Twinscie-Prov: Gerenciando a Proveniência sobre o Ciclo-de-vida de ML no Sistema Twinscie

Júlia Neumann Bastos[1], Fabio Porto[1], Fábio Levy Siqueira[2], Edson Gomi[2], Ismael Santos[3], Rodrigo Barreira[3], Isabela Siqueira[3], Eduardo Ogasawara[4]

[1]Laboratório Nacional de Computação Científica (LNCC) – Petrópolis, RJ – Brazil

[2]Escola Politécnica da Universidade de São Paulo (Poli-USP) – SP – Brazil

[3]Petrobras – RJ – Brazil

[4]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca – RJ – Brazil

julia@posgrad.lncc.br, fporto@lncc.br, {levy.siqueira, gomi}@usp.br,

{ismaelh, barreira, isabela.siqueira}@petrobras.com.br,

eogasawara@ieee.org

Abstract. *The increasing complexity of machine learning (ML) applications necessitates systems that ensure traceability and reproducibility. This work introduces the Twinscie-Prov approach, an adaptation of the W3C PROV standard, to structure data and process provenance throughout the ML lifecycle within the Twinscie system. Provenance data is stored in the Neo4j NoSQL database, enabling complex queries and auditing. Preliminary studies indicate that, for queries involving navigation through dependency graphs, typical in provenance data, the Neo4j implementation is up to five orders of magnitude faster than log-based approaches.*

Resumo. *O aumento da complexidade em aplicações de aprendizado de máquina exige sistemas que garantam rastreabilidade e reprodutibilidade. Este trabalho apresenta a abordagem Twinscie-Prov, uma adequação do padrão W3C PROV para estruturar a proveniência de dados e processos ao longo do ciclo de vida de Machine Learning (ML) no sistema Twinscie. Os dados de proveniência são armazenados no sistema NoSQL Neo4j, permitindo consultas complexas e auditoria. Estudos preliminares mostram que, para consultas envolvendo navegação no grafo de dependências, típicas em dados de proveniência, a implementação no Neo4j é até cinco ordens de grandeza mais rápida que a baseada em logs.*

1. Introdução

A aplicação de modelos de *Machine Learning* (ML) em ambientes industriais e científicos complexos impulsiona o desenvolvimento de sistemas que apoiem todo o seu ciclo de vida, da ingestão de dados à operação em produção e reavaliação periódica. Esses sistemas devem lidar com múltiplos fluxos de dados, execuções encadeadas, versões de modelos e condições operacionais variáveis. Nesse contexto, rastreabilidade, reprodutibilidade e governança tornam-se essenciais, especialmente quando decisões automatizadas impactam processos críticos [Polyzotis et al. 2017].

O Twinscie [de Almeida et al. 2024] é um sistema projetado para apoiar o ciclo de vida de modelos de ML, incorporando os princípios de gêmeos digitais para representar sistemas reais de forma contínua e inteligente. Ele integra dados, algoritmos e modelos por meio de fluxos reutilizáveis e parametrizáveis, abrangendo desde o processamento até a inferência contínua e a tomada de decisão automatizada. Suporta diferentes estratégias de construção e operação de modelos, incluindo abordagens supervisionadas, não supervisionadas, importação de modelos externos e formação de comitês, além de gerenciar metadados e automatizar tarefas analíticas.

A estruturação de dados e processos no Twinscie gera naturalmente informações de proveniência, documentação das origens, transformações e usos de entidades computacionais ao longo do tempo, como *datasets*, modelos de ML, *dataflows*, funções e ambientes computacionais. A gestão sistemática desses dados favorece a reprodutibilidade, auditabilidade e transparência em aplicações baseadas em ML.

Entre os modelos conceituais voltados à representação de proveniência, o W3C PROV [Moreau and Groth 2013] destaca-se por sua flexibilidade e padronização. Ele descreve como entidades, atividades e agentes se relacionam na geração, modificação e uso de artefatos computacionais, possibilitando múltiplos níveis de detalhamento. Apesar de amplamente adotado em *workflows* científicos, sua aplicação em sistemas de ML ainda enfrenta desafios práticos, como a adaptação a fluxos de trabalho complexos e dinâmicos.

Muitas ferramentas que apoiam o ciclo de vida de modelos ainda carecem de mecanismos padronizados e completos para capturar e consultar dados de proveniência, principalmente em fluxos com múltiplas etapas. Essa lacuna compromete a reprodutibilidade e dificulta a auditoria dos processos, limitando a confiabilidade de sistemas de decisão automatizados. Torna-se necessário, portanto, adotar abordagens que combinem padronização conceitual com suporte técnico a fluxos dinâmicos, assegurando rastreabilidade de ponta a ponta.

Este trabalho propõe a abordagem Twinscie-Prov, baseada no modelo W3C PROV, para estruturar a proveniência das operações realizadas pelo sistema Twinscie. A proposta inclui o mapeamento completo do banco de dados do sistema, modelando entidades, atividades e agentes conforme o W3C PROV, e a formulação de consultas conceituais para rastrear fluxos de trabalho. Também são apresentados experimentos que avaliam o impacto da coleta de proveniência utilizando o banco de dados grafo Neo4j [Neo4j 2003], considerando o tempo de gravação dos dados e a eficiência das consultas em cenários com e sem rastreabilidade. Os resultados demonstram o custo-benefício da adoção de uma base conceitual padronizada para garantir maior transparência e controle em sistemas de ML.

