UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

JOSÉ VINÍCIUS EVANGELISTA DIAS DE SOUZA

BENCHMARK COMPARATIVO DE FERRAMENTAS DE CÓDIGO ABERTO PARA *MAP MATCHING ONLINE*

QUIXADÁ

2026

JOSÉ VINÍCIUS EVANGELISTA DIAS DE SOUZA

BENCHMARK COMPARATIVO DE FERRAMENTAS DE CÓDIGO ABERTO PARA *MAP MATCHING ONLINE*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Regis Pires Magalhães.
Coorientador: Prof. Dr. José Antônio Fernandes de Macêdo.

QUIXADÁ
2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S238b Souza, José Vinícius Evangelista Dias de.
Benchmark comparativo de ferramentas de código aberto para map matching online / José Vinícius Evangelista Dias de Souza. – 2026.
109 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2026.
Orientação: Prof. Dr. Regis Pires Magalhães.
Coorientação: Prof. Dr. José Antônio Fernandes de Macêdo.

1. Map matching. 2. Benchmark. 3. Online. 4. Geolocalização. 5. Ferramentas de código aberto. I. Título.
CDD 005.1

JOSÉ VINÍCIUS EVANGELISTA DIAS DE SOUZA

BENCHMARK COMPARATIVO DE FERRAMENTAS DE CÓDIGO ABERTO PARA *MAP MATCHING ONLINE*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: 22/01/2026.

BANCA EXAMINADORA

Prof. Dr. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. José Antônio Fernandes de Macêdo (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. José Florêncio de Queiroz Neto
Universidade Federal do Ceará (UFC)

Profa. Dra. Lívia Almada Cruz
Universidade Federal do Ceará (UFC)

À minha família, pela confiança e pelo apoio constantes. Mãe, seu zelo e dedicação foram a base que manteve viva a esperança de seguir em frente. Pai, sua presença trouxe a segurança e a certeza de que nunca estive sozinho nessa jornada.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por minha vida, por sempre me guiar e por mais essa conquista.

Agradeço aos meus pais, à sua fé inabalável e a todos os sacrifícios feitos por mim. Agradeço por sempre estarem presentes e por todo o carinho.

Aos meus irmãos, que sempre estiveram ao meu lado, zelando pelo meu bem-estar e me animando nos momentos difíceis.

Ao Prof. Dr. Regis Pires Magalhães, pela excelente orientação, pelo conhecimento e por me conceder a oportunidade de fazer parte do Insight Data Science Lab.

Agradeço ao Prof. Dr. José Antônio Fernandes Macedo e ao Prof. Dr. José Florêncio de Queiroz Neto por terem me coorientado durante este trabalho, pelos conselhos e ensinamentos, e pela oportunidade de fazer parte da equipe da IGEOS.

À Profa. Dra. Lívia Almada Cruz, pela oportunidade concedida de fazer parte do Insight Data Science Lab e por integrar a banca.

Agradeço ao Prof. Dr. Enyo José Tavares Gonçalves por todas as orientações na escrita desta monografia.

Agradeço ao João, ao Samir, ao Lage e ao Erick por me receberem na IGEOS, por me guiarem com paciência e por cada conselho compartilhado ao longo dessa jornada.

Aos meus amigos, por todas as sugestões, críticas e, principalmente, pelas conversas bem-humoradas no refeitório e na cantina.

Aos meus amigos do ensino médio, sou grato por todos os momentos de descontração e apoio.

“Sabemos que todas as coisas cooperam para o bem daqueles que amam a Deus, daqueles que são chamados segundo o seu propósito.”
(Romanos 8:28, Bíblia Sagrada)

RESUMO

O processo de mapear medições de GPS, frequentemente imprecisas, ruidosas ou esparsas, a via correspondente em um mapa digital é denominado *map matching*. Com o crescimento expressivo do número de dispositivos móveis equipados com sensores de geolocalização, esse procedimento tornou-se um pilar para inúmeras aplicações modernas, sendo diretamente condicionado à precisão e à eficiência das abordagens algorítmicas empregadas. Os métodos de *map matching* dividem-se em dois paradigmas principais: *online*, no qual o trajeto é processado incrementalmente, e *offline*, no qual a trajetória completa é analisada como uma etapa de pós-processamento. Em decorrência dessa relevância, formou-se um amplo ecossistema de ferramentas de código aberto, oferecendo múltiplas alternativas a desenvolvedores e pesquisadores. Contudo, a escolha informada de uma solução adequada, especialmente para cenários *online*, é dificultada pela escassez de *benchmarks* comparativos, reprodutíveis e pela ausência de avaliações empíricas sistemáticas de ferramentas originalmente *offline* quando adaptadas para execução incremental. Nesse contexto, este trabalho desenvolve e aplica um *benchmark* sistemático e reprodutível para a avaliação de ferramentas de *map matching online* de código aberto, incluindo soluções nativamente *offline* adaptadas por meio de técnicas como processamento de mini-lotes para funcionarem *online*. Como principais contribuições, são apresentados um *benchmark harness* automatizado com orquestração baseada em contêineres, a biblioteca `mmlib`, que padroniza a interface do cliente e viabiliza a execução incremental das ferramentas, e um conjunto de dados sintéticos com *ground truth* controlado, gerado por simulação de tráfego urbano. As ferramentas avaliadas são comparadas de forma multidimensional, considerando acurácia espacial, latência, consumo de recursos computacionais e capacidade de processamento contínuo. Os resultados obtidos fornecem evidências empíricas que auxiliam engenheiros de *software* na escolha informada de soluções para aplicações em tempo real e oferecem uma base extensível para a avaliação de novas abordagens e ferramentas no domínio de *map matching*.

Palavras-chave: *map matching*; *benchmark*; *online*; geolocalização; ferramentas de código aberto;

ABSTRACT

The process of mapping GPS measurements, which are often inaccurate, noisy, or sparse, to the corresponding road segment in a digital map is known as map matching. With the significant growth in the number of mobile devices equipped with geolocation sensors, this procedure has become a cornerstone for numerous modern applications, being directly constrained by the accuracy and efficiency of the algorithmic approaches employed. Map matching methods are divided into two main paradigms: online, in which the route is processed incrementally, and offline, in which the complete trajectory is analyzed as a post-processing step. As a result of this relevance, a broad ecosystem of open-source tools has emerged, offering multiple alternatives to developers and researchers. However, the informed selection of an appropriate solution, especially for online scenarios, is hindered by the lack of comparative, reproducible benchmarks and by the absence of systematic empirical evaluations of tools originally offline when adapted for incremental execution. In this context, this work develops and applies a systematic and reproducible benchmark for evaluating open-source online map matching tools, including natively offline solutions adapted through techniques such as mini-batch processing. As main contributions, we present an automated benchmark harness with container-based orchestration, the `mmlib` library, which standardizes the client interface and enables incremental execution of the tools, and a synthetic dataset with controlled ground truth, generated via urban traffic simulation. The evaluated tools are compared in a multidimensional manner, considering spatial accuracy, latency, computational resource consumption, and continuous processing capability. The obtained results provide empirical evidence that helps software engineers make informed choices of solutions for real-time applications and offer an extensible basis for evaluating new approaches and tools in the map matching domain.

Keywords: map matching; benchmark; online; geolocation; open-source tools

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
BVSW	*bounded variable-size sliding window* / janela deslizante de tamanho variável limitado
FOSS	*Free and Open-Source Software* / Software Livre e de Código Aberto
FSW	*fixed-size sliding window* / janela deslizante de tamanho fixo
GLONASS	*Globalnaya Navigatsionnaya Sputnikovaya Sistema* / Sistema Global de Navegação por Satélite
GNSS	*Global Navigation Satellite System* / Sistema Global de Navegação por Satélite
GPS	*Global Positioning System* / Sistema de Posicionamento Global
HMM	*Hidden Markov Model* / Modelo de Markov Oculto
JVM	*Java Virtual Machine* / Máquina Virtual Java
LOS	Line-Of-Sight / Sinal de visada direta
NLOS	Non-Line-Of-Sight / Sinal sem visada direta
NLP	*Natural Language Processing* / Processamento de Linguagem Natural
OHMM	*Online* HMM
OSM	*OpenStreetMap*
OSS	*Open Source Software* / *Software* de Código Aberto
SGBD	Sistema de Gerenciamento de Banco de Dados
SUMO	*Simulation of Urban MObility* / Simulação de Mobilidade Urbana
SUT	*System Under Test* / Sistema sob Teste
VSW	*variable-size sliding window* / janela deslizante de tamanho variável

SUMÁRIO

1 INTRODUÇÃO

O processamento e a análise de dados de localização em tempo real, provenientes de uma crescente variedade de dispositivos móveis equipados com sensores de geolocalização, tornaram-se pilares para inúmeras aplicações modernas, sendo fundamentais para sistemas de mobilidade urbana (UBER TECHNOLOGIES INC., 2025; GOOGLE LLC, 2025b), plataformas de logística e entrega, e algoritmos avançados de recomendação de rotas (Zheng; Xie, 2011; Yuan *et al.*, 2010).

No entanto, os dados brutos desses sensores são imprecisos e inerentemente ruidosos devido a condições como a interferência de sinal em cânions urbanos (Zhang; Hsu, 2021), erros de medição dos sensores de *Global Positioning System* / Sistema de Posicionamento Global (GPS) (U.S. GOVERNMENT, 2025) e a baixa frequência de amostragem (Newson; Krumm, 2009). Dessa forma, resultam em trajetórias que frequentemente se desviam da malha viária real, tornando-as inadequadas para aplicações que exigem alta precisão espacial e consistência topológica.

Nesse contexto, a correspondência de mapas, ou *map matching*, manifesta-se como um processo fundamental, cuja principal função é associar uma sequência de coordenadas imprecisas à rota mais provável em um mapa digital. Contudo, essa tarefa é intrinsecamente complexa. Conforme apontado por Bernstein e Kornhauser (1996), a abordagem intuitiva de simplesmente associar cada ponto a via mais próxima é insuficiente, uma vez que observações ruidosas podem ser equivocadamente atribuídas a vias paralelas ou a diferentes níveis da malha viária, como viadutos e túneis, gerando trajetórias inconsistentes.

Os métodos de *map matching* podem ser classificados de acordo com a disponibilidade das observações em tempo real (*online*) ou após a coleta completa da trajetória (*offline*), conforme discutido por Hashemi e Karimi (2014). Enquanto abordagens *offline* operam sobre trajetórias completas, soluções *online* devem inferir incrementalmente o caminho percorrido à medida que novas observações são recebidas, impondo restrições adicionais de latência e previsibilidade.

Para lidar com esses desafios, uma das famílias de algoritmos mais proeminentes é a baseada em HMM. Essa abordagem probabilística modela as coordenadas ruidosas de *Global Navigation Satellite System* / Sistema Global de Navegação por Satélite (GNSS) como observações de uma sequência de estados ocultos, correspondentes aos segmentos reais da malha viária, empregando algoritmos de inferência, como o de Viterbi (1967), para estimar a trajetória

mais provável.

Os trabalhos de Newson e Krumm (2009) e Goh *et al.* (2012) marcaram um ponto de inflexão na adoção prática dessa abordagem, demonstrando sua eficácia nos contextos *offline* e *online*, respectivamente. A partir desses avanços, a comunidade acadêmica e de desenvolvimento de software passou a adotar amplamente algoritmos baseados em HMM, resultando em diversas ferramentas de código aberto.

Conforme destacado por Wöltche (2023), a maioria dessas ferramentas consiste em implementações originalmente concebidas para o paradigma *offline*. Embora tais soluções possam ser adaptadas para cenários *online* por meio de janelas deslizantes ou processamento em micro-lotes (Goh *et al.*, 2012; Newson; Krumm, 2009), permanece em aberto a questão de como essas adaptações impactam, de forma sistemática, a acurácia, a latência, o custo computacional e a capacidade de processamento contínuo das ferramentas.

Diante dessa lacuna, este trabalho desenvolve e aplica um *benchmark* sistemático e reprodutível para a avaliação comparativa de ferramentas de *map matching* de código aberto no paradigma *online*, incluindo soluções originalmente *offline* adaptadas para execução incremental. Para tal, foi construída uma suíte experimental completa, composta por um conjunto de dados sintético com *ground truth* rigoroso, um *benchmark harness* automatizado baseado em contêineres e uma biblioteca de abstração que padroniza a interação com as ferramentas avaliadas.

A investigação é orientada por questões de pesquisa explícitas, definidas na seção 1.2, que estruturam a análise experimental sob múltiplas dimensões, abrangendo qualidade de correspondência, eficiência temporal, eficiência computacional e capacidade de processamento contínuo. Como resultado, este trabalho produz uma caracterização empírica inédita do comportamento dessas ferramentas em cenários de processamento incremental, oferecendo subsídios concretos para a escolha informada de soluções de *map matching* em aplicações reais.

1.1 Objetivos

O objetivo geral deste trabalho é definir, desenvolver e conduzir um *benchmark* sistemático e reprodutível para a avaliação de ferramentas de *map matching* de código aberto. A análise se concentra em comparar a acurácia, o desempenho e o uso de recursos de duas classes de soluções que operam no paradigma *online*: aquelas nativamente projetadas para o paradigma *online* e aquelas que, sendo originalmente *offline*, foram adaptadas para este modo de operação.

Com isso, este trabalho visa preencher a lacuna de conhecimento existente e disponibilizar subsídios técnicos que orientem a seleção da ferramenta mais apropriada para as demandas de aplicações modernas.

1.1.1 Objetivos Específicos

Para alcançar o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

I. Projetar e gerar um conjunto de dados sintético com *ground truth* controlado, utilizando simulação de tráfego, a fim de viabilizar a avaliação da acurácia em cenários urbanos simples e desafiadores.
II. Produzir um artefato de software funcional, na forma de uma camada de adaptação, capaz de habilitar a execução de ferramentas offline sob as restrições do paradigma online.
III. Avaliar quantitativamente os resultados a partir da execução sistemática da suíte de *benchmark* sobre as ferramentas selecionadas.
IV. Realizar uma análise comparativa aprofundada que identifique os *trade-offs*, os pontos fortes e as fraquezas de cada ferramenta, fornecendo recomendações para a comunidade.

1.2 Questões de pesquisa

A partir desses objetivos específicos, foram formuladas questões de pesquisa que orientam a análise experimental e permitem avaliar, de forma sistemática, os resultados produzidos pelo *benchmark* desenvolvido. São elas:

I. **QP1 – Qualidade de correspondência**

   Como a qualidade de correspondência das ferramentas de map matching varia em função da taxa de amostragem das trajetórias, considerando métricas de acurácia, F1-score e erro, incluindo o erro de Newson e Krumm?

   Esta questão busca investigar de que forma a densidade temporal das observações GNSS influencia a qualidade do *map matching*. A análise dessa relação é central para compreender a robustez das ferramentas frente a dados ruidosos e esparsos, cenário comum em aplicações de tempo real e em ambientes com restrições de coleta.

II. **QP2 – Eficiência temporal**

   Como a latência média de processamento das ferramentas varia em função da taxa de amostragem no paradigma online?

Esta questão visa caracterizar o comportamento temporal das ferramentas sob diferentes regimes de amostragem, permitindo analisar sua adequação a aplicações sensíveis a atraso. A latência de processamento é um fator crítico em sistemas *online*, nos quais decisões devem ser tomadas de forma incremental e com restrições temporais estritas.

III. **QP3 – Eficiência computacional**

Como o custo computacional médio das ferramentas varia em função da taxa de amostragem para uma dada qualidade de correspondência?

Esta questão orienta a análise do compromisso entre custo computacional e qualidade dos resultados. Ao relacionar métricas de uso de CPU e memória com métricas de acurácia e erro, busca-se identificar soluções que ofereçam maior eficiência, isto é, que atinjam níveis adequados de qualidade com menor consumo de recursos.

IV. **QP4 – Capacidade de processamento contínuo**

Como a capacidade média de processamento contínuo das ferramentas varia em função da taxa de amostragem das trajetórias?

Esta questão tem como foco a vazão das ferramentas, entendida como a capacidade de processar fluxos contínuos de dados GNSS de forma sustentada. Essa dimensão é particularmente relevante em cenários de larga escala, nos quais múltiplas trajetórias são processadas simultaneamente e a estabilidade do desempenho ao longo do tempo é um requisito essencial.

Em conjunto, essas questões estruturam a metodologia experimental e guiam a análise dos resultados apresentados ao longo deste trabalho, oferecendo um panorama abrangente do comportamento das ferramentas avaliadas sob diferentes perspectivas operacionais.

