permite maior robustez e resiliência na execução do sistema.

Essas consultas demonstram que a `ANCHOR` é capaz de responder a consultas de proveniência que envolvem *p-prov*, *r-prov* e *a-prov*. Além disso, a `ANCHOR` permite exportar o grafo de proveniência no padrão W3C PROV. A Figura 9 apresenta o grafo exportado para o *workflow* de busca por subárvores frequentes em bancos de dados filogenéticos. Devido à complexidade e ao tamanho do grafo, apenas um subconjunto dos elementos é exibido, omitindo, por exemplo, chamadas a LLMs e outros detalhes de baixo nível. A execução é iniciada a partir de um *prompt* fornecido pelo agente usuário ao agente sistema, o que dá origem ao *workflow*. Em seguida, o agente sistema delega a requisição ao agente orquestrador, responsável por decompor o problema em um conjunto de tarefas, definir um plano de execução e coordenar a invocação das ferramentas necessárias. O processo culmina na chamada da ferramenta *final_answer*, responsável por gerar a resposta final do *workflow*,

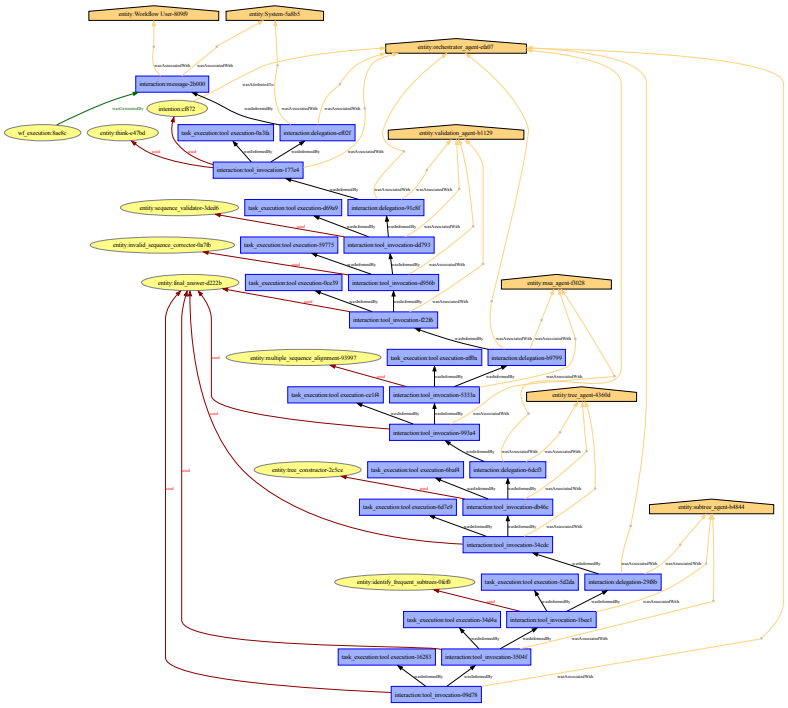

**Figura 9. Grafo de proveniência do *workflow* agêntico.**

que é então retornada ao usuário. Por fim, destaca-se o caráter *interaction-centric* da abordagem, que facilita a compreensão de *workflows* agênticos ao explicitar, além da execução de ferramentas, as interações e as decisões entre os agentes.

## 5. Conclusões

Este artigo apresentou a ANCHOR, uma abordagem agnóstica a *framework* agêntico para a captura e a representação de dados de proveniência em *workflows* agênticos. Como principal contribuição, destaca-se o suporte explícito a aspectos como a delegação, a negociação e a hierarquia entre agentes, ainda pouco explorados na literatura. A abordagem adota uma perspectiva *interaction-centric*, que facilita a compreensão e o rastreamento de execuções por meio do encadeamento de interações, evidenciando entidades, tarefas derivadas e intenções associadas às decisões. Os resultados experimentais indicam um *overhead* introduzido pela ANCHOR de aproximadamente 2%, considerado desprezível no cenário avaliado. As consultas SQL demonstram que o modelo apoia tanto análises de proveniência comuns quanto consultas que exploram a dimensão multiagente. Adicionalmente, foi demonstrado o suporte à exportação para o padrão W3C PROV em um caso de uso real. Como trabalhos futuros, pretende-se avaliar a abordagem em diferentes configurações de *workflows* agênticos e investigar a integração com múltiplos *frameworks* além do BeeAI. Também se vislumbra o uso da proveniência como suporte ao planejamento automatizado de *workflows*, permitindo o reúso de conhecimento previamente validado e a recomendação de execuções relevantes.

## Agradecimentos

Os autores agradecem à CAPES, ao CNPq e à FAPERJ pelo apoio parcial à realização deste trabalho. Adicionalmente, os autores declaram, de forma transparente, que ferramentas baseadas em LLMs foram utilizadas exclusivamente para revisão textual.