O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta os principais conceitos que fundamentam este trabalho, incluindo o de gêmeos digitais e a arquitetura do sistema Twinscie; a Seção 3 revisa os principais trabalhos relacionados à gerência de proveniência em ML; a Seção 4 descreve a metodologia adotada, seguida pela Seção 5, que apresenta um caso de uso aplicado; a Seção 6 discute os resultados experimentais; e, por fim, a Seção 7 apresenta as conclusões e perspectivas futuras.

2. Contextualização e Fundamentos

Esta seção apresenta os conceitos e tecnologias que sustentam a abordagem proposta. São discutidos o paradigma de gêmeos digitais, que inspira a modelagem e o rastreamento de execuções computacionais dinâmicas, e a arquitetura do sistema responsável por coordenar fluxos de dados e registrar sua proveniência. Essa contextualização estabelece as bases para compreender os experimentos e as contribuições deste trabalho.

2.1. Gêmeos Digitais

Gêmeos Digitais (*Digital Twins*) são representações digitais de entidades físicas ou processos computacionais, capazes de refletir seu estado em tempo real e evoluir com novas interações e mudanças [Grieves 2014]. Inicialmente populares na indústria, vêm sendo adotados também em ciência de dados e aprendizado de máquina, por sua capacidade de representar o ciclo de vida completo de modelos e fluxos computacionais.

Na ciência de dados, sua adoção impõe desafios específicos à gestão da proveniência. Representar modelos e pipelines como entidades em constante evolução exige monitoramento contínuo das transformações, execuções e decisões ao longo do tempo. Isso demanda mecanismos que registrem não apenas os resultados, mas também a estrutura, a sequência e a lógica das execuções.

Nesse contexto, a proveniência deve integrar registros retrospectivos (execuções passadas) e prospectivos (configurações planejadas e estrutura dos fluxos). Também se intensifica a necessidade de interoperabilidade entre componentes heterogêneos, como sistemas de versionamento de modelos, ferramentas de captura automática e fluxos declarados pelo usuário.

2.2. Arquitetura do Sistema Twinscie

A arquitetura do Twinscie foi concebida para coordenar serviços computacionais de forma integrada e registrar, de maneira estruturada, informações de proveniência. A Figura 1 apresenta uma visão geral da arquitetura, destacando os elementos mais relevantes à execução dos serviços e à coleta de proveniência. No núcleo do sistema, um módulo central orquestra solicitações e interage com um catálogo de metadados que descreve os artefatos computacionais.

Os serviços no Twinscie são modelados como *dataflows*, organizados em grafos de funções com dependência de dados, estabelecendo relações de produtor e consumidor. Cada função representa uma etapa de transformação ou treinamento; funções de aprendizado são classificadas como *learners*. As execuções podem ocorrer em diferentes plataformas, previamente registradas no catálogo, como clusters Apache Spark, serviços da AWS ou supercomputadores como o Santos Dumont [LNCC 2015].

Para apoiar a execução de serviços de aprendizado de máquina, o sistema integra o *framework* MLflow, garantindo interoperabilidade com bibliotecas como PyTorch, TensorFlow e scikit-learn. O MLflow Server também captura automaticamente dados de proveniência, conforme discutido na Seção 4.4.

Durante a execução dos fluxos, o sistema armazena artefatos produzidos, como dados transformados, modelos treinados e inferências, que podem ser recuperados via

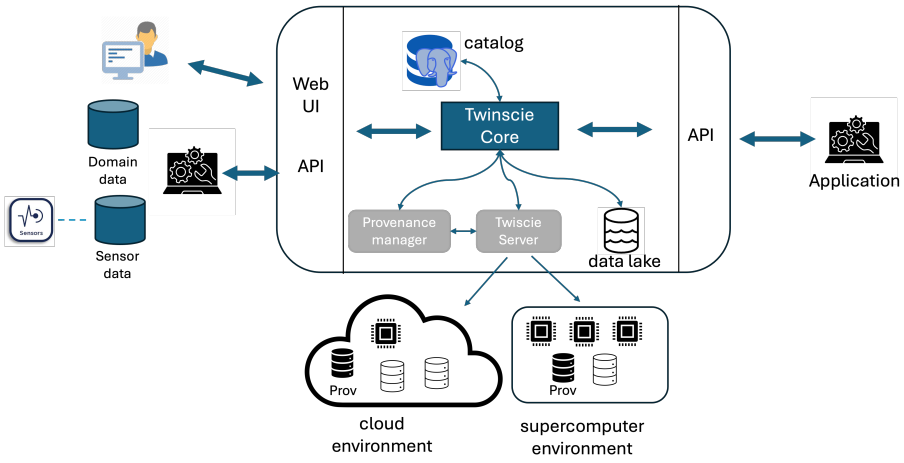

Figura 1. Visão geral do sistema Twinscie

API por aplicações consumidoras. Essas aplicações realizam o *download* dos resultados e os integram aos seus processos operacionais.

Com suporte completo ao ciclo de vida de modelos de ML, da preparação de dados à inferência, o Twinscie oferece uma infraestrutura robusta para a introdução de inteligência preditiva em aplicações reais, promovendo automação, rastreabilidade e reutilização de fluxos analíticos.