1.3 Organização do trabalho

Este trabalho está organizado em capítulos que refletem de forma progressiva o embasamento teórico, o desenvolvimento metodológico e a análise experimental conduzida. No Capítulo 2, é apresentado o referencial teórico, no qual são fundamentados os conceitos essenciais de *map matching*, seus principais desafios e abordagens algorítmicas, com ênfase em métodos baseados em HMM. Nesse capítulo, também são discutidos os princípios fundamentais de *benchmarks* empíricos de *software*, incluindo critérios de qualidade, reprodutibilidade e validade experimental, que norteiam a concepção do *benchmark* proposto.

O Capítulo 3 apresenta os trabalhos relacionados, situando este estudo no contexto

da literatura existente e discutindo abordagens que possuem objetivos ou características similares, bem como suas limitações frente ao escopo adotado.

No Capítulo 4, são descritos os procedimentos metodológicos empregados, abrangendo a definição da suíte de *benchmark*, a adaptação das ferramentas selecionadas para o paradigma *online*, o desenvolvimento do *benchmark harness* e a estratégia de instrumentação e coleta de métricas.

O Capítulo 5 detalha a construção do conjunto de dados sintético com *ground truth* controlado, utilizado como carga de trabalho no *benchmark*. São descritas as etapas de obtenção da malha viária, geração das trajetórias, pós-processamento e validação do conjunto de dados.

Os resultados experimentais e sua análise são apresentados no Capítulo 6, no qual as ferramentas avaliadas são comparadas de forma sistemática à luz das questões de pesquisa definidas, considerando métricas de acurácia, eficiência temporal, custo computacional e capacidade de processamento contínuo.

Por fim, o Capítulo 7 apresenta as conclusões do trabalho, sintetizando as principais contribuições, discutindo as implicações práticas dos resultados obtidos e apontando direções para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta a fundamentação teórica do trabalho, partindo dos conceitos essenciais do *map matching* (seção 2.1) para, em seguida, detalhar a técnica baseada em modelos ocultos de Markov (seção 2.2). Com essa base estabelecida, a discussão se aprofunda no paradigma de *Map Matching Online* (seção 2.3). Por fim, alinhado ao objetivo central desta pesquisa, é apresentado o conceito fundamental de *benchmark* (seção 2.4).

2.1 *Map matching*

O propósito geral de um algoritmo de *map matching* é identificar corretamente qual segmento o veículo está viajando e determinar sua localização naquele segmento (Quddus *et al.*, 2007). Porém, é necessário o pleno entendimento do problema de *map matching* em si antes de considerar seus algoritmos. As próximas subseções abordam, respectivamente: (2.1.1) as causas e implicações dos erros de medição provenientes de sistemas GNSS, que motivam o uso do *map matching*; (2.1.2) sua definição formal e conceitual; (2.1.3) seus principais desafios técnicos; (2.1.4) seus paradigmas; (2.1.5) suas principais abordagens; e (2.1.6) suas métricas de avaliação.

2.1.1 Erro de Medições de GNSS

Uma medida de localização é definida por uma observação pontual de um objeto no espaço geográfico acompanhada de um instante de tempo correspondente. Em geral, tais medições são obtidas via sistemas de posicionamento GNSS como o GPS, o *Globalnaya Navigatsionnaya Sputnikovaya Sistema* / Sistema Global de Navegação por Satélite (GLONASS) ou o Galileo [1]. No contexto de *map matching*, essa medida pontual pode ser representada como:

$$p_i = (\phi_i, \lambda_i, t_i) \tag{2.1}$$

No qual, ϕ_i é a latitude da medição, λ_i é a longitude e t_i é o instante temporal correspondente do i-ésimo ponto de uma trajetória. Ademais, devido à natureza bi-dimensional do problema de *map matching* em aplicações de mobilidade urbana, o componente de altitude é tradicionalmente desconsiderado, mesmo que alguns sistemas de geolocalização, como o GPS também a forneçam (U.S. GOVERNMENT, 2025).

[1] Sistema de navegação Galileo da União Europeia

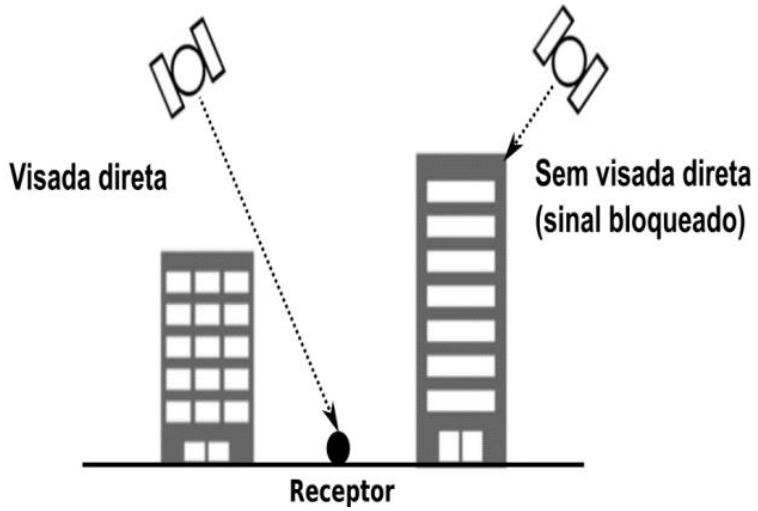
Contudo, é fundamental ressaltar que tais medições não são isentas de erro. Conforme o artigo de Pissardini *et al.* (2017), existem quatro principais causas responsáveis por essas falhas. São elas:

I. **Bloqueio GNSS**: O cenário ideal para aferições – ilustrado na Figura 1 – é obtido diretamente entre o satélite GNSS e o receptor, livre de bloqueios ou interferências causadas por outros elementos na transmissão. O sinal captado nessa condição é chamado de Line-Of-Sight / Sinal de visada direta (LOS). Portanto, bloqueio GNSS define qualquer cenário que impeça a propagação total ou parcial do sinal GNSS.
II. **Recepção sem visada direta**: Dissoante da recepção LOS, o sinal Non-Line-Of-Sight / Sinal sem visada direta (NLOS) ocorre numa circunstância no qual o receptor recebe apenas sinais dos satélites refletidos por elementos do ambiente, o que gera um erro da pseudo distância. Neste caso, o erro é sempre positivo, podendo variar de dezenas de metros a um valor potencialmente ilimitado (Groves *et al.*, 2012). Essa circunstância é mostrada na Figura 2.
III. **Multicaminhamento**: O quadro no qual um receptor obtém um mesmo sinal de um satélite por múltiplos caminhos em simultâneo, é chamado de multicaminhamento. Neste caso, podem ser incluídos ambos os tipos de sinal: LOS e NLOS, prejudicando assim, a seleção sobre qual sinal utilizar. A Figura 3 elucida essa conjunção.

Adicionalmente, destaca-se um cenário particular que agrava os erros de posicionamento: o cânion urbano. Este termo descreve, convencionalmente, ambientes caracterizados pelo adensamento de edifícios altos, construções e outras estruturas que restringem a visibilidade do céu para um receptor GNSS (Pissardini *et al.*, 2017). Nessas condições, a recepção de sinais diretos é limitada a satélites em elevações muito altas. Como resultado, o número de satélites nessas condições é intensamente menor, comprometendo a acurácia da solução de posicionamento (Groves *et al.*, 2012).

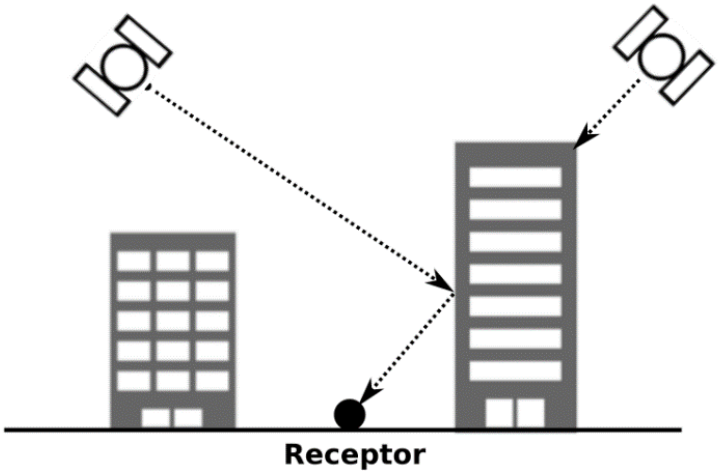
Em conjunto, essas fontes de erro demonstram que a trajetória obtida por um receptor GNSS, especialmente em ambientes urbanos, é uma representação ruidosa e frequentemente imprecisa do caminho real. Pontos podem se desviar dezenas de metros de sua localização verdadeira, criar "saltos" ilógicos na trajetória e falhar em representar curvas e manobras com fidelidade. Essa discrepância torna os dados brutos inadequados para uma vasta gama de aplicações que dependem de alta acurácia posicional, como sistemas de navegação, monitoramento de frotas e análise de mobilidade. Para mitigar o impacto desses erros e reconciliar as medições

Figura 1 – Exemplo de sinais que podem ser captados diretamente ou bloqueado por obstáculos no ambiente.



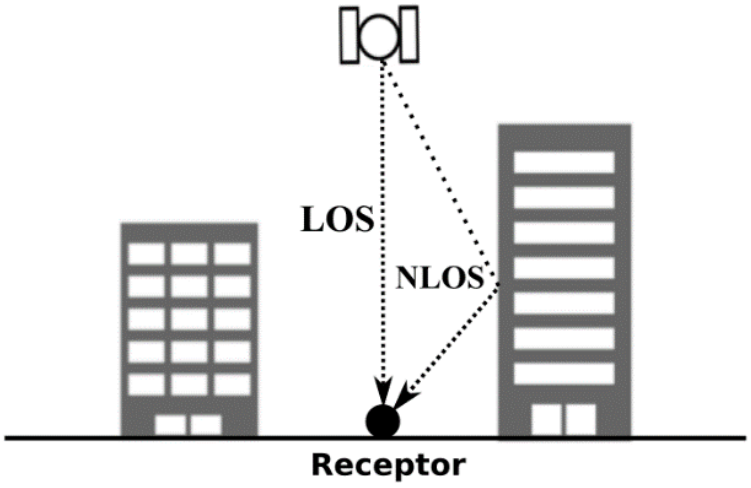
Fonte: Pissardini *et al.* (2017).

Figura 2 – Recepção de dados NLOS



Fonte: Pissardini *et al.* (2017).

Figura 3 – Ilustração do fenômeno de multicaminhamento, com a recepção simultânea de sinais diretos (LOS) e refletidos (NLOS).



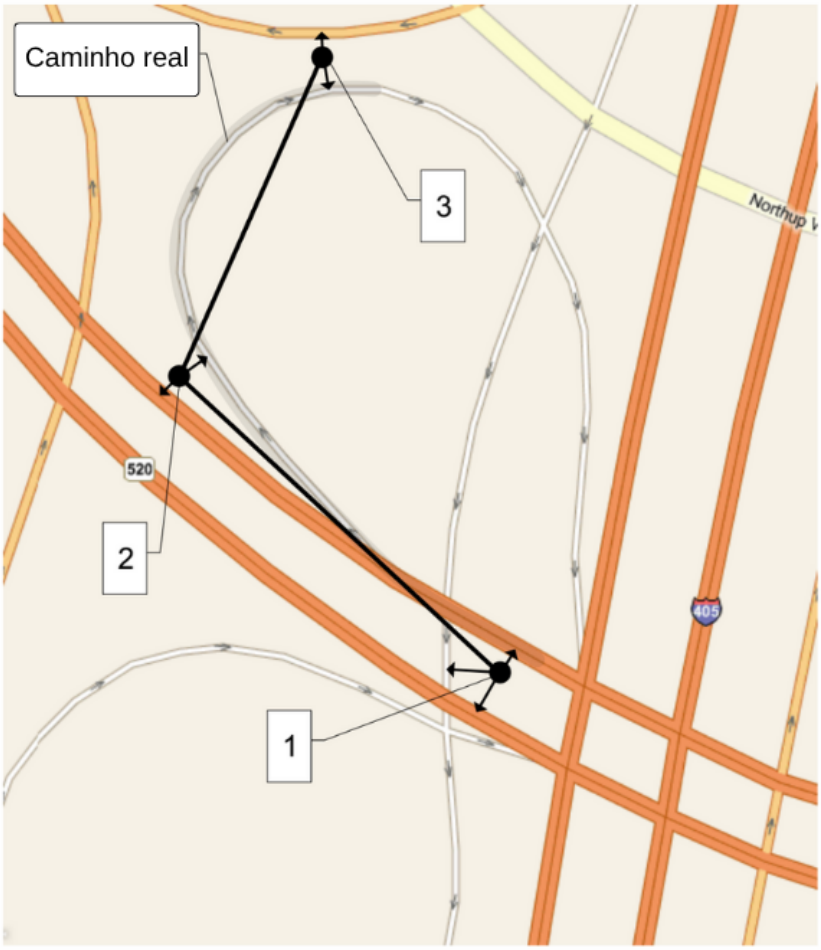
Fonte: Pissardini *et al.* (2017).

imprecisas com a realidade da malha viária, o processo de *map matching* torna-se uma etapa fundamental.

2.1.2 Definição formal e conceitual

Em essência, conforme o trabalho de White *et al.* (2000), o problema de *map matching*, ilustrado na Figura 4, pode ser definido como o processo de determinar a qual segmento da malha viária pertence uma medição de localização, bem como a trajetória correspondente formada por uma sequência dessas medições ao longo do tempo. Esse método é fundamental e atua como uma ponte crítica entre as posições geográficas brutas aferidas e os mapas digitais.

Figura 4 – *Map matching* consiste em comparar os locais medidos (pontos pretos) com a malha rodoviária para inferir a trajetória real percorrida pelo veículo (curva cinza-claro). A simples correspondência com a estrada mais próxima é propensa a erros.



Fonte: Adaptado de Newson e Krumm (2009).

Apesar de sua formulação parecer simples, a tarefa é notoriamente desafiadora. Embora, intuitivamente, a solução óbvia possa ser conectar o ponto mensurado ao segmento de

rua mais próximo, ela é propensa a erros de aferição de sensores GNSS mostrados na subseção 2.1.1. A Figura 4, adaptada do trabalho de Newson e Krumm (2009), ilustra essa dificuldade: os pontos pretos representam as medições sequenciais, enquanto a linha cinza-claro indica a trajetória real. Embora o trajeto correto seja visualmente evidente, ao atribuir o segundo e o terceiro ponto às estradas adjacentes mais próximas, esses registros são erroneamente associados. Como resultado, o caminho inferido diverge da realidade, evidenciando a complexidade do problema e a necessidade de métodos mais robustos.

Para desenvolver e analisar tais métodos robustos, é primordial, primeiramente, formalizar os componentes centrais do problema conforme definido por Bernstein e Kornhauser (1996):

I. **Trajetória**: Uma trajetória, T é formada por uma sequência de N pontos p_i tais que:

$$T = (p_i)_{i=1}^{N} \tag{2.2}$$

No qual p_i, definido na equação 2.1, representa o i-ésimo ponto de T.

II. **Segmento de Via**: Um segmento de via r, que representa um trecho da malha viária, possui dois componentes principais:

- **Geometria**: A forma do segmento é descrita por uma polilinha, ou seja, uma sequência de M vértices ordenados que definem sua localização no mapa:

$$r_{geom} = (v_m)_{m=1}^{M} \tag{2.3}$$

Onde cada vértice v_m é especificado por suas coordenadas de latitude ϕ_m e longitude λ_m.

- **Atributos Semânticos**: informações que descrevem as propriedades do segmento, como o limite de velocidade, o tipo de via e a permissão de tráfego (mão única ou dupla).

III. **Malha Viária**: A malha viária é formalmente representada como um grafo direcionado $G = (V, E)$, no qual V é um conjunto de **nós** (ou vértices) que representam as interseções das ruas ou os pontos finais dos segmentos, e E, por sua vez, um conjunto de **arestas**, cada uma correspondendo a um **segmento de via** r que conecta dois nós em V.

Portanto, o problema de *map matching* pode ser formalmente definido. Em sua forma mais fundamental, o objetivo é:

> "Dado uma trajetória T e uma malha viária G, encontrar a sequência de segmentos de via em G que corresponde mais probabilisticamente ao caminho real percorrido."

Figura 5 – Antes e depois do *map matching*.

Fonte: Elaborada pelo autor.