## Referências

Agno (2026). The runtime for agentic software. `https://www.agno.ai`. Acessado em Março de 2026.

Arize AI (2024). Openinference: Open standard for llm observability. `https://github.com/Arize-ai/openinference`. Accessed: 2026-03-25.

Babuji, Y., Woodard, A., et al. (2019). Scalable parallel programming in python with parsl. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, PEARC '19, pages 22:1–22:8. ACM.

BeeAI (2026). Agentic workflow framework. `https://framework.beeai.dev/`. Acessado em Março de 2026.

Chang, C.-C. and Echizen, I. (2025). The chronicles of foundation ai for forensics of multi-agent provenance. *arXiv preprint arXiv:2504.12612*.

Cloud Native Computing Foundation (2024). Opentelemetry: Observability framework for cloud-native software. `https://opentelemetry.io`. Accessed: 2026-03-25.

CrewAI (2026). The leading multi-agent platform. `https://crewai.com/`. Acessado em Março de 2026.

Davis, D. B., Featherston, J., Fukuda, M., and Asuncion, H. U. (2017). Data provenance for multi-agent models. In *2017 IEEE 13th International Conference on e-Science (e-Science)*, pages 39–48.

de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Deelman, E., Ferreira da Silva, R., Vahi, K., Rynge, M., Mayani, R., Tanaka, R., Whitcup, W., and Livny, M. (2021). The pegasus workflow management system: Translational computer science in practice. *Journal of Computational Science*, 52:101200. Funding Acknowledgments: NSF 1664162.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Comput. Sci. Eng.*, 10(3):11–21.

Friedman, S., Rye, J., LaVergne, D., Thomsen, D., Allen, M., and Tunis, K. (2020). Provenance-based interpretation of multi-agent information analysis.

Gómez-Carmona, O., Casado-Mansilla, D., de Ipiña, D. L., and García-Zubia, J. (2024). Human-in-the-loop machine learning: Reconceptualizing the role of the user in interactive approaches. *Internet of Things*, 25:101048.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu, T. (2025). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2).

Missier, P., Belhajjame, K., and Cheney, J. (2013). The w3c prov family of specifications for modelling provenance metadata. In *Proceedings of the 16th international conference on extending database technology*, pages 773–776.

Moraes, J. V., Ferrari, C., Rosseti, I., and de Oliveira, D. (2025). Exploring evolutionary patterns: A jupyter notebook for discovering frequent subtrees in phylogenetic tree databases. *Journal of Information and Data Management*, 16(1):232–239.

Pimentel, J. a. F., Murta, L., Braganholo, V., and Freire, J. (2017). noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts. *Proc. VLDB Endow.*, 10(12):1841–1844.

Pinhanez, C., Cavalin, P., Sanctos, C., Grave, M., and Primerano, Y. (2025). The non-determinism of small llms: Evidence of low answer consistency in repetition trials of standard multiple-choice benchmarks.

Plötzky, F., Britz, K., and Balke, W.-T. (2025). A conceptual model for attributions in event-centric knowledge graphs. *Data Knowledge Engineering*, 159:102449.

Schick, T., Dwivedi-Yu, J., Dessí, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Silva, V., de Oliveira, D., Valduriez, P., and Mattoso, M. (2018). Dfanalyzer: runtime dataflow analysis of scientific applications using provenance. *Proc. VLDB Endow.*, 11(12):2082–2085.

Sirqueira, T. F. M., Viana, M. L., Cunha, F. J. P. D., Nunes, I., and Lucena, C. J. P. D. (2018). Data provenance in multi-agent systems: relevance, benefits and research opportunities. *International Journal of Metadata, Semantics and Ontologies*, 13(1):9–19.

Souza, R., Gueroudji, A., DeWitt, S., Rosendo, D., Ghosal, T., Ross, R., Balaprakash, P., and Da Silva, R. F. (2025). Prov-agent: Unified provenance for tracking ai agent interactions in agentic workflows. In *2025 IEEE International Conference on eScience (eScience)*, pages 467–473. IEEE.

Souza, R., Skluzacek, T. J., Wilkinson, S. R., Ziatdinov, M., and da Silva, R. F. (2023). Towards lightweight data integration using multi-workflow provenance and data observability. In *IEEE International Conference on e-Science*.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Zadra, N., Rizzoli, A., and Rota-Stabelli, O. (2024). Comprehensive phylogenomic analysis of zika virus: Insights into its origin, past evolutionary dynamics, and global spread. *Virus Research*, 350:199490.

Zhao, W. X. et al. (2023). A survey of large language models. *arXiv:2303.18223*, 1(2).