3. Trabalhos Relacionados

A proveniência tem ganhado destaque em sistemas de aprendizado de máquina (ML), especialmente em contextos que demandam reprodutibilidade, auditoria e explicabilidade. O modelo W3C PROV [Moreau and Groth 2013], amplamente adotado, define uma estrutura com entidades, atividades e agentes, permitindo representar tanto execuções passadas quanto fluxos planejados.

Diversos trabalhos já exploraram o uso do W3C PROV em ML. O DLProv [Pina et al. 2024], por exemplo, representa o treinamento de modelos, mas não cobre o ciclo completo do aprendizado. Outras soluções, como [Schelter et al. 2017], automatizam o rastreamento de experimentos, mas sem adotar um modelo conceitual formal. Ferramentas como o MLflow se limitam ao histórico de experimentos, e mesmo iniciativas como o MLflow2PROV [Schlegel et al. 2023] não abrangem etapas como ingestão ou configurações ambientais.

Diferentemente dessas abordagens, o Twinscie-Prov não se limita a transformar logs do MLflow em um grafo PROV. Sua proposta está na capacidade de capturar a proveniência de forma mais abrangente e granular, integrando informações estruturais dos fluxos (proveniência prospectiva) com os resultados das execuções (proveniência retrospectiva). Enquanto o MLflow2PROV se concentra predominantemente no histórico dos modelos e parâmetros registrados, o Twinscie-Prov é capaz de representar formalmente os *dataflows*, funções, *datasets*, parâmetros e *learners*, além de registrar automaticamente a geração de artefatos, métricas de treinamento e estatísticas intermediárias. Isso permite consultas mais ricas e detalhadas, incluindo informações de estado e transformação ao longo da execução, algo que ferramentas como o MLflow2PROV não oferecem.

Apesar dos avanços, ainda há uma lacuna quanto à captura completa e integrada da proveniência ao longo do ciclo de vida de ML. Nesse cenário, modelos formais como o W3C PROV aplicados a fluxos dinâmicos surgem como alternativa promissora para garantir rastreabilidade ampla e reutilizável em aplicações reais.

4. Metodologia

Esta seção descreve a abordagem Twinscie-Prov adotada para estruturar e consultar a proveniência no sistema Twinscie. Inicialmente, são apresentadas as funcionalidades analisadas e os tipos de informações relevantes para a rastreabilidade. Em seguida, detalham-se os mecanismos de captura da proveniência, a modelagem segundo o padrão W3C PROV e sua implementação em banco de dados orientado a grafos.

4.1. Modelagem de Proveniência

O Twinscie oferece funcionalidades baseadas em *dataflows*, que coordenam operações como seleção de aprendizes, preparação de conjuntos de dados, treinamento de modelos, inferência e execução de algoritmos parametrizados. Essas operações manipulam entidades computacionais, como *datasets*, *learners* e modelos, que são transformadas e reutilizadas ao longo do ciclo de vida. As informações geradas nesses processos constituem metadados de proveniência, essenciais para rastrear a origem, o uso e a transformação dos artefatos envolvidos.

A Figura 2 apresenta o esquema do grafo de proveniência do Twinscie-Prov, conforme a semântica do W3C PROV. O grafo é composto por informações do catálogo do Twinscie, além de dados específicos de proveniência (ver Seção 4.4). Ele representa entidades, parâmetros, operadores e execuções ao longo do fluxo de ML, com foco nos elementos mais relevantes para esta seção.

Os nós do grafo são coloridos conforme sua natureza: amarelo representa entidades (como hiperparâmetros e *learners*); azul indica atividades (como execuções de *dataflows*, processamento e treinamento); e laranja identifica agentes (como ambientes de execução). As arestas representam relações como *used*, *wasGeneratedBy* e *wasAssociatedWith*, evidenciando a trajetória dos artefatos computacionais ao longo do ciclo de vida do modelo. A organização visual permite observar claramente os encadeamentos de etapas e a propagação dos artefatos, tornando a rastreabilidade explícita e facilitando auditorias sobre o histórico de decisões computacionais.

4.2. Consultas para Rastreabilidade e Análise de Proveniência

A partir das funcionalidades do Twinscie, foram identificadas situações que exigem rastreabilidade refinada, como estratégias distintas de construção de modelos, incluindo treinamento local, importação de modelos externos e formação de comitês. Cada uma dessas abordagens envolve relações específicas entre entidades e atividades, que precisam ser corretamente representadas no modelo de proveniência.