Esta definição abrange desde a simples correspondência de pontos a segmentos individuais até a inferência de rotas completas, que é o foco dos métodos mais avançados. A eficácia deste processo é claramente exibida na Figura 5. No exemplo, a trajetória original, composta pelos pontos de GPS brutos, exibe erros de aferição notórios, com medições que se desviam significativamente da malha viária. Em contrapartida, o trajeto resultante demonstra a capacidade do algoritmo em corrigir esses desvios e reconstruir um caminho não apenas geometricamente preciso, mas também topologicamente consistente com a rede de ruas.

2.1.3 Desafios

De acordo com a revisão de Chao *et al.* (2020), existem três grandes desafios que afetam o *map matching*, todos provenientes dos erros de medição GNSS; são eles:

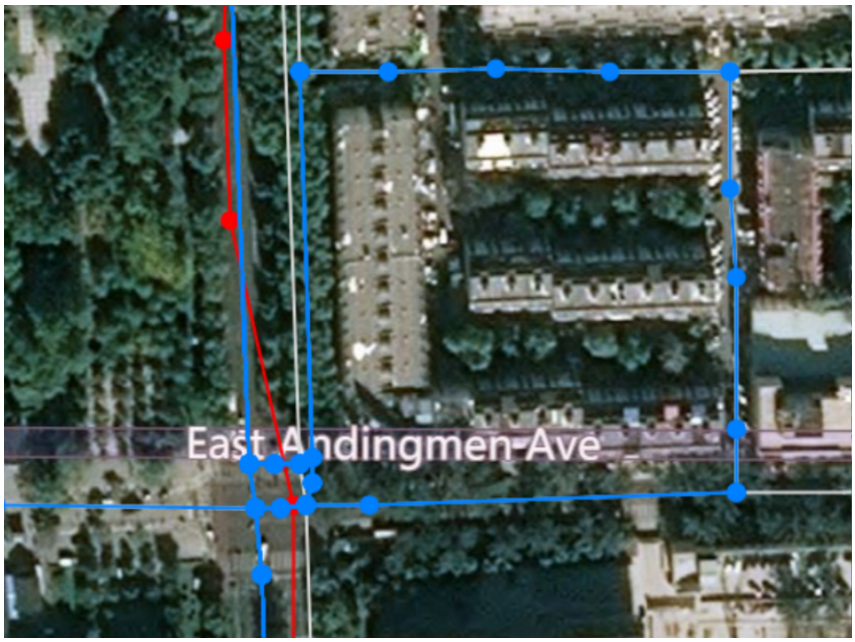
I. **Desvio desnecessário**: Este problema, exibido na Figura 6, ocorre quando a rota final contém um desvio ilógico, sendo mais comum em trajetórias de alta frequência. Ele é causado quando dois pontos consecutivos e muito próximos são mapeados de forma que o segundo ponto p_{i+1} é associado a um trecho da via anterior ao do primeiro ponto p_i. Isso força o algoritmo a calcular uma rota longa e irreal para conectar os dois locais de forma topologicamente válida.

II. **Quebra de correspondência**: Ilustrada na Figura 7, a quebra de correspondência ocorre quando a rota final gerada pelo algoritmo é desconectada, formando um "salto" na trajetória. A principal causa são os *outliers* de medição (pontos com erro muito grande), que podem

ser mapeados para um local na malha viária que não possui conexão com o ponto anterior. A solução mais comum para este problema é a identificação e remoção desses *outliers* durante o pré-processamento dos dados.

III. **Incerteza da correspondência**: Este problema descreve cenários onde o algoritmo tem dificuldade em determinar a via correta devido a ambiguidades. O principal fator que agrava essa incerteza é a densidade da malha viária: quanto mais ruas próximas existirem, maior a chance de uma correspondência errada. A baixa qualidade da trajetória de GPS original também aumenta essa incerteza. Garantir a precisão em áreas urbanas densas, portanto, permanece um desafio em aberto.

Figura 6 – Desvio desnecessário em *map matching*. As linhas vermelhas e azuis, são das trajetórias reais e calculadas, respectivamente.



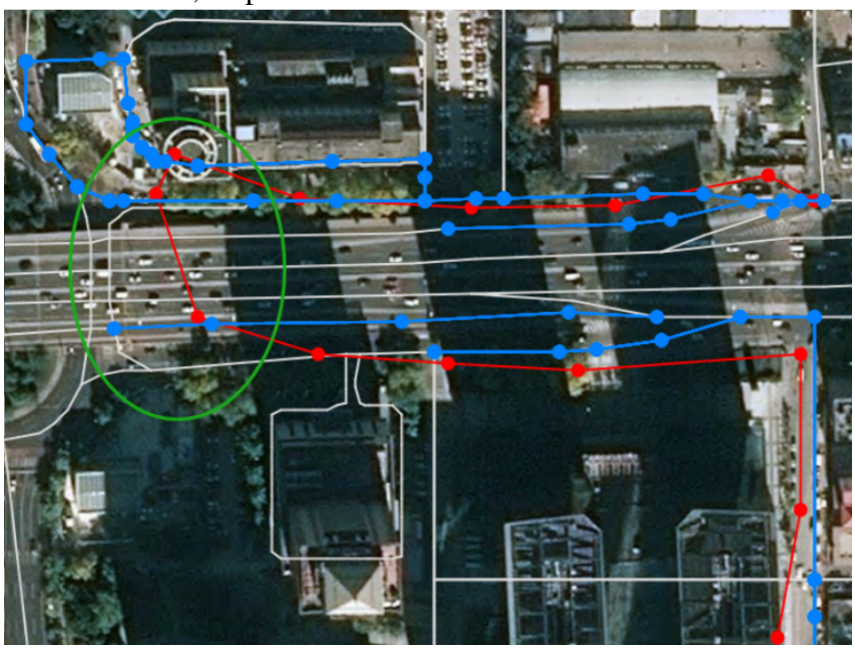
Fonte: Adaptado de Chao *et al.* (2020).

2.1.4 Paradigmas de Processamento em Map Matching

Os métodos de *map matching* dividem-se em duas categorias quanto à disponibilidade dos dados: incremental (*online*) e pós-processamento (*offline*), conforme apontado na revisão de Hashemi e Karimi (2014). A escolha entre os paradigmas é ditada pelos requisitos da aplicação e impõe diferentes desafios e abordagens algorítmicas.

O paradigma ***offline***, opera sobre a trajetória completa, que, por sua vez, é fornecida integralmente ao algoritmo antes do início da correspondência. Sua principal vantagem é a

Figura 7 – Quebra de correspondência em *map matching*. As linhas vermelhas e azuis, são das trajetórias reais e calculadas, respectivamente.



Fonte: Adaptado de Chao *et al.* (2020).

visão global dos dados. Isso permite a aplicação de algoritmos de otimização que analisam pontos passados e futuros para resolver ambiguidades complexas, resultando em uma acurácia potencialmente superior. Isso torna essa abordagem ideal para aplicações que não exigem resposta imediata, como análises de dados históricos de mobilidade, planejamento urbano e pesquisas acadêmicas.

Em contrapartida, o paradigma ***online*** foi pensado para cenários onde a latência é um fator crítico, como em sistemas de navegação veicular, e os dados são processados em um estilo de *streaming* (Chao *et al.*, 2020). Neste modo, o algoritmo processa cada ponto GNSS incrementalmente, sem conhecimento sobre o futuro da rota. A decisão sobre a correspondência correta deve ser tomada com baixa latência de forma contínua. Esta limitação impõe o uso de algoritmos de processamento incremental, como Filtros de Partículas ou variantes de HMM com janelas deslizantes ou em lotes. Tais métodos permitem a manutenção de múltiplas trajetórias candidatas simultaneamente, resolvendo ambiguidades topológicas incrementalmente, enquanto otimizam o custo computacional e são abordados de forma detalhada na seção 2.3.

2.1.5 Abordagens

A revisão de Quddus *et al.* (2007) classifica as abordagens de *map matching* em quatro categorias: geométricas, topológicas, probabilísticas e avançadas.

As abordagens geométricas, pioneiras na área, utilizam apenas a geometria da malha viária. Elas associam cada ponto de GNSS ao segmento de via mais próximo, considerando o formato das vias, mas ignorando como elas se conectam. As principais técnicas dessa classe são o *point-to-point*, *point-to-curve* e *curve-to-curve*, que, apesar de simples e rápidas, são muito suscetíveis a erros em áreas com vias paralelas ou viadutos (Bernstein; Kornhauser, 1996; White *et al.*, 2000).

Para superar as limitações geométricas, as abordagens topológicas incorporam a conectividade da rede viária na análise. Em vez de avaliar pontos isoladamente, elas também consideram a sequência de pontos e comparam-na com os caminhos possíveis no mapa. Um exemplo representativo é o algoritmo *ST-Matching* de Lou *et al.* (2009), que funciona com base na similaridade entre a forma da trajetória GNSS e a forma dos caminhos candidatos no grafo da malha viária, garantindo rotas topologicamente consistentes.

As abordagens probabilísticas representam um avanço significativo, pois tratam a incerteza de forma explícita, buscando encontrar a rota mais provável, e não apenas a mais próxima. Essas técnicas, como as propostas por Newson e Krumm (2009) e Goh *et al.* (2012), baseadas em HMM, ou as que utilizam filtros de partículas, como a de Kempinska *et al.* (2016), modelam tanto o erro da medição do GPS quanto a probabilidade de transição entre os segmentos de via.

Finalmente, a categoria de abordagens avançadas engloba técnicas mais recentes que não se enquadram perfeitamente nas anteriores, muitas vezes combinando-as ou utilizando novos paradigmas. Destacam-se aqui os métodos baseados em aprendizado de máquina, que tratam o map matching como um problema de tradução de sequências, aplicando modelos inspirados da área de *Natural Language Processing* / Processamento de Linguagem Natural (NLP) e arquiteturas como os Transformers (Mohammadi; Smyth, 2025; Jin *et al.*, 2022).

Dentre essas categorias, as abordagens probabilísticas baseadas em HMM são o foco deste trabalho. Conforme aponta o *benchmark* de Wöltche (2023), esta técnica representa o estado da arte para a maioria das ferramentas de map matching de código aberto de alto desempenho. Por essa razão, a seção 2.2 foi dedicada a aprofundar os fundamentos teóricos desta abordagem.

2.1.6 Métricas

Como forma de avaliar algoritmos de *map matching* de maneira quantitativa, utilizam-se métricas empíricas que comparam o trajeto resultante da abordagem ($P_{calculado}$) com a trajetória real de referência (P_{real} ou *ground truth*). Um dos conjuntos de métricas mais consolidadas na literatura, apresentado por Li *et al.* (2014), fundamenta-se nos conceitos de Precisão e Revocação, os quais são combinados de forma harmônica por meio do F_1 *score*. A partir deste, define-se a taxa de erro como o complemento de 1 do F_1, o qual quantifica o grau global de imperfeição da correspondência ao sintetizar as perdas de precisão e de revocação em um único índice, a fim de fornecer uma avaliação única e robusta.

Neste contexto, a métrica de precisão mede a fração do caminho calculado que está correta, avaliando a acurácia do resultado. Já a revocação, por sua vez, mede a fração do caminho real que foi corretamente identificada. Formalmente, considerando x, uma lista de arestas e $Comprimento(x)$, o somatório dos comprimentos das arestas de x:

$$Precisão = \frac{Comprimento(P_{calculado} \cap P_{real})}{Comprimento(P_{calculado})} \tag{2.4}$$

$$Revocação = \frac{Comprimento(P_{calculado} \cap P_{real})}{Comprimento(P_{real})} \tag{2.5}$$

$$F_1 = 2 \cdot \frac{Precisão \cdot Revocação}{Precisão + Revocação} \tag{2.6}$$

$$TaxaDeErro = 1 - F_1 \tag{2.7}$$

Para medir o erro, Newson e Krumm (2009) apresentam uma métrica que quantifica o quão diferente a trajetória inferida $(P_{calculado})$ é da real (P_{real}). A métrica é composta pela soma de dois tipos de erro: o comprimento das arestas incorretamente adicionadas $((d_-))$ e o comprimento das arestas da trajetória real que não foram correspondidas (d_+), definidos por:

$$d_- = Comprimento(P_{real}) - Comprimento(P_{calculado} \cap P_{real}) \tag{2.8}$$

$$d_+ = Comprimento(P_{calculado}) - Comprimento(P_{calculado} \cap P_{real}) \tag{2.9}$$

$$NewsonKrumError\ (NKError) = \frac{d_- + d_+}{Comprimento(P_{real})} \tag{2.10}$$

Esta métrica pode ser interpretada como a razão do erro total (soma das partes ausentes e das partes incorretamente adicionadas) em decorrência do comprimento da trajetória verdadeira. Logo, um valor próximo de zero indica uma alta fidelidade entre a rota calculada e a real, enquanto valores mais altos significam uma diferença significativa, o que torna esta uma medida direta e intuitiva entre as duas trajetórias.

2.2 Técnica de *Map Matching* baseada em Modelos ocultos de Markov

Esta seção apresenta a técnica baseada em HMM, dada a sua condição de estado da arte nas principais ferramentas de *map matching* de código aberto (Wöltche, 2023). Primeiramente, são exibidos os conceitos básicos, iniciando por processos estocásticos na subseção 2.2.1, depois por Cadeias de Markov, que são um tipo de processo estocástico na subseção 2.2.2; por fim, será apresentado o HMM em si na subseção 2.2.3. Então, após os conceitos básicos, é apresentada a modelagem do *map matching* como HMM na subseção 2.2.4.

2.2.1 Processo estocástico

Processo estocástico é o termo que define um modelo matemático que descreve a evolução de um sistema regido por probabilidades ao longo de uma medida de progresso (geralmente, o tempo).

Conforme Ross (2014a), um processo estocástico é formalmente definido por um conjunto de variáveis aleatórias,$\{X(t), t \in T\}$ sendo t o índice de X de modo que, para cada $t \in T$ $X(t)$ é uma variável aleatória. Esse índice é comumente associado ao tempo, sendo esta a medida de progresso padrão, salvo especificação contrária. Assim, $X(t)$ representa o estado do processo no instante t.

A título de exemplificação, pode-se citar o número de pessoas na fila de um banco ao longo do dia. Nesse exemplo, $X(t)$ é a quantidade de pessoas em espera no instante t. Essa quantidade é aleatória devido à incerteza tanto no tempo de atendimento de cada cliente quanto

na chegada de novos clientes ao sistema. Essa incerteza é o que difere os processos estocásticos dos processos determinísticos, sendo essa modelagem chave para modelar cenários como a flutuação do preço das ações no mercado financeiro, a chegada de pacotes de dados em um roteador de rede ou a própria trajetória de um veículo em meio ao trânsito urbano, onde cada decisão futura possui um grau de imprevisibilidade.

2.2.2 Cadeia de Markov

Em essência, uma cadeia de Markov é um processo estocástico com uma propriedade particular: o estado futuro X_{n+1} depende apenas do estado atual X_n, sendo condicionalmente independente dos estados passados $X_{n-1}, X_{n-2}, \ldots, X_0$.

Formalmente, conforme apresentado por Ross (2014b), uma cadeia de Markov é definida por um processo estocástico $\{X_n, n = 0, 1, 2, \ldots, \}$ que assume um número finito ou enumerável de valores possíveis, de modo que $X_n = i$ significa que, no tempo n, o processo está no estado i, e nele há uma probabilidade fixa P_{ij} de que o próximo estado será j, sendo chamada de probabilidade de transmissão de i para j. Logo, para todos os estados $i_0, i_1, \ldots, i_{n-1}, i, j$ e para todos $n \geq 0$, na Equação 2.11:

$$P\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_1 = i_1, X_0 = i_0\} = P_{ij} \tag{2.11}$$

Essa equação, conhecida como Propriedade de Markov, é a característica intrínseca do modelo. Ela estabelece que, para prever o futuro do sistema, toda a informação relevante está contida exclusivamente no estado presente.

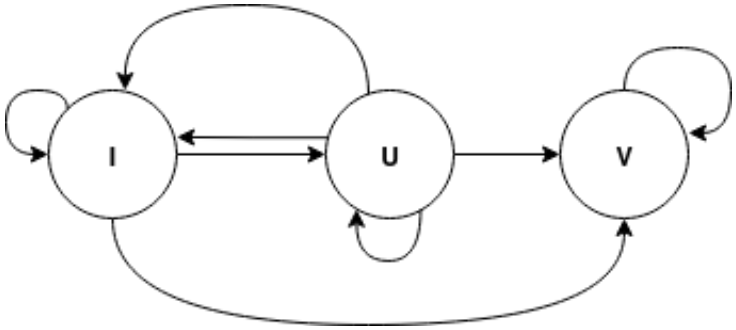
Como exemplo prático, considere a icônica fala do personagem fictício Sherlock Holmes na obra de Doyle (1887) intitulada “Um Estudo em Vermelho”:

> “Uma vez eliminado o impossível, o que restar, não importa o quão improvável, deve ser a verdade.” (Doyle, 1887)

A lógica dedutiva de Sherlock pode ser interpretada e modelada como um processo estocástico de refinamento de hipóteses. Para o exemplo, propõe-se uma cadeia de Markov de três estados representando, cada um, hipóteses impossíveis (**I**), hipóteses improváveis (**U**) e a verdade (**V**), respectivamente. À medida que novas evidências são observadas, as hipóteses

transitam entre estados até alcançar um estado absorvente[2] que representa a conclusão lógica implacável: a verdade. A Figura 8 ilustra essa modelagem.