Para guiar essa modelagem, foi elaborado um conjunto de vinte perguntas representativas, organizadas por tipo de informação: nível de entidade (atributos de *datasets*, modelos ou funções) e nível de relacionamento (derivações, transformações e encadeamentos de execução). A Figura 3 apresenta seis dessas perguntas, escolhidas por sua diversidade e relevância, abrangendo desde configurações de hiperparâmetros até

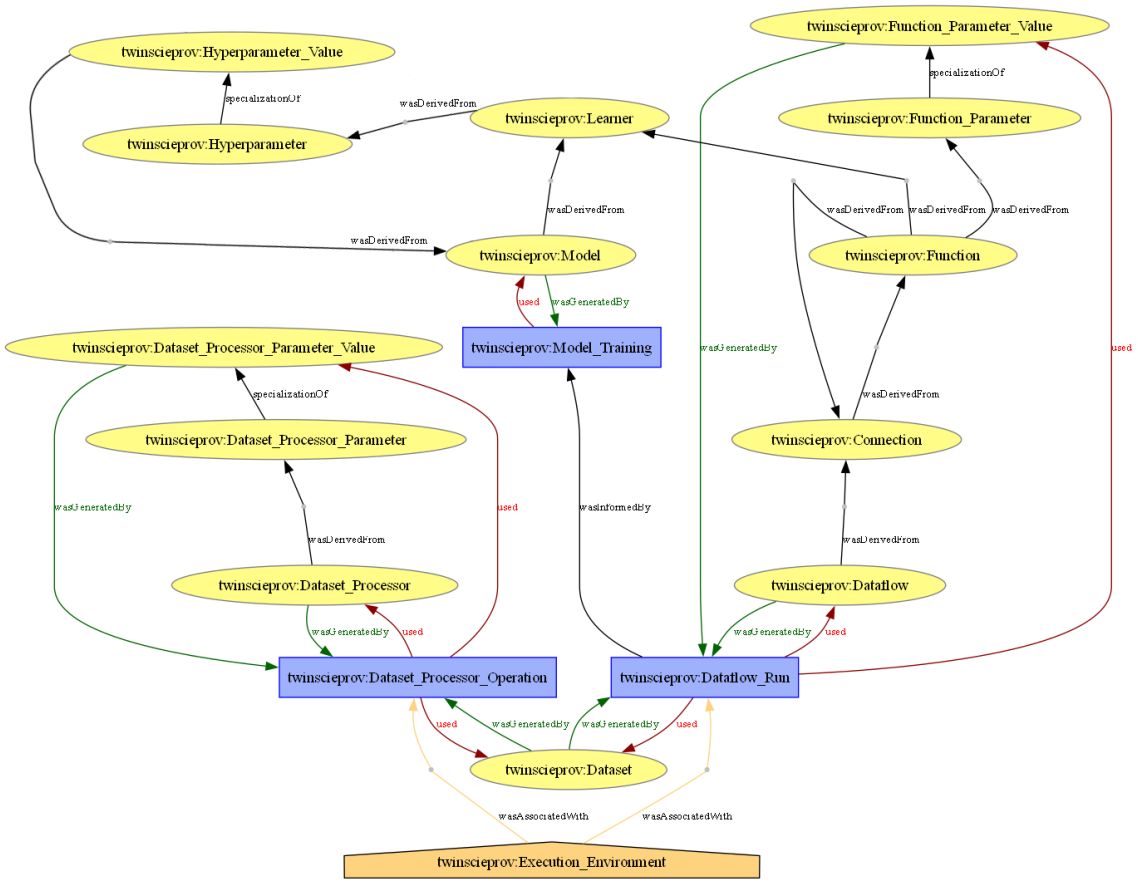

Figura 2. Grafo de proveniência prospectiva referente aos artefatos do Twinscie.

transformações de dados e impacto de decisões analíticas. Cada pergunta foi utilizada para validar o modelo e estruturar consultas baseadas nos registros de proveniência.

ID	Consulta	Tipo
Q1	Quais hiperparâmetros foram utilizados para treinar um determinado modelo?	Entidade
Q2	Qual foi o ambiente computacional utilizado na execução de um determinado dataflow?	Entidade
Q3	Qual foi o tempo médio de processamento de um dataflow executado?	Entidade
Q4	Quais funções e seus parâmetros contribuíram para a execução de um dataflow especifico?	Relacionamento
Q5	Qual foi o pipeline completo de processamento de dados e treinamento para um modelo específico?	Relacionamento
Q6	Como é estabelecida a cadeia de proveniência que conecta os artefatos de treinamento ao modelo gerado?	Relacionamento

Figura 3. Conjunto de questões formuladas para orientar a modelagem e a captura de proveniência no Twinscie.

4.3. Modelagem dos Requisitos no Twinscie-Prov

Com base nas perguntas formuladas e nas funcionalidades do Twinscie, foi definido um conjunto de requisitos de rastreabilidade que orientaram a estruturação do modelo de proveniência adotado. Esses requisitos foram tratados por meio da modelagem explícita de entidades, atividades e agentes conforme os conceitos do W3C PROV, permitindo descrever com precisão as transformações e dependências de cada operação.

A representação no Twinscie-Prov, conforme o grafo da Figura 2, inclui entidades como `Dataset` e `Model`, associadas a atividades como `Model_Training`, `Dataset_Processor_Operation` e `Dataflow_Run`, que refletem os processos computacionais do sistema. As conexões são descritas pelas relações `used`, `wasGeneratedBy`, `wasDerivedFrom`, entre outras, expressando o fluxo de transformação de dados e a evolução dos modelos.

Essa modelagem permitiu capturar diferentes padrões de proveniência observados, como nas consultas Q1 e Q4. A Q1, por exemplo, envolve a atividade `Model_Training`, que conecta hiperparâmetros (`Hyperparameter`) e seus valores (`Hyperparameter_Value`) ao artefato de treinamento (`Learner`), permitindo identificar quais parâmetros foram utilizados, seus valores concretos e sua influência sobre o modelo final (`Model`).

A Q4 visa identificar funções e parâmetros que contribuíram para um `Dataflow_Run`. A entidade `Function` representa cada operação do fluxo, com parâmetros descritos em `Function_Parameter` e seus valores registrados como `Function_Parameter_Value`. Isso possibilita compreender quais operações foram aplicadas, quais configurações foram utilizadas e como impactaram os resultados.

A Listagem 1 mostra como a Q4 é representada no Twinscie-Prov segundo o modelo W3C PROV. O trecho exemplifica a criação das entidades e atividades e suas relações, evidenciando a estrutura de proveniência prospectiva ao descrever formalmente os elementos do *dataflow* e as dependências entre operações.

```
dataflow = prov_doc.entity('twinscieprov:Dataflow')
connection = prov_doc.entity('twinscieprov:Connection')
function = prov_doc.entity('twinscieprov:Function')
functionParam = prov_doc.entity('twinscieprov:Function_Parameter')
functionParamV = prov_doc.entity('twinscieprov:Function_Parameter_Value')
dataflowRun = prov_doc.activity('twinscieprov:Dataflow_Run')

prov_doc.wasDerivedFrom(dataflow, function)
prov_doc.wasDerivedFrom(function, functionParam)
prov_doc.specializationOf(functionParam, functionParamV)

prov_doc.wasDerivedFrom(dataflow, connection)
prov_doc.wasDerivedFrom(connection, function)
prov_doc.wasDerivedFrom(function, connection)

prov_doc.used(functionParamV, dataflowRun)
prov_doc.wasGeneratedBy(functionParamV, dataflowRun)

prov_doc.used(dataflowRun, dataflow)
prov_doc.wasGeneratedBy(dataflowRun, dataflow)
```

Listagem 1. Representação da proveniência de dataflow conforme a Consulta Q4

A representação dos requisitos no grafo de proveniência garante que cada pergunta formulada tenha respaldo nos elementos e nas relações registrados. Dessa forma, o Twinscie-Prov assegura a capacidade de responder a consultas retrospectivas sobre a origem, composição, transformação e avaliação de qualquer entidade computacional en-

volvida no ciclo de vida de modelos, promovendo uma rastreabilidade detalhada e uma auditoria completa.

Além disso, o Twinscie-Prov realiza automaticamente a coleta de proveniência retrospectiva, registrando os dados produzidos durante a execução dos fluxos. Essa coleta é feita por meio de *hooks*, como ilustrado na Listagem 2, que mostra um fragmento do processo de instrumentação. Durante a execução de cada nó do pipeline, são acionados os métodos `before_node_run` e `after_node_run`, responsáveis por registrar o início e o término da execução de cada componente.

```
from kedro.framework.hooks import hook_impl

@hook_impl
def before_node_run(self, node, catalog, inputs, is_async, session_id):
    self.provenance.register_start_node(node.name)

@hook_impl
def after_node_run(self, node, catalog, inputs, outputs, is_async,
    session_id):
    self.provenance.register_end_node(node.name)
```

Listagem 2. Instrumentação via hooks para coleta retrospectiva

Essa abordagem assegura que tanto os aspectos planejados de um fluxo (proveniência prospectiva) quanto os dados efetivamente gerados durante sua execução (proveniência retrospectiva) sejam registrados de forma integrada no Twinscie-Prov.

4.4. Captura de Proveniência

A captura de proveniência no Twinscie contempla tanto a proveniência prospectiva, estruturada no catálogo do sistema, quanto a retrospectiva, gerada durante a execução dos fluxos. No primeiro caso, são registrados metadados sobre *dataflows*, funções, *learners*, *datasets* e modelos. Na proveniência retrospectiva, distinguem-se três tipos de registros: (i) geração de artefatos, (ii) métricas de treinamento e (iii) registros definidos pelo usuário.

Para a geração de artefatos, o Twinscie utiliza o MLflow para recuperar os resultados produzidos e associá-los às funções responsáveis. As métricas de treinamento são capturadas automaticamente via `autolog`, com valores registrados a cada época e enviados ao servidor Twinscie para atualização no banco de dados de proveniência.

Registros definidos pelo usuário, como estatísticas de variáveis (e.g., média, desvio padrão), podem ser adicionados por meio de *hooks* configuráveis nos nós do *dataflow*. Esses registros são exportados em JSON, armazenados no MLflow e integrados posteriormente ao grafo de proveniência.

4.5. Implementação do Modelo de Proveniência em Neo4j

Para viabilizar a representação e consulta eficiente da proveniência capturada no Twinscie-Prov, optou-se por implementar o modelo baseado no padrão W3C PROV utilizando o banco de dados orientado a grafos Neo4j. A escolha pelo Neo4j se deu por sua capacidade nativa de representar relações complexas entre entidades, atividades e agentes, além de possibilitar a execução de consultas altamente expressivas por meio da linguagem

Cypher. Essa abordagem mostrou-se especialmente vantajosa para o contexto do Twinscie, no qual a rastreabilidade depende da análise de encadeamentos de transformações, derivações e composições entre elementos computacionais.