Figura 8 – Exemplo de Modelagem de uma Cadeia de Markov



Fonte: Elaborada pelo autor.

Esse exemplo, ainda que abstrato, é útil para demonstrar como cadeias de Markov podem modelar processos mentais, decisões progressivas ou sistemas de filtragem.

2.2.3 Hidden Markov Model

Um modelo oculto de Markov, ou HMM (*Hidden Markov Model*), é uma extensão da cadeia de Markov clássica. Ele é composto por dois processos estocásticos interligados: uma cadeia de Markov que não é observável diretamente (oculta), mas pode ser observada através de outro conjunto de processos estocásticos que produzem a sequência de observações visíveis (Rabiner, 1989).

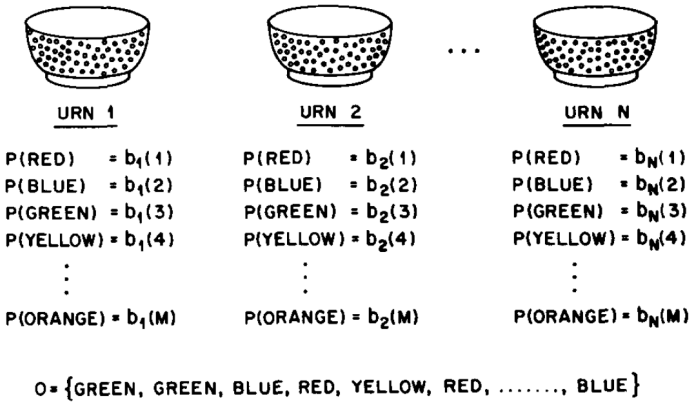
Para o pleno entendimento, utiliza-se o exemplo clássico proposto por Rabiner (1989), ilustrado na Figura 9. Nesse cenário hipotético, um experimento é conduzido em uma sala por uma pessoa, enquanto o observador está do lado de fora. Na sala, há um conjunto de N urnas, cada uma contendo uma grande quantidade de bolas coloridas. A mistura de cores é diferente em cada urna. A pessoa dentro da sala realiza um procedimento em cada passo de tempo t: ela escolhe uma urna com base em um processo de decisão probabilístico e, então, retira uma bola aleatoriamente da urna escolhida, anuncia a cor para o observador e a devolve à mesma urna.

Os observadores sabem o número de urnas e a proporção de cores em cada uma. Contudo, no final do experimento, o observador sabe a sequência de cores anunciadas (ex: "vermelho, azul, azul, vermelho..."), porém, não sabe a sequência de urnas escolhidas.

Neste cenário, os componentes do HMM são mapeados da seguinte forma:

[2] Um estado que, uma vez alcançado, não pode ser deixado.

Figura 9 – Uma urna de N estados e um modelo das esferas que ilustrando um caso geral de HMM



Fonte: Rabiner (1989).

I. **Estados Ocultos (q_t)**: A escolha da urna no tempo t. Esta é a informação que está "oculta" para o observador. A sequência de escolhas de urnas forma a cadeia de Markov subjacente.

II. **Observações (O_t):** A cor da bola sorteada no tempo t. Esta é a única informação que o observador sabe diretamente.

III. **Probabilidades de Transição (A):** A probabilidade de a pessoa escolher a urna j no próximo passo, dado que ela escolheu a urna i no passo atual. Essas probabilidades definem a cadeia de Markov oculta.

IV. **Probabilidades de Emissão (B):** A probabilidade de sortear uma bola de uma cor específica vinda de um estado oculto específico (ou seja, da urna específica que foi escolhida). Essa probabilidade é definida pela proporção de bolas coloridas em cada urna.

O HMM, portanto, fornece o subsídio matemático necessário para responder a perguntas complexas, como: **"Dada a sequência de cores observada, qual é a sequência mais provável de urnas que foram escolhidas?"**. Essa questão é exatamente análoga à questão do problema do *map matching*.

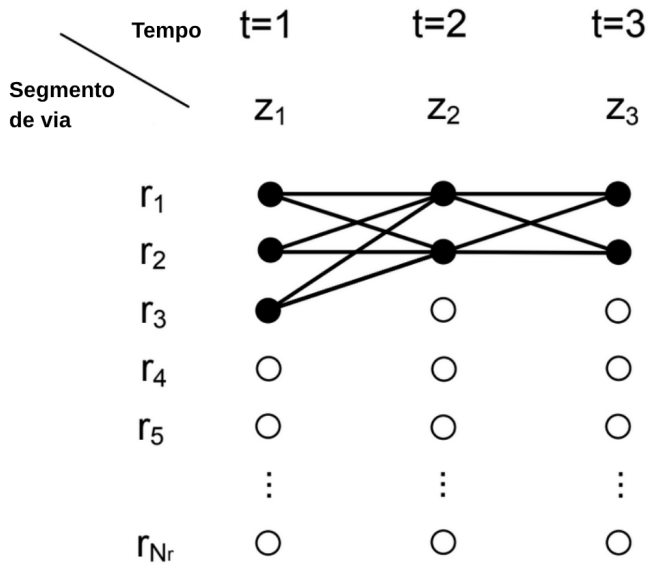
2.2.4 Modelagem do map matching Como um Problema HMM

Proposta por Newson e Krumm (2009), a modelagem do problema de *map matching* com um HMM é uma analogia direta ao exemplo das urnas e bolas apresentado anteriormente. Nesta formulação, a sequência de estados ocultos é a verdadeira trajetória do veículo, representada por uma sequência de segmentos de via $(r_1, r_2, \ldots, r_t)$. No caso, as observações, por sua vez, são justamente a sequência de coordenadas de GNSS imprecisas $(z_1, z_2, \ldots, z_t)$ que o

dispositivo registra. O objetivo é encontrar a sequência de estados que mais provavelmente "emitiu" a sequência de observações.

Nesse contexto, as probabilidades de transição (A) são governadas pela topologia da malha viária, representando a probabilidade de se mover de um segmento de via para outro adjacente. Já as probabilidades de emissão (B), por sua vez, quantificam a chance de uma medição de GNSS (z_t) ter sido gerada por um segmento de via candidato r_i, geralmente com base na distância entre eles. A Figura 10 ilustra essa relação, na qual, para cada observação z_t, existe um conjunto de segmentos candidatos.

Figura 10 – Para cada medição z_t, o HMM considera todos os segmentos de via r_i, bem como todas as transições entre eles



Fonte: Adaptado de Newson e Krumm (2009).

Essa abordagem tem como principal vantagem a análise da trajetória de forma integral, levando em conta diversas possibilidades de mapeamento para cada ponto GNSS. Então, dado o modelo, em vez de procurar todas as rotas possíveis, o algoritmo de Viterbi (1967) é utilizado para identificar, de forma eficaz, o trajeto mais provável que explica a sequência de observações. Essa habilidade de identificar uma solução globalmente ótima, em vez de apenas localmente, confere ao método uma grande resistência a ruídos e ambiguidades, o que justifica sua relevância nas técnicas contemporâneas de *map matching*.

2.3 *Map Matching Online*

Como o foco deste estudo é o paradigma *online*, esta seção se aprofunda nas estratégias que viabilizam o processamento incremental de trajetórias em fluxo (*streaming*). Conforme já abordado na subseção 2.1.4, o desafio fundamental do *map matching online* reside na necessidade do mapeamento incremental, ponto a ponto, sem o conhecimento de dados futuros (Chao *et al.*, 2020). Tal restrição impõe um dilema entre a acurácia, beneficiada pela posse de um contexto mais amplo da trajetória, e a baixa latência exigida pelas aplicações. Neste panorama, a principal estratégia empregada na literatura é a técnica de janela deslizante (*sliding window*) (Mattheis *et al.*, 2014; Goh *et al.*, 2012; Rehrl *et al.*, 2018; Fu *et al.*, 2021), que busca um bom equilíbrio entre a análise local e a resposta instantânea.

Esta técnica opera sobre um subconjunto limitado e recente da trajetória mediante uma "janela" de tamanho N para tomar uma decisão sobre o mapeamento do ponto mais atual. Ao considerar um pequeno lote de pontos em vez de um único, o algoritmo ganha contexto local, o que aumenta significativamente a robustez contra observações ruidosas e melhora a consistência da rota, sem a necessidade de aguardar a trajetória completa. O trabalho de Goh *et al.* (2012) apresenta, investiga e discute variações dessa abordagem, propondo e avaliando sistematicamente diferentes implementações da janela. Em seu estudo, os autores introduziram três variações principais desta técnica, cada uma com um balanço distinto entre complexidade e acurácia:

I. **Janela Deslizante Fixa (FSW - *Fixed Sliding Window*):** É a abordagem tradicional e mais simples, na qual o algoritmo opera sobre uma janela de tamanho fixo (contendo os últimos N pontos). Embora seja fácil de implementar, sua natureza rígida pode ser ineficiente, processando mais dados que o necessário em trechos simples ou não tendo contexto suficiente em trechos complexos.

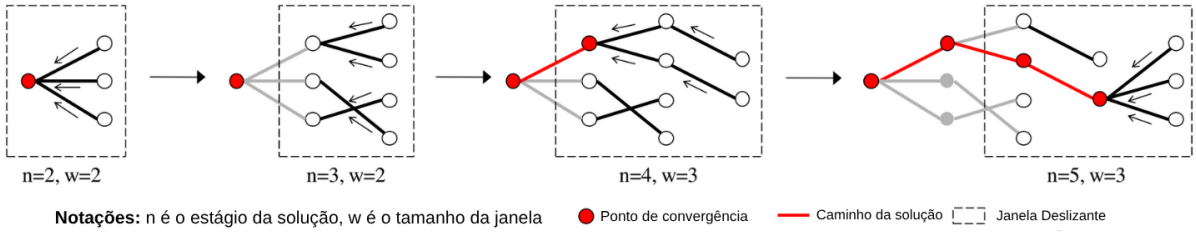
II. **Janela Deslizante Variável (VSW - *Variable Sliding Window*):** Descrita como uma estratégia de otimização, seu objetivo é encontrar a solução globalmente ótima mesmo com a incerteza de dados futuros por meio de seu funcionamento dinâmico: a janela se expande à medida que novos pontos são recebidos e se contrai (gerando um resultado) apenas quando um "ponto de convergência" correto é identificado. A principal vantagem é sua adaptabilidade à complexidade da malha viária, mas sua desvantagem teórica é a ausência de uma garantia sobre o tamanho máximo da janela, o que poderia levar a atrasos de saída em cenários extremos.

III. **Janela Deslizante Variável Limitada (BVSW - *Bounded Variable Sliding Window*):** Uma modificação do VSW que introduz um limite superior para o tamanho da janela. No caso, este limite é atingido antes da identificação de um ponto de convergência, o algoritmo é forçado a emitir a solução mais provável até aquele momento. Essa abordagem garante um tempo de resposta máximo (pior caso), resolvendo a principal desvantagem do VSW, mas ao custo de, potencialmente, gerar uma solução que não é ótima.

Os experimentos conduzidos por Goh *et al.* (2012) demonstraram que ambas as abordagens dinâmicas, VSW e BVSW, superaram a estratégia tradicional FSW em termos de acurácia e latência de saída. Isso consolidou as janelas de tamanho variável como uma técnica de ponta para algoritmos de *map matching online*.

A Figura 11 ilustra o processo de expansão e contração da técnica VSW, aplicada no algoritmo *Online* HMM (OHMM) de Goh *et al.* (2012).

Figura 11 – Ilustração do mecanismo da Janela Deslizante Variável (VSW)



Fonte: Adaptado de Goh *et al.* (2012).

Além da definição do tamanho da janela de análise, uma estratégia complementar frequentemente adotada para viabilizar o *map matching online* é a execução incremental por lotes (*batch processing*). Essa abordagem consiste em agrupar um número limitado de pontos consecutivos antes de submetê-los ao algoritmo de correspondência, em vez de processar cada observação de forma estritamente individual. O trabalho seminal de Newson e Krumm (2009) já indicava que algoritmos originalmente concebidos para processamento *offline* podem ser adaptados ao contexto *online* por meio da execução em pequenos lotes, permitindo a reutilização de modelos probabilísticos robustos sem a necessidade de aguardar a trajetória completa.

Do ponto de vista operacional, o processamento por *batches* introduz um compromisso fundamental entre latência de processamento e estabilidade da solução: lotes reduzidos minimizam o tempo de resposta (ideal para aplicações em tempo real), enquanto lotes maiores fornecem um contexto local mais rico, aumentando a acurácia da correspondência. Na prática, essa estratégia é frequentemente combinada com técnicas de janela deslizante, nas quais cada

lote representa uma atualização parcial da janela de análise. Isso confere ao sistema um controle explícito sobre o volume de dados processados por iteração e, consequentemente, sobre o atraso máximo na saída do sistema.

2.4 *Benchmark*

Esta seção apresenta os fundamentos do *benchmarking* como um método científico de avaliação. Primeiramente, é apresentada a definição formal e as propriedades de um *benchmark* de qualidade na subseção 2.4.1. Em seguida, aborda-se o papel do *benchmarking* na Engenharia de Software Empírica e o conceito de espaço de características na subseção 2.4.2. Posteriormente, discutem-se as abordagens de avaliação (*Black, White* e *Grey Box*) na subseção 2.4.3. Por fim, a subseção 2.4.4 detalha a arquitetura lógica de um experimento de *benchmark*, composta por sistemas sob teste, *workloads* e a *benchmark suite*.

2.4.1 Definição

Formalmente definido por Kistowski *et al.* (2015), um *benchmark* é um artefato padronizado para a avaliação comparativa de sistemas ou componentes que competem entre si de acordo com determinadas características, como desempenho, eficiência ou segurança. Nesse contexto, ao definir medições padronizadas que produzem resultados objetivos e comparáveis, *benchmarks* são amplamente utilizados para comparar ferramentas, técnicas ou algoritmos (Hasselbring, 2021).

Para cumprir sua função de fornecer uma avaliação fidedigna e útil, a simples realização de medições não é suficiente. Segundo Kistowski *et al.* (2015), a qualidade e a aceitação de um *benchmark* dependem de cinco propriedades fundamentais:

I. **Relevância**: O comportamento exercitado pelo *benchmark* deve ser estritamente correlacionado aos comportamentos de interesse para os usuários dos resultados.
II. **Reprodutibilidade**: Capacidade de produzir resultados consistentes quando o *benchmark* é executado sob a mesma configuração experimental.
III. **Justiça**: Garantia de que diferentes sistemas ou configurações concorram com base em seus próprios méritos, sem limitações artificiais ou favorecimentos indevidos.
IV. **Verificabilidade**: Capacidade de prover confiança de que os resultados obtidos são precisos e condizentes com o comportamento real do sistema avaliado.

V. **Usabilidade**: Ausência de obstáculos técnicos excessivos que impeçam usuários de executar o *benchmark* em seus próprios ambientes de teste.

Em conjunto, essas propriedades estabelecem os requisitos mínimos para que um *benchmark* possa ser considerado um instrumento científico válido de avaliação comparativa. Ao assegurá-las, fornece-se uma base confiável para a análise e a interpretação dos resultados obtidos. Isso fundamenta as decisões de projeto da arquitetura experimental adotada neste trabalho e orientam a definição da *benchmark suite*, dos *workloads* e das métricas empregadas nas etapas subsequentes.

2.4.2 Benchmark na Engenharia de Software Empírica

O *benchmarking* é amplamente reconhecido como um método central na Engenharia de Software Empírica, sendo caracterizado como uma forma de experimentação controlada que permite comparar sistematicamente soluções sob condições bem definidas (Hasselbring, 2021). Nesse contexto, um *benchmark* estabelece um ambiente padronizado no qual diferentes sistemas, algoritmos ou abordagens podem ser avaliados de maneira justa, reprodutível e mensurável.