Cada entidade do modelo W3C PROV foi representada como um nó no grafo. As atividades também foram mapeadas como nós específicos, conectando-se a entidades e agentes por meio das relações estabelecidas pelo modelo. Esse mapeamento garante compatibilidade com o modelo conceitual W3C PROV, permitindo que o grafo gerado, ilustrado na Figura 4, mantenha a semântica original das relações de proveniência, ainda que algumas informações tenham sido propositalmente omitidas para fins de simplificação visual.

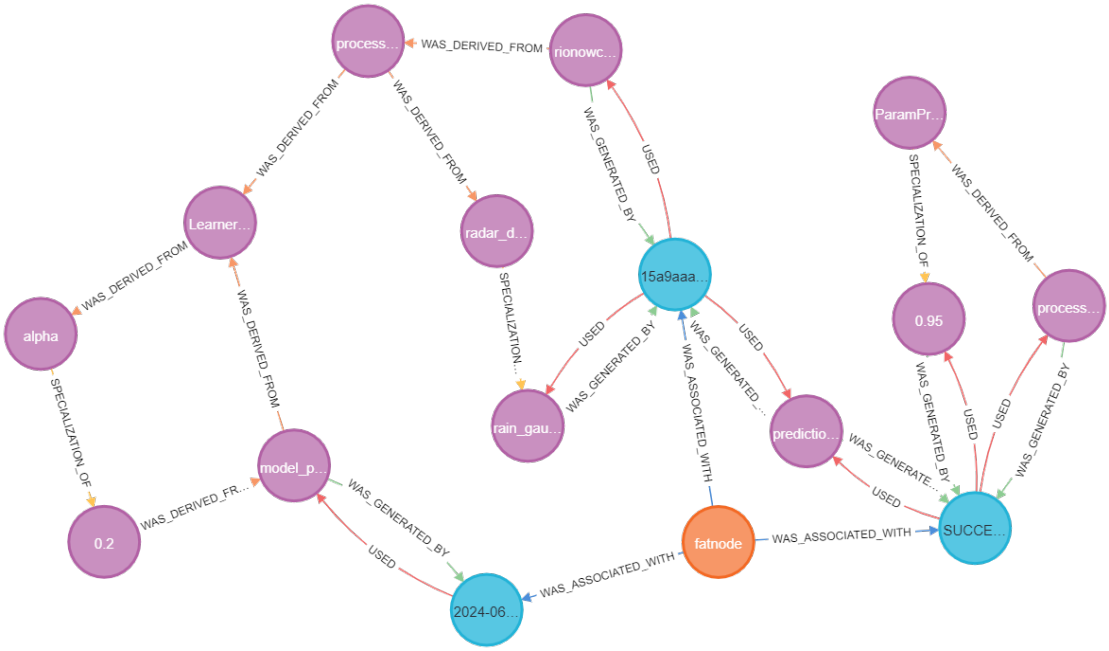

Figura 4. Grafo de proveniência armazenado no banco de dados Neo4j, representando entidades (em rosa), atividades (em azul) e agentes (em laranja) segundo o modelo W3C PROV.

Com a base de proveniência estruturada no Neo4j, tornou-se possível executar consultas que percorrem o ciclo de vida dos modelos no Twinscie, respondendo a perguntas que exigem múltiplos encadeamentos de relações, como a consulta Q1 ou a consulta Q4. A flexibilidade do modelo gráfico mostrou-se essencial para capturar a complexidade dos processos do Twinscie, viabilizando uma infraestrutura de rastreabilidade robusta e extensível.

5. Caso de Uso

Sistemas de apoio ao ciclo de vida de modelos de aprendizado de máquina, como o Twinscie, ganham relevância especial quando aplicados em contextos reais que demandam robustez, rastreabilidade e respostas em tempo hábil. O sistema está implementado no Centro de Operações e Monitoramento da cidade do Rio de Janeiro (COR), como parte do projeto Rionowcast [Porto et al. 2022]. O sistema apoia o treinamento e a inferência de modelos de previsão de curto prazo de chuva na cidade do Rio de Janeiro.

Foram construídos dois modelos de *Deep Learning*: um baseado na arquitetura CONVLSTM [Shi et al. 2015], e outro desenvolvido pelo grupo de trabalho, o ST-CONVS2S [Castro et al. 2021]. Ambos os modelos foram treinados com um conjunto de dados construído a partir de informações observacionais coletadas pelo radar de Mangaratiba (RJ) e por pluviômetros do serviço Alerta Rio, associado à Prefeitura do Rio de Janeiro. Os dados de treinamento correspondem ao período de janeiro de 2016 a fevereiro de 2024. O resultado parcial da proveniência capturada para este caso de uso pode ser observado na Figura 4.

6. Resultados e Discussões

Os experimentos realizados tiveram como objetivo avaliar o desempenho da coleta e da consulta de dados de proveniência em dois cenários distintos: (i) utilização de um banco de dados orientado a grafos (Neo4j) e (ii) registro em arquivos de *log*, gravados sequencialmente em disco e processados posteriormente por *scripts* Python, que realizam a leitura direta em memória.

6.1. Ambiente de Execução

Todos os testes foram conduzidos em ambiente controlado, utilizando uma máquina com processador AMD Ryzen 7 7800X3D (8 núcleos físicos e 16 threads), 32 GB de memória RAM e sistema operacional Windows 10. O banco de dados utilizado foi o Neo4j 5.24.0 *Community Edition*, com configurações padrão de instalação. Durante os experimentos, o ambiente foi mantido dedicado, sem execução de outros processos concorrentes, garantindo a estabilidade e a reprodutibilidade das medições.

A abordagem baseada em *logs* utilizou arquivos em formato JSON armazenados localmente, processados por *scripts* desenvolvidos em Python 3.11, que realizam leitura sequencial e estruturação dos dados diretamente em memória. Para o cenário com Neo4j, as consultas foram escritas em Cypher e executadas por meio do driver oficial da linguagem em Python.

Todos os tempos reportados correspondem à média das três últimas de quatro execuções consecutivas, sendo a primeira descartada para eliminar os efeitos de aquecimento do ambiente (cache, buffers e carregamentos iniciais do banco e do sistema operacional). Entre cada ciclo de execução, o ambiente foi reinicializado, a fim de evitar interferências de cache nos resultados.

6.2. Desempenho das Consultas

Nos experimentos iniciais, observou-se que a abordagem baseada em *logs* apresentou desempenho superior na etapa de gravação, independentemente do volume de instâncias processadas. Esse comportamento manteve-se estável mesmo sob cargas elevadas, como no caso de 100.000 instâncias da consulta Q1, cuja gravação foi concluída em apenas 1,58 segundos, em contraste com os 116,50 segundos registrados no Neo4j (Tabela 1).

Por outro lado, ao analisar o tempo de execução das consultas, o cenário se inverte. Embora o Neo4j tenha apresentado tempos ligeiramente mais altos em casos simples, demonstrou maior estabilidade e escalabilidade à medida que a complexidade das relações aumentava. Essa diferença pode ser atribuída à estrutura dos dados: enquanto o Neo4j gerencia nativamente as relações entre entidades, a abordagem com *logs* exige a

reconstrução manual dos encadeamentos, o que acarreta maior custo computacional em consultas que percorrem múltiplos nós e arestas, como ocorre na Q1.

Tabela 1. Tempo médio de gravação e consulta (em segundos) das consultas Q1 e Q4, comparando Neo4j e logs em diferentes volumes de dados

Instâncias	Q1				Q4			
	Neo4j		Logs		Neo4j		Logs	
	Grav.	Cons.	Grav.	Cons.	Grav.	Cons.	Grav.	Cons.
1000	1,56	0,05	0,01	0,01	1,27	0,02	0,01	0,01
10000	11,54	0,43	0,16	0,09	12,52	0,02	0,13	0,09
50000	59,58	1,44	0,80	0,52	62,88	0,04	0,67	0,49
100000	116,50	2,03	1,58	1,09	120,91	0,11	1,33	1,02

Na consulta de dados, o Neo4j apresentou desempenho competitivo em alguns cenários, como na Q4 com 1000 e 10000 instâncias, mas, no geral, os *logs* foram mais estáveis e rápidos. Essa diferença pode ser explicada pela natureza das consultas: a Q1 possui maior profundidade no grafo (cinco nós e quatro saltos), embora seja uma consulta em nível de entidade, o que impacta diretamente o custo computacional no Neo4j. Em contrapartida, os *logs* são lidos de forma sequencial e filtrada, o que tende a apresentar complexidade constante.

Diante desses resultados, o próximo experimento investiga com mais detalhe os limites das abordagens testadas frente a consultas mais complexas. A motivação parte da hipótese de que, embora os *logs* tenham ótimo desempenho em operações simples e sequenciais, podem ser significativamente prejudicados quando a tarefa envolve múltiplas transformações ou a reconstrução de cadeias de dependência mais profundas, como no ”pipeline completo de processamento dos dados e treinamento de um modelo específico”, representado pela Q5 na Figura 3.

Assim, o segundo experimento simula cenários em que é necessário percorrer caminhos longos no grafo de proveniência ou reconstruir trajetórias inversas a partir de vários registros de log, buscando entender o impacto da complexidade estrutural na performance de cada abordagem.

Essa análise busca investigar se a estrutura de gravação e leitura de *logs* se torna um gargalo quando é necessário reconstruir contextos históricos complexos, uma característica intrínseca à proveniência retrospectiva. Por outro lado, o banco de dados orientado a grafos, embora com maior custo inicial de gravação, pode oferecer vantagens significativas na consulta de cadeias relacionais extensas.

A partir disso, os resultados apresentados na Tabela 2, que tratam da consulta Q5, confirmam a hipótese de que o Neo4j se destaca em consultas complexas. Mesmo com o aumento no volume de instâncias, os tempos de consulta permaneceram baixos e estáveis, por exemplo, 0,01 segundo para 100.000 instâncias, com baixo desvio padrão (DP). Em contraste, os *logs* apresentaram crescimento exponencial no tempo de resposta, ultrapassando 875 segundos com apenas 15.000 instâncias. Para 50.000 e 100.000 instâncias, as consultas via *logs* não puderam ser concluídas, pois a execução ultrapassou várias horas sem retorno de resultado, tornando-se inviável na prática.