Além disso, a criação e a adoção ampla de um *benchmark* em uma determinada área de pesquisa tendem a acelerar o progresso técnico. Conforme argumentado por Bartz-Beielstein *et al.* (2020), a existência de *benchmarks* consolidados é frequentemente interpretada como um indicativo de maturidade científica, pois fornece uma base comum para comparação, validação e evolução incremental de soluções.

Para que os resultados obtidos por meio de *benchmarking* sejam considerados válidos e generalizáveis, é essencial que o conjunto de instâncias avaliadas ofereça cobertura adequada do chamado Espaço de Características do problema. Esse espaço corresponde ao conjunto de atributos mensuráveis que influenciam o desempenho das soluções avaliadas. Segundo Bartz-Beielstein *et al.* (2020), a diversidade dessas instâncias deve abranger uma ampla gama de valores, de modo a evitar vieses experimentais e aumentar a probabilidade de que os resultados reflitam cenários reais com diferentes níveis de dificuldade.

No contexto deste trabalho, o Espaço de Características é definido por propriedades intrínsecas das trajetórias utilizadas no *benchmark*, tais como o comprimento total da trajetória, a densidade temporal dos pontos, a presença de curvas acentuadas e variações sistemáticas da taxa de amostragem. A consideração explícita desses fatores permite avaliar de forma mais abrangente o comportamento das ferramentas de *map matching* sob diferentes condições operacionais.

2.4.3 Abordagens de Avaliação: Black, White e Grey Box

A profundidade da avaliação de um sistema depende do nível de acesso ao seu funcionamento interno. Segundo Khan e Khan (2012), as abordagens de avaliação podem ser classificadas em:

I. ***Black Box***: O sistema é tratado como uma caixa-preta, sendo avaliadas apenas suas entradas e saídas, sem conhecimento do código-fonte ou da lógica interna.
II. ***White Box***: A avaliação explora a estrutura interna do sistema, utilizando o código-fonte e o fluxo de controle para derivar casos de teste e analisar o comportamento interno.
III. ***Grey Box***: Abordagem híbrida na qual o avaliador possui conhecimento dos algoritmos e estruturas internas do sistema, mas realiza os testes por meio das interfaces externas disponibilizadas.

Neste trabalho, adota-se uma perspectiva de avaliação *Grey Box*, uma vez que o conhecimento prévio sobre os algoritmos de *map matching* e o acesso ao código-fonte das ferramentas *Open Source Software* / *Software* de Código Aberto (OSS) permitem projetar configurações experimentais mais precisas, embora a coleta de métricas ocorra exclusivamente por meio de interfaces de serviço.

2.4.4 Arquitetura do Experimento

Para operacionalizar a avaliação, a arquitetura lógica de um experimento de *benchmark* é decomposta nos seguintes componentes fundamentais:

I. **SUT (*System Under Test*)**: O sistema ou componente alvo da avaliação, que recebe as entradas e produz os resultados a serem mensurados. No contexto deste trabalho, os SUTs são as ferramentas de código aberto de *map matching* selecionadas.
II. **Matriz de Configuração**: O arranjo sistemático de parâmetros controlados que define a amplitude e a granularidade do espaço experimental. No contexto de *map matching*, a matriz de configuração pode incluir, por exemplo, o nível de ruído aplicado às trajetórias, a taxa de amostragem dos pontos GPS, o tamanho da janela deslizante e o raio de busca na malha viária (Newson; Krumm, 2009; Goh *et al.*, 2012; Rehrl *et al.*, 2018).
III. ***Workload* (Carga de Trabalho)**: Conjunto de dados e demandas de processamento submetidos ao sistema para simular, de forma controlada, o comportamento real. No contexto de correspondência de mapas, o *workload* é o conjunto de trajetórias a processar

e o padrão de submissão desses dados ao sistema, avaliado em diferentes configurações experimentais definidas pela matriz de configuração.

IV. ***Benchmark Suite***: A especificação formal e padronizada do experimento, que assegura a reprodutibilidade ao definir o que será medido e como. A suíte compreende a definição da carga de trabalho, da matriz de configuração, da escolha das métricas de eficiência e do protocolo de execução que garante condições experimentais equivalentes para todos os SUTs.

V. ***Benchmark Harness* (Infraestrutura de Execução)**: Artefato de *software* responsável por instanciar e executar de forma automatizada os experimentos do *benchmark*, coordenando a submissão da carga de trabalho, a coleta sistemática de métricas e a consolidação dos resultados obtidos. No contexto de *map matching*, o *harness* compreende o orquestrador de experimentos e a camada de interoperabilidade *client-side*, sendo encarregado de coordenar a execução da suíte de testes sobre os SUTs selecionados, emular o fluxo de dados GNSS e coletar métricas de acurácia, desempenho de ponta a ponta e consumo de recursos computacionais.

A decomposição do experimento em componentes, como sistema sob teste, carga de trabalho e parâmetros controlados, é consistente com abordagens consolidadas na literatura de *benchmarking*, nas quais a parametrização da carga de trabalho é utilizada para explorar diferentes propriedades de desempenho sob condições controladas (Maron; Fernandes, 2018).

3 TRABALHOS RELACIONADOS

Os trabalhos relacionados neste capítulo são todos centrados em *map matching* em diferentes contextos. O primeiro, presente na seção 3.1, apresenta um algoritmo de *map matching online* baseado em HMM intitulado OHMM, juntamente com as técnicas de janelas deslizantes já abordadas no Capítulo 2. O segundo, descrito na seção 3.2, elabora um guia detalhado do uso de ferramentas OSS de *map matching* no contexto de *Big Data*[1]. O terceiro trabalho, apresentado na seção 3.3, investiga o uso do simulador SUMO para a geração de trajetórias sintéticas com *ground truth*, estabelecendo um *benchmark* comparativo de algoritmos de *map matching* sob cenários controlados. O quarto trabalho, exibido na seção 3.4, estabelece um *benchmark* que compara uma implementação de *map matching* denominada GraphiumMM, que opera tanto no paradigma *online* quanto *offline*, com outra ferramenta. Por fim, o quinto trabalho, exposto na seção 3.5, contribui com a abordagem e implementação de um algoritmo *offline* e um *benchmark* que o compara com outras ferramentas OSS. Em suma, esses trabalhos representam avanços relevantes no campo do *map matching*. No entanto, eles apresentam limitações ou em termos de escopo, ou em métricas avaliadas, ou no paradigma abordado, ou na abrangência de avaliação. Essas questões são analisadas de forma comparativa na seção 3.6.

3.1 *Online map-matching based on Hidden Markov model for real-time traffic sensing applications*

Em Goh *et al.* (2012), visando encontrar um método para *map matching online* com um bom balanço entre tempo de resposta e acurácia, foram propostos o algoritmo OHMM e um *framework*[2] geral para a modelagem de algoritmos baseados em HMM *online*. O método foi validado por meio das métricas de acurácia e tempo de resposta ao executar o algoritmo sob dados de quatro trajetórias coletadas e não publicadas em rotas de ônibus em Singapura, sendo duas rurais e duas urbanas.

Como resultado, o algoritmo proposto, utilizando ou *variable-size sliding window* / janela deslizante de tamanho variável (VSW) ou *bounded variable-size sliding window* / janela deslizante de tamanho variável limitado (BVSW) superou abordagens tradicionais que usam *fixed-size sliding window* / janela deslizante de tamanho fixo (FSW) em acurácia e tempo de resposta. Em decorrência do resultado obtido, a abordagem de Goh *et al.* (2012) consolidou-se como

[1] Conjuntos de dados exacerbadamente grandes que são difíceis de processar por ferramentas e técnicas tradicionais
[2] Conjunto de regras, princípios ou diretrizes que fornecem uma base ou estrutura

estado da arte para abordagens markovianas em tempo real (Mattheis *et al.*, 2014), sendo utilizada em projetos OSS amplamente reconhecidos pela comunidade, como o Barefoot[3](Mattheis *et al.*, 2014).

Portanto, este trabalho se relaciona com o presente estudo de forma simbiótica e fundamental. Embora ambos se concentrem em *map matching online* via HMM, seus objetivos divergem: Goh *et al.* (2012) propõe um novo método, enquanto esta pesquisa avalia e compara, mediante a condução de um *benchmark*, o ecossistema de ferramentas existentes OSS. Uma similaridade entre os estudos é o uso de métricas de acurácia e temporais na avaliação.

3.2 *A Practical Guide to an Open-Source Map-Matching Approach for Big GPS Data*

O trabalho de Saki e Hagen (2022) busca encontrar alternativas às soluções comerciais que são caras e limitadas para grandes trajetórias históricas, como os serviços do Google Maps (GOOGLE LLC, 2025a) e do GraphHopper (GRAPHHOPPER, 2025). Para isso, os autores recorreram a opções de código aberto e escolheram uma ferramenta *offline* de alta adesão pela comunidade, no caso o OSS Valhalla[4] como opção viável e de baixo custo.

Para validar o Valhalla, foram elaborados e implementados um *framework* e uma arquitetura, ambos baseados em nuvem e com foco em *Big Data*. Este foi o ambiente aprovisionado na Amazon Web Services (AWS), no qual a ferramenta foi executada sobre uma amostra de aproximadamente 1,2 milhões de trajetórias de alta frequência provenientes de um conjunto de dados privado de viagens em Frankfurt am Main, na Alemanha. A avaliação do desempenho, com base em métricas de acurácia e temporais, resultou em uma taxa de sucesso de 95,2% nas correspondências realizadas, mostrando o poder e a eficiência da ferramenta.

Assim como o presente trabalho, o estudo mencionado avalia, com base em resultados empíricos, a viabilidade de ferramentas OSS para *map matching*, considerando métricas de acurácia e temporais. No entanto, os trabalhos divergem quanto ao foco e ao objetivo. Enquanto Saki e Hagen (2022) fornece um guia prático e uma validação para uma única ferramenta *offline*, este trabalho concentra-se em uma análise comparativa de múltiplas ferramentas e seu desempenho, especificamente, no paradigma *online*.

[3] https://github.com/bmwcarit/barefoot
[4] https://github.com/valhalla/valhalla

3.3 Benchmarking Trajectory Matchers with SUMO

Para mitigar a falta de dados de trajetórias com *ground truth*, Erdmann e Ebendt (2014) buscaram validar empiricamente a viabilidade do *Simulation of Urban MObility* / Simulação de Mobilidade Urbana (SUMO) na geração de dados sintéticos com *ground truth*. Neste trabalho, os autores argumentam que, por o SUMO ser um simulador e gerar trajetórias diretamente a partir da malha viária como fonte de verdade, pois a exata correspondência entre a trajetória em si e a malha é totalmente conhecida.

Nesse contexto, Erdmann e Ebendt (2014) estabelecem um *benchmark* comparativo de três algoritmos de *map matching*, avaliados sob quatro cenários sintéticos gerados com o SUMO [5]. Essa suíte foi elaborada a partir de trajetórias sintéticas geradas em duas malhas viárias privadas distintas, sob uma matriz de duas configurações: vazia e congestionada. A primeira é um cenário em que a velocidade máxima é a mesma encontrada na malha viária, análoga a um cenário sem trânsito. Já na segunda configuração, a velocidade máxima é multiplicada por um fator $r \in [0,2\ ,\ 1]$ para simular um cenário de lentidão. A avaliação foi conduzida por meio de métricas de acurácia baseadas na correspondência entre as sequências de segmentos inferidos e as rotas de referência.

As trajetórias são, posteriormente, degradadas por meio de subamostragem temporal e adição de ruído espacial amostrado de uma distribuição gaussiana bi-dimensional com desvio padrão de 5 metros, permitindo avaliar o impacto de erros de amostragem e de posicionamento sobre o desempenho dos algoritmos. A qualidade do *map matching* é mensurada por métricas de acurácia baseadas na correspondência entre as rotas inferidas e as rotas de referência, considerando tanto o número quanto o comprimento dos segmentos corretamente identificados.

Como a avaliação considera o comprimento das arestas corretamente e incorretamente associadas, a métrica em si equivale ao complemento do erro definido por Newson e Krumm (2009), isto é, $1 - NKError$, interpretável como um índice normalizado de qualidade da correspondência.

Embora o trabalho represente uma contribuição fundamental ao demonstrar a viabilidade do uso de dados sintéticos com *ground truth* para benchmarking de *map matching*, sua avaliação concentra-se exclusivamente no paradigma *offline*, considerando trajetórias completas e métricas de acurácia calculadas a posteriori. Aspectos cruciais para aplicações modernas, como processamento incremental, latência ponta-a-ponta e custo computacional sob restrições

[5] https://eclipse.dev/sumo/

de tempo real, não são abordados.

Além disso, o estudo foca na comparação de algoritmos específicos, sem discutir aspectos de engenharia de software relacionados à integração, automação e reprodutibilidade de ferramentas completas de *map matching* em ambientes controlados.

O presente trabalho adota o mesmo princípio metodológico de geração de dados sintéticos com *ground truth* via SUMO, porém amplia significativamente o escopo da avaliação ao investigar ferramentas de *map matching* operando no paradigma *online*. Para isso, são incorporadas métricas de desempenho temporal, uso de recursos computacionais e uma arquitetura de *benchmark* orientada à reprodutibilidade e instrumentação ponta-a-ponta, aproximando a análise das demandas reais de sistemas de *map matching* em produção.

3.4 *Optimization and Evaluation of a High-Performance Open-Source Map-Matching Implementation*

Também foi realizada uma avaliação de ferramentas OSS no trabalho de Rehrl *et al.* (2018). No referido trabalho, visando superar outras alternativas de código aberto, especialmente para baixos intervalos de amostragem entre 1 e 30 segundos de aferições GNSS, foi desenvolvida uma ferramenta de alta precisão e desempenho, baseada em uma abordagem geométrica e topológica, que funciona em ambos os paradigmas: *offline* e *online*. No caso do último, os autores mencionam que utilizaram a técnica de VSW proposta por Goh *et al.* (2012). A solução elaborada foi intitulada GraphiumMM[6] e implementada como um *plugin* Neo4j[7] para o *framework* – de armazenamento distribuído de grafos de transporte – Graphium[8], o que lhe confere capacidade de processamento rápido de grafos e roteamento.

Para avaliar tal artefato, compararam-no com outra ferramenta de referência de código aberto, nativamente online, o Barefoot (Mattheis *et al.*, 2014), por meio de métricas de acurácia e de tempo, executando-os sob uma carga de trabalho de um conjunto de dados publicado de 32 trajetórias, originalmente aferidas com uma frequência de 1 *hz*, mas também re-amostradas em outras 10 frequências maiores, entre 1 e 120 segundos, em ambos os modos: pós-processamento e incremental.

Os resultados da avaliação comparativa posicionam o GraphiumMM com vantagens significativas em casos de intervalos de amostragem entre 1 e 15 segundos em ambos os para-

[6] https://github.com/graphium-project/graphium-neo4j
[7] GDBMS de código aberto. Disponível em: https://neo4j.com
[8] https://github.com/graphium-project/graphium

digmas. Entretanto, ressalta-se que o Barefoot apresenta um desempenho melhor em termos de qualidade da correspondência no modo *online* e tende a associar uma porcentagem maior de pontos à trajetória. Diante dessas revelações, os autores concluem que não há uma ferramenta universalmente superior e que a escolha ideal depende de um compromisso (*trade-off*) entre acurácia e velocidade.

Em síntese, o trabalho de Rehrl *et al.* (2018) relaciona-se com o presente trabalho por estabelecer um *benchmark* sistemático de ferramentas OSS de *map matching* no paradigma *online*, com base em métricas de acurácia e tempo. Em contrapartida, a principal diferença reside no escopo da avaliação e na forma de construção da carga de trabalho: enquanto Rehrl *et al.* (2018) se concentra na comparação de duas ferramentas sob um único conjunto de dados reais, o presente trabalho estabelece um *benchmark* mais abrangente, avaliando múltiplas soluções do ecossistema de código aberto a partir de um conjunto de dados sintético com *ground truth* totalmente conhecido. Essa abordagem permite maior controle experimental, a exploração sistemática de diferentes configurações de processamento *online* e a incorporação conjunta de métricas de acurácia, desempenho temporal e uso de recursos computacionais.

3.5 *Open source map matching with Markov decision processes: A new method and a detailed benchmark with existing approaches*

Semelhante ao trabalho apresentado na seção anterior, Wöltche (2023) propõe uma abordagem e a confronta com outras ferramentas de código aberto consolidadas pela comunidade. Porém, diferentemente do *benchmark* de Rehrl *et al.* (2018), Wöltche (2023) confronta sua abordagem com o ecossistema de ferramentas de forma abrangente, mediante um *extenso benchmark* sistemático focado no paradigma *offline*.