Embora os tempos de gravação dos *logs* sejam significativamente menores, esse

Tabela 2. Tempo médio (em segundos) e desvio padrão de gravação e consulta da consulta Q5, comparando Neo4j e logs

Instâncias	Neo4j				Log			
	Gravação		Consulta		Gravação		Consulta	
	Média	DP	Média	DP	Média	DP	Média	DP
1000	1,54	0,12	0,00	0,00	0,04	0,00	2,40	0,05
5000	6,53	0,21	0,01	0,00	0,25	0,00	60,24	0,57
10000	12,76	0,07	0,01	0,00	0,50	0,01	287,60	7,00
15000	19,44	0,32	0,02	0,01	0,73	0,00	875,19	8,81
50000	62,92	0,26	0,04	0,00	–	–	–	–
100000	131,00	1,41	0,40	0,04	–	–	–	–

ganho é rapidamente superado pela ineficiência nas consultas complexas. Já o Neo4j, mesmo com maior custo de gravação, mostra-se mais adequado para cenários que exigem recuperação estrutural completa da proveniência.

7. Conclusão

Este trabalho apresentou uma abordagem para estruturar e consultar dados de proveniência no sistema Twinscie, com base no modelo W3C PROV. A modelagem proposta abrangeu entidades, atividades e agentes ao longo do ciclo de vida de modelos de aprendizado de máquina, permitindo rastrear com precisão a origem, transformação e uso de artefatos computacionais. A integração do modelo PROV ao Twinscie viabilizou consultas representativas sobre rastreabilidade, reforçando a capacidade do sistema de prover reprodutibilidade e auditabilidade em aplicações reais. A implementação do grafo de proveniência no Neo4j garantiu compatibilidade com o modelo conceitual e consultas eficientes em estruturas complexas.

Os experimentos comparativos evidenciaram que, embora o registro em *logs* apresente menor custo de gravação, essa abordagem torna-se inviável em cenários que exigem consultas profundas ou reconstituição de cadeias analíticas. Nessas situações, o Neo4j demonstrou desempenho superior, mantendo baixa latência mesmo com grandes volumes de dados e múltiplas relações entre elementos computacionais.

Conclui-se que a adoção de um modelo formal de proveniência, aliada a um banco de dados orientado a grafos, oferece uma solução robusta, flexível e escalável para sistemas de aprendizado de máquina. Essa combinação não apenas amplia a transparência e o controle sobre processos automatizados, mas também fortalece a capacidade de diagnóstico, validação e explicabilidade em ambientes reais de operação.

Como trabalho futuro, pretende-se explorar consultas mais avançadas, incorporar mecanismos de visualização interativa da proveniência e investigar estratégias de otimização da estrutura de grafos, com foco em aplicações em tempo real e em cenários com maior volume de dados.

Agradecimentos

Este trabalho foi desenvolvido no âmbito de projeto de P,D&I com recursos da Petrobras, com apoio da ANP – Agência Nacional do Petróleo, Gás Natural e Biocombustíveis, Brasil, associado às Cláusulas de P,D&I.

Referências

Castro, R., Souto, Y. M., Ogasawara, E. S., Porto, F., and Bezerra, E. (2021). Stconvs2s: Spatiotemporal convolutional sequence to sequence network for weather forecasting. *Neurocomputing*, 426:285–298.

de Almeida, V. K., de Oliveira, D. E., de Barros, C. D. T., Scatena, G. d. S., Queiroz Filho, A. N., Siqueira, F. L., Costa, Ogasawara, E., and Porto, F. e. a. (2024). A digital twin system for oil and gas industry: A use case on mooring lines integrity monitoring. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 322–331, New York, NY, USA. Association for Computing Machinery.

Grieves, M. (2014). Digital twin: Manufacturing excellence through virtual factory replication. Technical report, Florida Institute of Technology.

LNCC (2015). Sdumont. `https://sdumont.lncc.br`.

Moreau, L. and Groth, P. (2013). Prov-overview: An overview of the prov family of documents. `https://www.w3.org/TR/prov-overview/`.

Neo4j (2003). Graph database & analytics. `https://neo4j.com`.

Pina, D., Chapman, A., Kunstmann, L., de Oliveira, D., and Mattoso, M. (2024). Dlprov: A data-centric support for deep learning workflow analyses. In *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning*, DEEM '24, page 77–85, New York, NY, USA. Association for Computing Machinery.

Polyzotis, N., Roy, S., Whang, S. E., and Zinkevich, M. (2017). Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1723–1726, New York, NY, USA. Association for Computing Machinery.

Porto, F., Ferro, M., Ogasawara, E. S., Moeda, T., de Barros, C. D. T., da Silva, A. C., Zorrilla, R., Pereira, R. S., Castro, R. N., Silva, J. V., Salles, R., Fonseca, A. J., Hermsdorff, J., Magalhães, M., Sá, V., Simões, A., Cardoso, C., and Bezerra, E. (2022). Machine learning approaches to extreme weather events forecast in urban areas: Challenges and initial results. *Supercomput. Front. Innov.*, 9(1):49–73.

Schelter, S., Böse, J.-H., Kirschnick, J., Klein, T., and Seufert, S. (2017). Automatically tracking metadata and provenance of machine learning experiments.

Schlegel, A., Auer, S., and Vidal, M.-E. (2023). Mlflow2prov: Creating provenance graphs from mlflow metadata. In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS)*, pages 579–586.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., and Woo, W.-c. (2015). Convolutional lstm network: a machine learning approach for precipitation nowcasting. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 802–810, Cambridge, MA, USA. MIT Press.