No trabalho de Wöltche (2023), os autores buscaram um novo método de mapeamento de trajetórias mais robusto a ruídos e *outliers* e apresentaram uma abordagem própria baseada em HMM, disponibilizando-a como ferramenta *offline* OSS, nomeada Map Matching 2[9].

A validação da abordagem proposta foi conduzida por um *benchmark* extensivo que a confronta com seis outras ferramentas OSS, utilizando quatro conjuntos de dados públicos – um deles publicado pelo próprio autor – que representam cenários diversos, típicos e desafiadores. A comparação foi fundamentada em três métricas — qualitativas, temporais e de uso de recursos

[9] https://github.com/iisys-hof/map-matching-2

computacionais (processador e memória) — e avaliou variações nas configurações de pré-processamento da malha viária. O *benchmark* resultante revela que a nova abordagem superou as soluções existentes nas três métricas comparadas.

O presente trabalho e o *benchmark* de Wöltche (2023) compartilham o objetivo central de criar um *benchmark* sistemático para ferramentas de *map matching* OSS, utilizando múltiplas fontes de dados e os mesmos tipos de métricas de avaliação. Já a distinção crucial entre os dois diz respeito ao paradigma analisado: o estudo de Wöltche (2023) foca em *softwares* que operam no modo *offline*, enquanto este estudo se dedica àqueles que operam no modo *online*. Assim, o trabalho de referência serve como um "espelho" metodológico e deixa em aberto a análise comparativa para o domínio do paradigma de *map matching online*, evidenciando a relevância do estudo atual.

3.6 Análise comparativa

Por mais que todos os trabalhos descritos acima sejam correlatos no contexto de *map matching*, nenhum deles, isoladamente ou em conjunto, fornece subsídios suficientes para uma escolha informada de uma ferramenta OSS voltada para aplicações em tempo real. A literatura atual apresenta um panorama claro: existem validações de abordagens individuais, como as de Goh *et al.* (2012) e Saki e Hagen (2022), bem como *benchmarks* baseados em dados reais ou sintéticos, como os de Erdmann e Ebendt (2014), Rehrl *et al.* (2018) e Wöltche (2023). Contudo, observa-se uma lacuna recorrente: essas avaliações concentram-se majoritariamente no paradigma *offline*, apresentam um escopo restrito de ferramentas e métricas, ou não abordam de forma sistemática os custos temporais e computacionais associados ao processamento incremental.

É nesse contexto que a presente pesquisa se insere, ao propor um *benchmark* sistemático voltado à avaliação de ferramentas de *map matching* no paradigma *online*. Diferentemente de estudos anteriores, o trabalho adota um conjunto de dados sintético gerado por simulação, com *ground truth* totalmente conhecido, permitindo controle rigoroso dos cenários avaliados e medições consistentes de acurácia. Além disso, a análise incorpora métricas temporais, de uso de recursos computacionais e de qualidade da correspondência, aproximando a avaliação das demandas reais de sistemas de *map matching online*.

O Quadro 1 elucida visualmente a lacuna preenchida, bem como as principais semelhanças e diferenças dos trabalhos relacionados mencionados neste capítulo.

Quadro 1 – Comparação entre os trabalhos relacionados e o trabalho proposto.

Referência	**Natureza**	**Escopo**	**Tipo**	**Métricas**	**Conjuntos de Dados**
Goh *et al.* (2012)	Prova de Conceito	Implementação Própria (Não publicada)	*Online*	Acurácia Temporal	Quatro trajetórias de ônibus em Singapura (Não publicado)
Saki e Hagen (2022)	Estudo de Caso	Valhalla	*Offline*	Acurácia Temporal	18 milhões de viagens em Frankfurt am Main (Privado)
Erdmann e Ebendt (2014)	*Benchmark*	Implementações Próprias (Não publicadas)	*Offline*	Acurácia Estrutural	Trajetórias sintéticas com *ground truth* geradas via SUMO (Malhas privadas)
Rehrl *et al.* (2018)	*Benchmark*	Barefoot GraphiumMM	*Online* e *Offline*	Acurácia Temporal	32 trajetórias na Áustria[1]
Wöltche (2023)	*Benchmark*	Barefoot FMM GraphHopper MapMatching2 OSRM pgMapMatching Valhalla	*Offline*	Acurácia Temporal Recursos	Viagem em Seattle[2] Viagem em Melbourne[3] trajetórias Worldwide[4] trajetórias Hof[5]
Este Trabalho	**Benchmark**	**Barefoot GraphiumMM GraphHopper OSRM**	***Online***	**Acurácia Temporal Recursos**	**trajetórias sintéticas em O'Hare, Chicago**

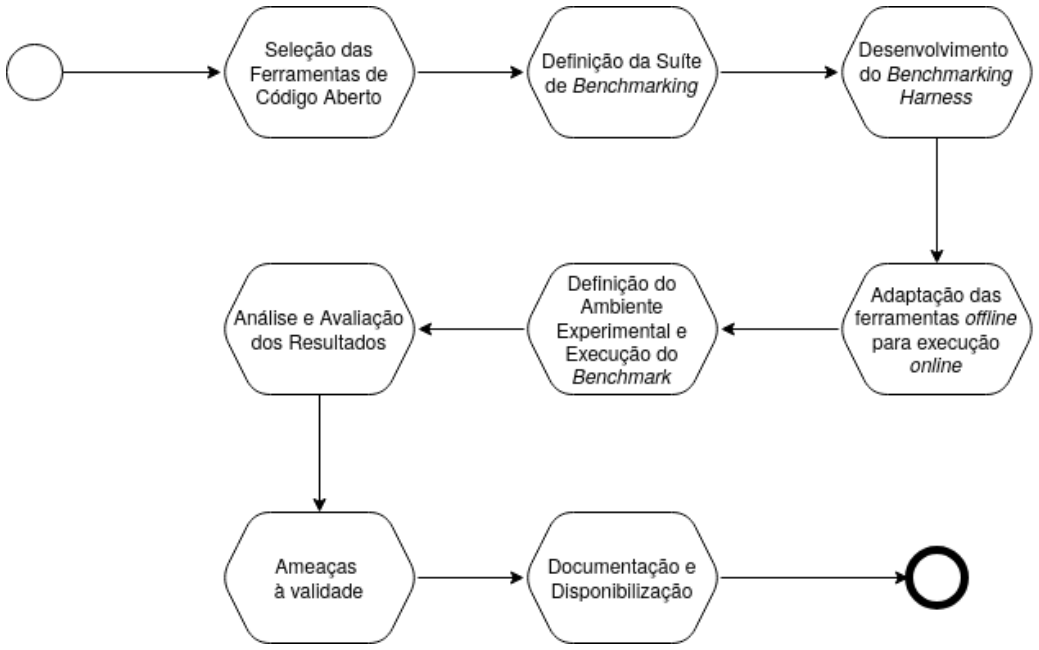
Notas: [1] Rehrl *et al.* (2018); [2] Newson e Krumm (2009); [3] Li *et al.* (2014); [4] Kubička *et al.* (2015); [5] Wöltche (2023).

Fonte: Elaborado pelo autor.

4 PROCEDIMENTOS METODOLÓGICOS

A metodologia adotada baseia-se no paradigma de *benchmark* experimental descrito na literatura de Engenharia de Software Empírica (Kistowski *et al.*, 2015; Hasselbring, 2021). O processo metodológico foi estruturado em etapas sequenciais, resumidas na Figura 12. As seções seguintes estão organizadas da seguinte maneira: a seção 4.1apresenta o processo de seleção e adequação experimental das ferramentas de *map matching* de código aberto; a seção 4.2 estabelece a suíte do *benchmark*, contemplando carga de trabalho, matriz de configuração e métricas; a seção 4.3 descreve o *benchmark harness* e sua arquitetura; a seção 4.4 detalha a adaptação das ferramentas *offline* para execução *online* em micro-lotes; a seção 4.5 caracteriza o ambiente experimental e a condução da campanha; por fim, as seções 4.6, 4.7 e 4.8 abordam, respectivamente, a avaliação dos resultados coletados, as ameaças à validade e os aspectos de documentação e reprodutibilidade do estudo.

Figura 12 – Sequência de atividades para execução do benchmark



Fonte: Elaborado pelo autor.

4.1 Seleção das Ferramentas de Código Aberto

Esta seção descreve o processo de definição do portfólio de ferramentas de *map matching* de código aberto utilizadas neste *benchmark*. Inicialmente, a subseção 4.1.1 detalha a metodologia adotada para o levantamento, filtragem e ranqueamento das ferramentas, incluindo

os critérios de elegibilidade, priorização por paradigma e análise de adoção e vitalidade dos projetos. Em seguida, a subseção 4.1.2 apresenta individualmente as ferramentas selecionadas, descrevendo suas características arquiteturais e algorítmicas, bem como as adequações experimentais realizadas para viabilizar a extração dos metadados necessários à avaliação comparativa.

4.1.1 Metodologia de Seleção das Ferramentas

De início, a seleção das ferramentas que compõem o escopo deste *benchmark* seguiu um processo de múltiplas etapas, projetado para identificar um conjunto representativo do estado da arte em *map matching* OSS, justificando, dessa forma, a relevância do *benchmark*. O processo combina um filtro de elegibilidade inicial com um ranqueamento subsequente, conforme detalhado a seguir:

1. **Levantamento e Filtro de Elegibilidade:** Realizou-se um levantamento de ferramentas na literatura técnica e em repositórios públicos. Para garantir a validade interna do experimento, as ferramentas deveriam atender cumulativamente aos seguintes critérios de exclusão:
    a) **Licença Livre:** O código-fonte deve ser distribuído sob uma licença *Free and Open-Source Software* / Software Livre e de Código Aberto (FOSS) –que permite uso e modificação– como MIT[1], GPL-3[2], BSD-3[3] ou Apache 2.0[4].
    b) **Funcionalidade Explícita:** A ferramenta deve ter como um de seus principais recursos a funcionalidade de *map matching*.
    c) **Manutenção Mínima:** O projeto deve demonstrar atividade, tendo recebido alguma atualização em seu repositório, seja um *commit* ou *release* nos últimos 5 anos.
    d) **Viabilidade de Automação:** Exigiu-se a presença de interfaces de comunicação padrão (HTTP ou gRPC) e operação autônoma (*standalone*). Este critério é fundamental para a construção de um ambiente de testes reproduzíveis e para a mitigação de latências externas não controladas.
2. **Priorização por Paradigma:** Dentre as ferramentas elegíveis, foi dada prioridade àquelas que oferecem suporte nativo ao paradigma *online*, por serem o foco principal deste estudo.
3. **Classificação por Adoção na Comunidade:** As ferramentas elegíveis foram ranqueadas

[1] https://opensource.org/license/mit
[2] https://www.gnu.org/licenses/gpl-3.0.pt-br.html
[3] https://opensource.org/license/bsd-3-clause
[4] https://www.apache.org/licenses/LICENSE-2.0

conforme sua popularidade e adoção, utilizando o número de estrelas no GitHub como métrica indicativa de confiança e maturidade do ecossistema (Borges; Valente, 2018).

4. **Análise de Vitalidade e Usabilidade:** Como validação final, avaliou-se a qualidade da documentação técnica. A disponibilidade de imagens oficiais em contêineres Docker[5] foi considerada um critério decisivo por assegurar o isolamento do ambiente e a eliminação de variáveis externas relacionadas a conflitos de dependências do sistema operacional. Isso garantiu maior isolamento e reprodutibilidade do *benchmark*.

Ao final desse processo, concluído em 9 de julho de 2025, foram selecionadas dez ferramentas. Elas foram apresentadas na Tabela 1 inicialmente organizadas conforme o paradigma de execução, listando-se primeiramente as ferramentas de caráter *online* e, em seguida, as de caráter *offline*, separadas por uma linha distintiva. Posteriormente, dentro de cada grupo, as ferramentas foram ordenadas em função do número de estrelas recebidas no GitHub.

Tabela 1 – Levantamento de Ferramentas OSS de *Map Matching* Elegíveis

Ferramenta	Paradigma Nativo	Licença	Estrelas (GitHub)	Ano de Início	Última Atividade	Suporte Docker
Barefoot	*Online e Offline*	Apache-2.0	675	2015	2021	✓
GraphiumMM	*Online* e *Offline*	GPL-3	14	2017	2021	✓
MaMa	*Online*	MIT	12	2022	2023	✓
OSRM	*Offline*	BSD-2	6.8k	2010	2025	✓
GraphHopper	*Offline*	Apache-2.0	5.8k	2012	2025	✓
Valhalla	*Offline*	MIT	4.8k	2014	2025	✓
FMM	*Offline*	Apache-2.0	945	2017	2024	✓
pgMapMatch	*Offline*	GPL-3	81	2018	2023	—
MapMatching2	*Offline*	AGPL-3	64	2021	2025	✓
Horizon	*Offline*	Apache-2.0	55	2019	2025	✓

Fonte: Elaborado pelo autor.

Com base no levantamento, foram selecionadas tomando como referência a Tabela 1 as duas primeiras ferramentas nativamente *online* e as duas primeiras *offline*, formando um portfólio de **quatro ferramentas**, exibido na tabela Tabela 2.

Tabela 2 – Portfólio Final de Ferramentas Selecionadas para o Benchmark

Ferramenta	Paradigma Nativo	Licença	Estrelas (GitHub)
Barefoot	*Online e Offline*	Apache-2.0	675
GraphiumMM	*Online* e *Offline*	GPL-3	14
OSRM	*Offline*	BSD-2	6.8k
GraphHopper	*Offline*	Apache-2.0	5.8k

Fonte: Elaborado pelo autor.

As ferramentas selecionadas foram submetidas a uma etapa de adequação funcional

[5] https://www.docker.com

visando viabilizar a extração dos dados necessários ao cálculo das métricas de acurácia baseadas nos identificadores das arestas do grafo viário, neste caso, referentes à malha do *OpenStreetMap* (OSM). A apresentação individual das ferramentas e a descrição detalhada das adequações realizadas encontram-se na subseção 4.1.2.

4.1.2 Apresentação e Adequação Experimental das Ferramentas

Esta subseção apresenta, de forma individual, as ferramentas selecionadas para compor o *benchmark*, descrevendo suas características de funcionamento, arquitetura e paradigma de processamento. Também são detalhadas as adequações experimentais realizadas para viabilizar a extração dos metadados necessários à avaliação comparativa. Em todos os casos, buscou-se preservar integralmente o comportamento algorítmico original das ferramentas, utilizando suas configurações padrão sempre que possível. As únicas modificações realizadas dizem respeito à persistência e à exposição dos identificadores das arestas do OSM, os quais são essenciais para o cálculo das métricas de avaliação definidas neste trabalho. As ferramentas são apresentadas na seguinte ordem: Barefoot (subseção 4.1.2.1), GraphiumMM (subseção 4.1.2.2), OSRM (subseção 4.1.2.3) e Graphhopper (subseção 4.1.2.4). Em seguida, a subseção 4.1.2.5 discute a representatividade da seleção perante o estado da arte de ferramentas OSS de *map matching*.

Todas as adaptações, arquivos de configuração e demais artefatos experimentais utilizados foram disponibilizados no repositório oficial do projeto, assegurando transparência, reprodutibilidade e a possibilidade de verificação independente dos experimentos conduzidos.

4.1.2.1 Barefoot

O Barefoot[6] é uma ferramenta de *map matching* nativamente orientada para o paradigma *online*, oferecendo também suporte ao processamento *offline*. Sua arquitetura é distribuída e composta por três serviços principais. O primeiro é responsável pelo *map matching online*, o segundo executa o processamento *offline* e o terceiro consiste em um servidor de mapas implementado como um Sistema de Gerenciamento de Banco de Dados (SGBD) PostgreSQL[7] com PostGIS[8], utilizado para a construção e o armazenamento da malha viária.

No modo *offline*, a comunicação entre os componentes ocorre por meio de requisições HTTP. Já no modo *online*, a ferramenta utiliza comunicação baseada em *sockets* com o auxílio

[6] https://github.com/bmwcarit/barefoot
[7] https://www.postgresql.org/
[8] https://postgis.net/

do ZeroMQ[9], um mecanismo de mensageira *brokerless*. Essa escolha arquitetural tende a reduzir a latência de comunicação e é compatível com cenários de alta taxa de amostragem e grande volume de veículos, sendo, em princípio, adequada para aplicações em tempo quase real.

Do ponto de vista algorítmico, o Barefoot é inspirado na abordagem proposta por Goh *et al.* (2012), conhecida como *Online Hidden Markov Model* (OHMM). Essa abordagem estende o modelo clássico de *map matching* baseado em *Modelos Ocultos de Markov*, originalmente proposto por Newson e Krumm (2009), por meio de um algoritmo de janela deslizante que permite a atualização incremental das estimativas à medida que novas observações são recebidas.

Durante a adequação experimental ao *benchmark*, verificou-se que, em sua configuração padrão, o Barefoot não incluía explicitamente os identificadores das arestas do OSM no *payload* de saída. Para viabilizar a avaliação comparativa, foi realizada uma modificação mínima no código-fonte com o objetivo de incluir esses identificadores nos resultados do *map matching*. Essa modificação não altera o comportamento algorítmico nem impacta o desempenho computacional da ferramenta. A versão customizada utilizada neste trabalho encontra-se disponível em https://github.com/JoseEdSouza/barefoot.

4.1.2.2 GraphiumMM

O GraphiumMM[10] é uma ferramenta de *map matching* nativamente orientada ao paradigma *online*, oferecendo também suporte ao processamento *offline*. Diferentemente das demais ferramentas analisadas, o GraphiumMM é implementado como um *plugin* do *Neo4j*, um sistema gerenciador de banco de dados orientado a grafos.

A ferramenta é distribuída como um artefato no formato *JAR*, sendo executada diretamente no mesmo ambiente de execução do banco de dados. Essa integração reduz a necessidade de serialização entre a camada de persistência e a lógica de processamento, o que pode contribuir para uma menor sobrecarga de comunicação, especialmente em cenários sensíveis à latência.

O GraphiumMM implementa um algoritmo de correspondência topológico-geométrico que, embora fundamentado no processamento *offline*, é adaptado para a operação *online* por meio da introdução de um manipulador de estado (*state handler*) (Rehrl *et al.*, 2018). Nessa arquitetura, que incorpora a estratégia de janela deslizante de Goh *et al.* (2012), a gestão do

[9] https://zeromq.org
[10] https://github.com/graphium-project/graphium-neo4j

fluxo de dados é delegada ao cliente (*client-side*) responsável por manter um *buffer* local de observações e enviar, a cada requisição, os novos pontos de localização juntamente com o identificador do último segmento considerado seguro. Dessa forma, o *state handler* utiliza esse contexto recebido para processar a continuidade da trajetória e garantir a consistência topológica incrementalmente.

Nesse contexto, durante a adequação experimental, constatou-se que os identificadores das arestas do OSM não eram preservados no processo de importação da malha viária. Para contornar essa limitação, foram realizadas modificações pontuais na ferramenta interna de importação, Graphium[11], do grafo e ajustes no *payload* de saída, de modo a persistir e expor esses identificadores. Essas alterações não impactam o comportamento algorítmico nem o desempenho da ferramenta. As versões adaptadas do Graphium e do GraphiumMM encontram-se, respectivamente, em https://github.com/JoseEdSouza/graphium e https://github.com/JoseEdSouza/graphium-neo4j

4.1.2.3 OSRM

O *Open Source Routing Machine* (OSRM)[12] é uma ferramenta amplamente estabelecida para realizar *map matching* no paradigma *offline*. Ela funciona de maneira *standalone* e disponibiliza uma API HTTP bem documentada, sendo largamente empregada tanto no setor industrial quanto na pesquisa acadêmica. O OSRM adota uma abordagem baseada em *Modelos Ocultos de Markov*, seguindo a formulação clássica apresentada por Newson e Krumm (2009).

O OSRM permite a personalização do processo de importação da malha viária por meio de *scripts* escritos na linguagem *Lua*[13], os quais são executados durante a etapa de pré-processamento dos dados do OSM. Essa funcionalidade foi utilizada para preservar os identificadores originais das arestas do grafo viário, possibilitando o rastreamento das correspondências entre o grafo de entrada e os resultados do *map matching*.

A adequação experimental foi realizada exclusivamente por meio de um *script Lua* customizado, sem a necessidade de modificações no código-fonte da ferramenta. O *script* utilizado neste trabalho encontra-se disponível em https://github.com/JoseEdSouza/map-matching/blob/main/tools/osrm/config/car_with_wayids.lua.

[11] https://github.com/graphium-project/graphium
[12] https://github.com/Project-OSRM/osrm-backend
[13] https://www.lua.org

4.1.2.4 GraphHopper

O *GraphHopper*[14] é uma solução consolidada e amplamente utilizada para *map matching* no paradigma *offline*. A ferramenta funciona de forma *standalone* e oferece suporte nativo à configuração avançada do processo de importação da malha viária. De maneira análoga ao OSRM, o GraphHopper adota uma abordagem baseada em *Modelos Ocultos de Markov*, seguindo a formulação apresentada por Newson e Krumm (2009).

Além disso, o GraphHopper incorpora adaptações inspiradas na abordagem proposta por Goh *et al.* (2012), conferindo maior flexibilidade ao tratamento das observações durante o processamento do *map matching*.

A ferramenta já disponibiliza arquivos de configuração que permitem preservar e expor os identificadores das arestas do OSM. Dessa forma, foi necessário apenas ajustar parâmetros internos de configuração, sem qualquer modificação no código-fonte ou impacto no desempenho computacional. O arquivo de configuração utilizado encontra-se disponível em https://github.com/JoseEdSouza/map-matching/blob/main/tools/graphhopper/config/config.yaml.

4.1.2.5 Representatividade do Portfólio Selecionado

Do ponto de vista do delineamento experimental, buscou-se compor um portfólio representativo do ecossistema atual de ferramentas OSS de *map matching*. As soluções selecionadas cobrem os paradigmas *online* e *offline*, indo de métodos clássicos baseados em *Modelos Ocultos de Markov* (Newson; Krumm, 2009) até extensões para processamento incremental em tempo quase real, como a abordagem de Goh *et al.* (2012). O portfólio inclui ferramentas consolidadas, amplamente usadas em contextos industriais e acadêmicos, e soluções mais experimentais que exploram arquiteturas distribuídas e integrações com bancos de dados orientados a grafos. Assim, a seleção foi construída para assegurar diversidade metodológica e maturidade tecnológica, permitindo que o *benchmark* produza comparações relevantes dentro dos limites de sua suíte e carga de trabalho.

4.2 Definição da Suíte de *Benchmarking*

Visando assegurar a relevância, a suíte deste *benchmark* foi concebida para ser representativa de cenários reais de uso. Nesse cenário, a subseção 4.2.1, a subseção 4.2.2

[14] https://github.com/graphhopper/graphhopper

e a subseção 4.2.3 tratam, respectivamente, da carga de trabalho (*workload*), da matriz de configuração e das métricas estabelecidas.

4.2.1 Carga de trabalho

Para a execução do *benchmark*, optou-se pelo uso de trajetórias sintéticas em detrimento de conjuntos de dados reais, visando garantir a disponibilidade de um *ground truth* exato e incontestável para o cálculo das métricas de acurácia. A carga de trabalho foi composta por trajetórias geradas por meio do simulador de mobilidade urbana SUMO, a partir da malha viária da região de O'Hare (Chicago, EUA). Essa região foi selecionada justamente por sua complexidade topográfica, caracterizada pela presença de múltiplos níveis de vias, segmentos de alta velocidade e curvas acentuadas, configurando um cenário multivariado que abrange tanto trajetórias simples quanto desafiadoras para o *map matching*.

O conjunto de dados elaborado é composto por **1.759 trajetórias**, às quais foi adicionado a/ruído aleatoriamente amostrado com a semente 42 de uma distribuição gaussiana bidimensional ($\mu = 0m, \sigma = 5m$) para emular as imprecisões típicas de sensores GPS em condições reais, conforme adotado em trabalhos anteriores na literatura (Erdmann; Ebendt, 2014; Guastella *et al.*, 2025). O detalhamento técnico do processo de extração da malha viária, da parametrização da simulação no SUMO e das etapas de pós-processamento dos dados é apresentado no Capítulo 5.

Entretanto, a execução integral do *benchmark* sobre todas as trajetórias geradas implicaria em um custo computacional elevado. Assim, foi adotada uma estratégia de amostragem estratificada baseada no comprimento total das trajetórias. Inicialmente, as trajetórias foram divididas em quatro estratos (quartis) de comprimento. Em seguida, foram amostradas aleatoriamente cinco trajetórias por estrato, totalizando **20 trajetórias**. Esse procedimento assegurou a diversidade estrutural da carga de trabalho, preservando trajetórias curtas, médias e longas. A definição dos intervalos de comprimento de cada quartil e a quantidade total de trajetórias por estrato são apresentadas na Tabela 3, enquanto as trajetórias efetivamente selecionadas e suas principais características encontram-se detalhadas na Tabela 4.

As trajetórias selecionadas foram originalmente amostradas com um intervalo de 500 ms e, posteriormente, em quatro taxas de amostragem (*sample rate*, *sr*): 1 s, 5 s, 10 s e 20 s, permitindo avaliar o impacto da frequência de observação no desempenho e na acurácia dos *System Under Test* / Sistema sob Testes (SUTs).

Tabela 3 – Definição dos quartis de comprimento das trajetórias e amostragem realizada

Quartil	Intervalo de Comprimento (m)	Total de Trajetórias	Amostradas
Q1	519.33 – 2239.33	440	5
Q2	2241.52 – 3231.63	440	5
Q3	3232.84 – 4636.51	439	5
Q4	4638.27 – 8973.01	440	5

Fonte: Elaborada pelo autor.

Tabela 4 – Trajetórias selecionadas por amostragem estratificada (quartis de comprimento)

Quartil	ID da Trajetória	Nº de Pontos	Duração (s)	Comprimento (m)
Q1	1139	307	153,0	1842,40
Q1	1552	159	77,0	1772,05
Q1	749	1458	728,5	1760,97
Q1	300	352	172,5	2183,80
Q1	1622	275	137,0	1458,30
Q2	384	472	235,0	2456,24
Q2	619	1617	809,5	3018,98
Q2	803	1694	846,5	2701,09
Q2	1077	778	385,0	2499,04
Q2	770	1451	722,0	2942,21
Q3	1	654	326,5	4228,74
Q3	557	1326	662,5	4147,44
Q3	349	3295	1647,0	3714,84
Q3	1356	3696	1845,0	4442,26
Q3	495	7278	3635,5	3816,84
Q4	479	1025	498,5	6516,28
Q4	117	429	212,0	5309,14
Q4	736	590	294,5	5713,89
Q4	876	588	285,5	4646,15
Q4	1423	543	271,0	5334,92

Fonte: Elaborada pelo autor.

Nos casos em que os SUTs realizam processamento incremental por lotes, as execuções foram conduzidas sob três tamanhos de lote (*batch size*, *bs*): 5, 10 e 30 pontos por lote. Para os SUTs que operam estritamente ponto a ponto, apenas as variações de taxa de amostragem foram consideradas.

Durante os experimentos, os pontos GNSS de cada trajetória foram emitidos na ordem temporal original. O intervalo entre pontos consecutivos foi definido pelo *timestamp* de cada observação, usando um comando de espera (*sleep*) proporcional à diferença entre pontos. Para tornar o *benchmark* computacionalmente viável sem perder a consistência temporal, introduziu-se um fator de aceleração (*speed factor*, sf), que divide o intervalo de espera, fazendo com que o emissor use um atraso de $\Delta t / sf$. Assim, a execução é acelerada mantendo a coerência temporal da trajetória, pois as ferramentas avaliadas usam os *timestamps* do *payload* dos pontos GNSS, e não o instante de chegada das mensagens, como referência para o processamento do *map matching*.

Dessa forma, a carga de trabalho do *benchmark* foi delineada para totalizar 240

execuções nos cenários com processamento em lotes (20 trajetórias $\times$ 4 taxas de amostragem $\times$ 3 tamanhos de lote) e 80 execuções nos cenários sem processamento em lotes (20 trajetórias, $\times$ 4 taxas de amostragem). Em todas as configurações, adotou-se um fator de velocidade $sf = 30$. A subseção 4.2.2 apresenta a consolidação dessas configurações.

4.2.2 Matriz de configuração experimental

A campanha experimental foi definida como um arranjo fatorial, no qual cada execução corresponde à combinação entre um sistema sob teste (SUT), uma trajetória da carga de trabalho e um conjunto de parâmetros de simulação. A Tabela 5 resume os fatores controlados e seus níveis.

Tabela 5 – Matriz de configuração experimental (fatores controlados e níveis)

Fator	**Símbolo**	**Níveis**
Sistema sob teste	—	Barefoot, GraphiumMM, OSRM, GraphHopper
Trajetória	—	20 trajetórias (amostragem estratificada por quartis)
Taxa de amostragem	sr	$1\,s$, $5\,s$, $10\,s$, $20\,s$
Tamanho de micro-lote	bs	5, 10, 30 pontos por lote (exceto Barefoot)
Fator de aceleração temporal	sf	30 (fixo em todas as execuções)

Fonte: Elaborada pelo autor.

Dessa forma, como GraphiumMM, OSRM e GraphHopper operam de modo incremental em lotes, a campanha experimental foi planejada para 240 execuções por ferramenta ($20 \times 4 \times 3$), repetidas três vezes, totalizando 720 execuções (240×3). Além disso, foram realizadas 80 execuções com o Barefoot (20×4), somando 800 execuções na campanha completa ($720 + 80$).

4.2.3 Métricas

As métricas coletadas nesta suíte seguem uma combinação de abordagens *black-box* e *grey-box*. As métricas *black-box* correspondem a medições de ponta a ponta, realizadas a partir da perspectiva do cliente (*client-side*), refletindo o comportamento observado por aplicações que consomem serviços de *map matching*. Já as métricas *grey-box* são obtidas no lado do servidor (*server-side*), explorando informações internas expostas pelos sistemas sob teste (SUTs) quando disponíveis, sem interferir na lógica de execução dos algoritmos. Em conjunto, essas duas perspectivas permitem organizar as métricas em três classes principais, que podem ser coletadas

no lado do cliente, no lado do servidor ou simultaneamente em ambos os contextos.

1. **Métricas de Acurácia**: Avaliam a qualidade da correspondência entre a trajetória inferida e a trajetória de referência (*ground truth*). As métricas de acurácia consideradas são:
   - **Precisão**;
   - **Revocação**;
   - **Taxa de Erro**;
   - ***NewsonKrummError***.
2. **Métricas Temporais**: Descrevem a eficiência, a capacidade operacional e a adequação das ferramentas ao paradigma *online*. As métricas temporais consideradas incluem:
   - **Latência de resposta** por ponto ou por lote, mensurada de ponta a ponta no lado do cliente;
   - **Tempo total de processamento**, quantificado no lado do servidor;
   - **Vazão de processamento**, definida como a quantidade de pontos processados por unidade de tempo (pontos por segundo), podendo também ser expressa em termos de extensão linear processada por unidade de tempo (quilômetros por segundo).
3. **Métricas de Uso de Recursos Computacionais**: Estas métricas quantificam o custo computacional associado à execução de cada sistema avaliado. Incluem-se, nesse conjunto:
   - **Utilização de CPU** ao longo do processamento, mensurada tanto no lado do cliente quanto no lado do servidor;
   - **Utilização de memória RAM** durante o processamento, igualmente mensurada no lado do cliente e no lado do servidor;
   - **Valores médios e valores de pico** de utilização dos recursos computacionais observados ao longo da execução das trajetórias.

Em conjunto, essas métricas possibilitam a avaliação integrada da acurácia do processo de *map matching*, do desempenho temporal e do custo computacional dos sistemas analisados, considerando-se o contexto do paradigma *online*.

4.3 Desenvolvimento do *Benchmarking Harness*

Considerando as propriedades de qualidade de um *benchmark* discutidas no Capítulo 2, este concebeu o *benchmark harness* como o principal mecanismo para assegurar sua aplicação prática. Em particular, o desenvolvimento do *harness* garante a reprodutibilidade dos experimentos, a usabilidade da infraestrutura proposta e a verificabilidade dos resultados obtidos.

No contexto deste estudo, o *benchmark harness* corresponde ao conjunto de artefatos de *software* responsável por instanciar a suíte definida, executar os SUTs sob condições controladas e coletar métricas de forma padronizada. Sua arquitetura foi projetada para viabilizar comparações justas entre ferramentas heterogêneas, incluindo soluções nativamente *online* e soluções originalmente *offline* que foram adaptadas para execução incremental. Ao nível operacional, o *harness* integra uma camada de interoperabilidade com interface comum e instrumentação de métricas no lado do cliente, bem como um orquestrador de *benchmark* responsável por automatizar o ciclo de vida dos experimentos, o gerenciamento de serviços por meio de contêineres e a consolidação dos resultados.

A arquitetura do *harness* é descrita a seguir com o modelo C4, explicitando seu contexto de execução e os principais artefatos executáveis. O diagrama de contexto na Figura 13 posiciona o *harness* em relação ao usuário e a sistemas externos, como a Docker Engine e o sistema de arquivos. Adicionalmente, o diagrama de contêineres, ilustrado na Figura 14, detalha os serviços executados durante o *benchmark*, incluindo o orquestrador em Python[15], os serviços de *map matching* e a pilha de observabilidade. Por fim, o fluxo de execução de um experimento é apresentado no diagrama de processo da Figura 15.

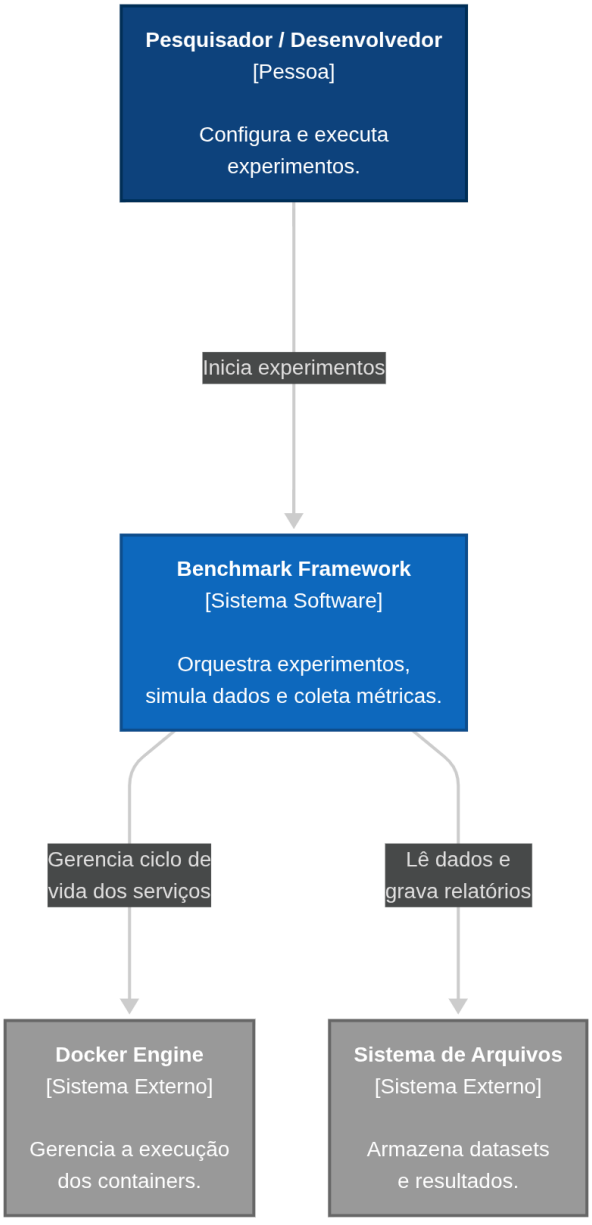
4.3.1 Interface comum e instrumentação no lado do cliente

Com o objetivo de viabilizar comparações justas e métricas consistentes entre ferramentas heterogêneas, foi desenvolvida uma biblioteca de interoperabilidade em Python, denominada `mmlib`. Essa biblioteca desempenha dois papéis centrais no *benchmark*. O primeiro é a padronização da interação com os SUTs por meio de uma interface comum. O segundo é a instrumentação sistemática de métricas de ponta a ponta no lado do cliente, de forma isonômica.

No que se refere à padronização, a `mmlib` abstrai o cliente de cada SUT como um *matcher*, definido em duas famílias de interfaces que contemplam, respectivamente, os paradigmas *offline* e *online*. A primeira corresponde a uma interface síncrona para processamento em lote, enquanto a segunda expõe uma interface assíncrona baseada em fluxo (*streaming*). Essa abstração permitiu uniformizar como as trajetórias são submetidas aos diferentes sistemas, independentemente de seus protocolos, formatos de entrada ou estratégias internas de processamento. A hierarquia de classes que define essas interfaces, bem como o relacionamento entre suas implementações concretas e adaptadores, é apresentada na Figura 16.

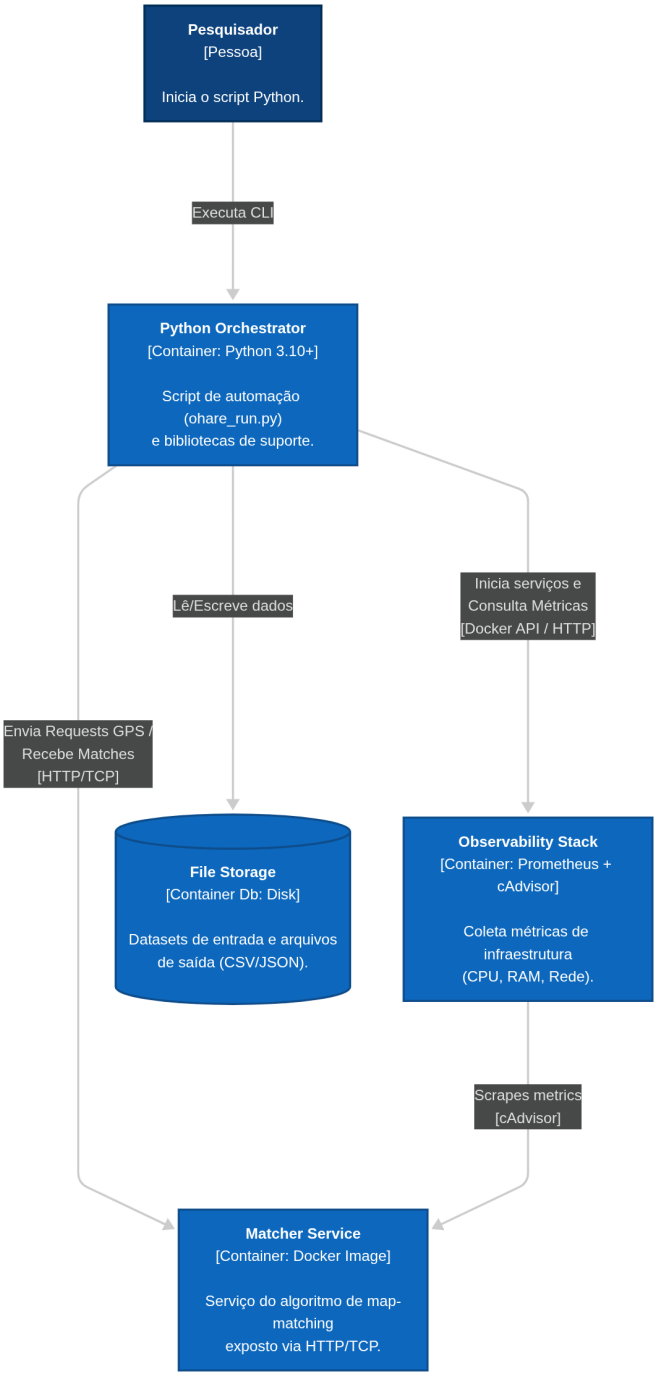
[15] https://www.python.org/

Figura 13 – Diagrama de contexto (C4) do *benchmark harness*.



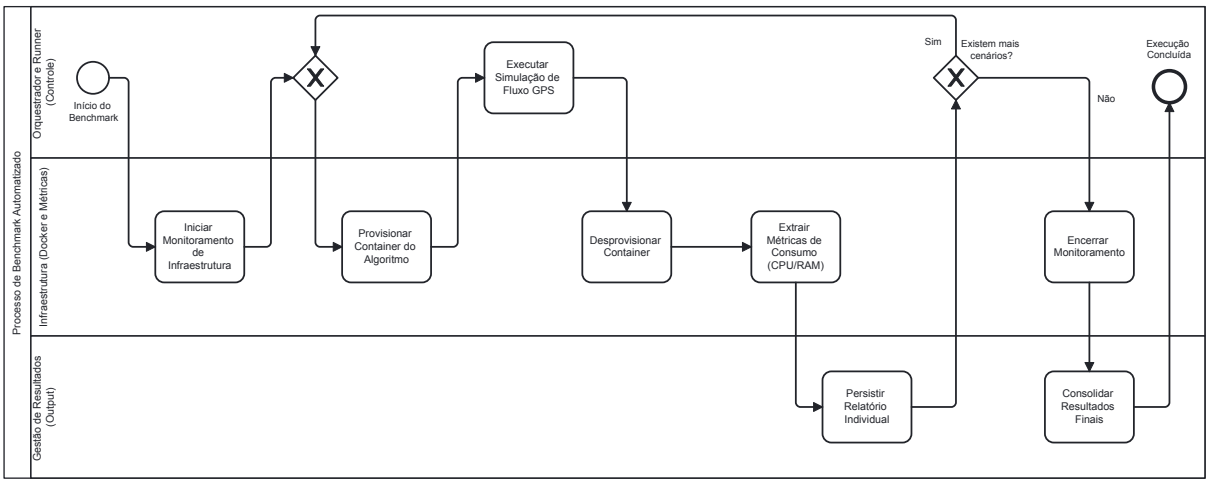
Fonte: Elaborado pelo autor.

Figura 14 – Diagrama de containers (C4) do *benchmark harness*.



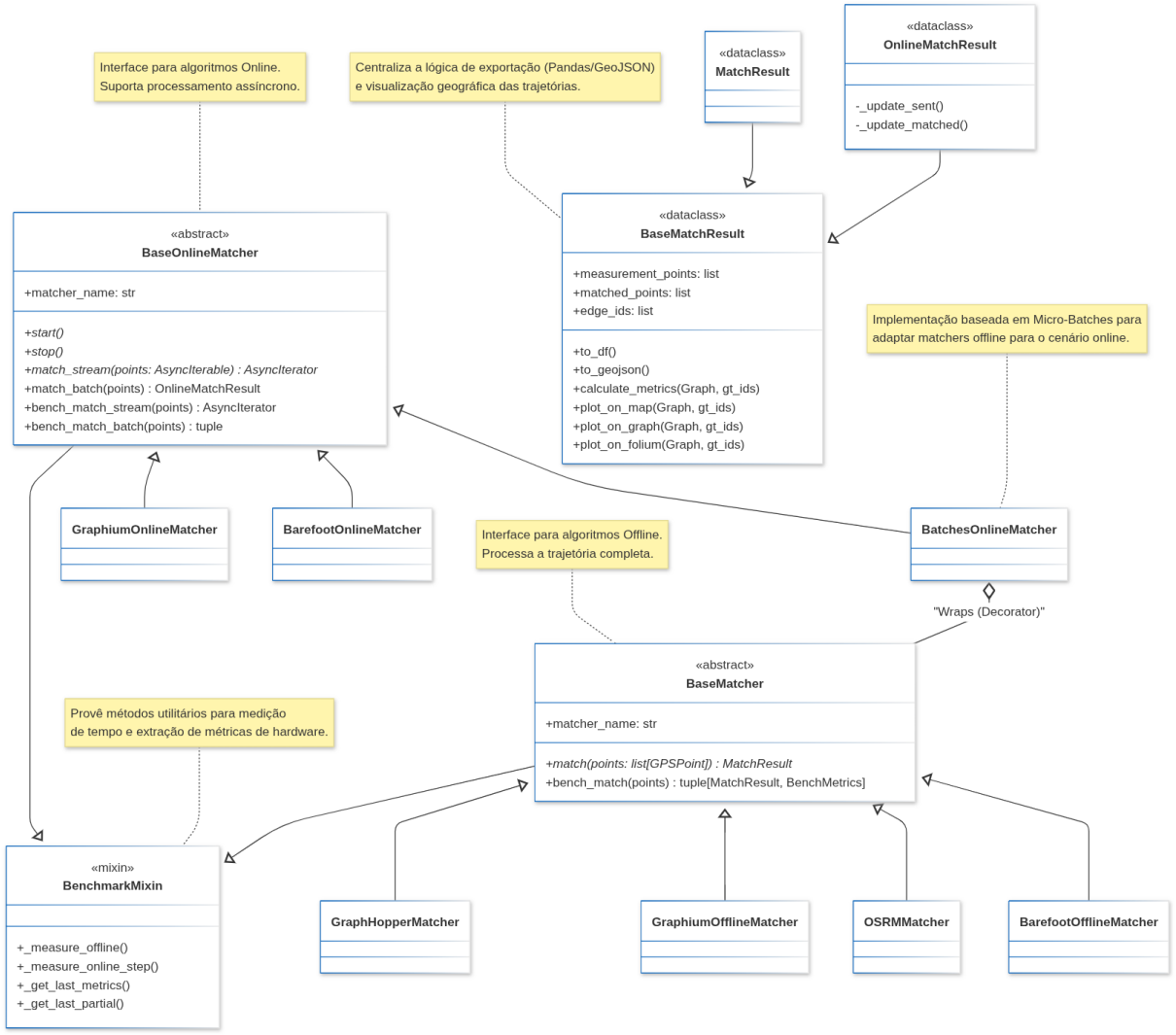
Fonte: Elaborado pelo autor.

Figura 15 – Fluxo de execução do *benchmark* e ciclo de vida dos serviços.



Fonte: Elaborado pelo autor.

Figura 16 – Hierarquia de interfaces e adaptadores na `mmlib`.



Fonte: Elaborado pelo autor.

Essa padronização por meio de uma interface comum foi essencial para assegurar uma avaliação justa, pois permitiu que o mesmo *workload*, a mesma matriz de configuração e o mesmo protocolo de submissão fossem aplicados a todos os SUTs de forma isonômica, eliminando vieses decorrentes de diferenças na forma de integração.

Além da padronização da interação, a `mmlib` incorporou, como parte de sua arquitetura, a instrumentação de métricas *black-box* de ponta a ponta no lado do cliente. Essa instrumentação foi implementada de forma transversal por meio do princípio de inversão de dependência, no qual o módulo coletor de métricas depende exclusivamente das interfaces comuns dos *matchers*. Dessa forma, todas as ferramentas foram medidas sob o mesmo mecanismo de coleta, garantindo consistência e comparabilidade entre os resultados. As métricas coletadas registram o tempo observado pela aplicação consumidora, incluindo custos de serialização, comunicação e espera pelo resultado do serviço.

4.3.2 Orquestrador de benchmark e automação de experimentos

Com a `mmlib`, a execução sistemática do *benchmark* foi automatizada por um orquestrador implementado no *script* principal de execução e em módulos auxiliares de infraestrutura. O orquestrador constituiu o elemento central do *benchmark harness*, sendo responsável por materializar, de forma prática, a execução da suíte definida sobre os diferentes SUTs, assegurando reprodutibilidade, rastreabilidade e mínima intervenção manual.

Em termos funcionais, o orquestrador teve três objetivos principais. O primeiro foi operacionalizar a orquestração do *benchmark harness*, coordenando o ciclo de vida completo dos experimentos. O segundo foi carregar o *workload* e executar sistematicamente a suíte de *benchmark* sobre cada SUT, respeitando todas as combinações definidas na matriz de configuração experimental. O terceiro foi garantir a coleta integrada e sincronizada de métricas *client-side* e *server-side*, permitindo uma análise conjunta de acurácia, desempenho e uso de recursos computacionais.

Para atender a esse último objetivo, a execução dos experimentos foi acompanhada por uma pilha de observabilidade baseada em Prometheus[16] e cAdvisor[17]. O Prometheus foi utilizado como sistema de monitoramento e consulta de métricas em formato de séries temporais, enquanto o cAdvisor atuou como agente de instrumentação dos contêineres, expondo métricas

[16] https://prometheus.io/
[17] https://github.com/google/cadvisor

de infraestrutura, como uso de CPU, memória e rede. Essa pilha foi inicializada no início da bateria experimental e permaneceu ativa durante toda a execução dos experimentos.

Nesse cenário, cada experimento consistiu na execução de uma instância da suíte de *benchmark* sobre um SUT específico. Antes de iniciar cada execução, o orquestrador recriava o contêiner do serviço de *map matching*, atribuindo a ele identificadores e *labels* particulares daquele experimento. Em seguida, um emissor de dados GNSS era configurado para simular o trajeto de um veículo, publicando pontos de acordo com a taxa de amostragem e o fator de aceleração temporal estabelecidos na matriz de configuração. Durante essa etapa, a biblioteca `mmlib` ficava encarregada de enviar os dados ao SUT e coletar métricas de ponta a ponta no lado do cliente, descrevendo a latência e a vazão percebidas pela aplicação consumidora.

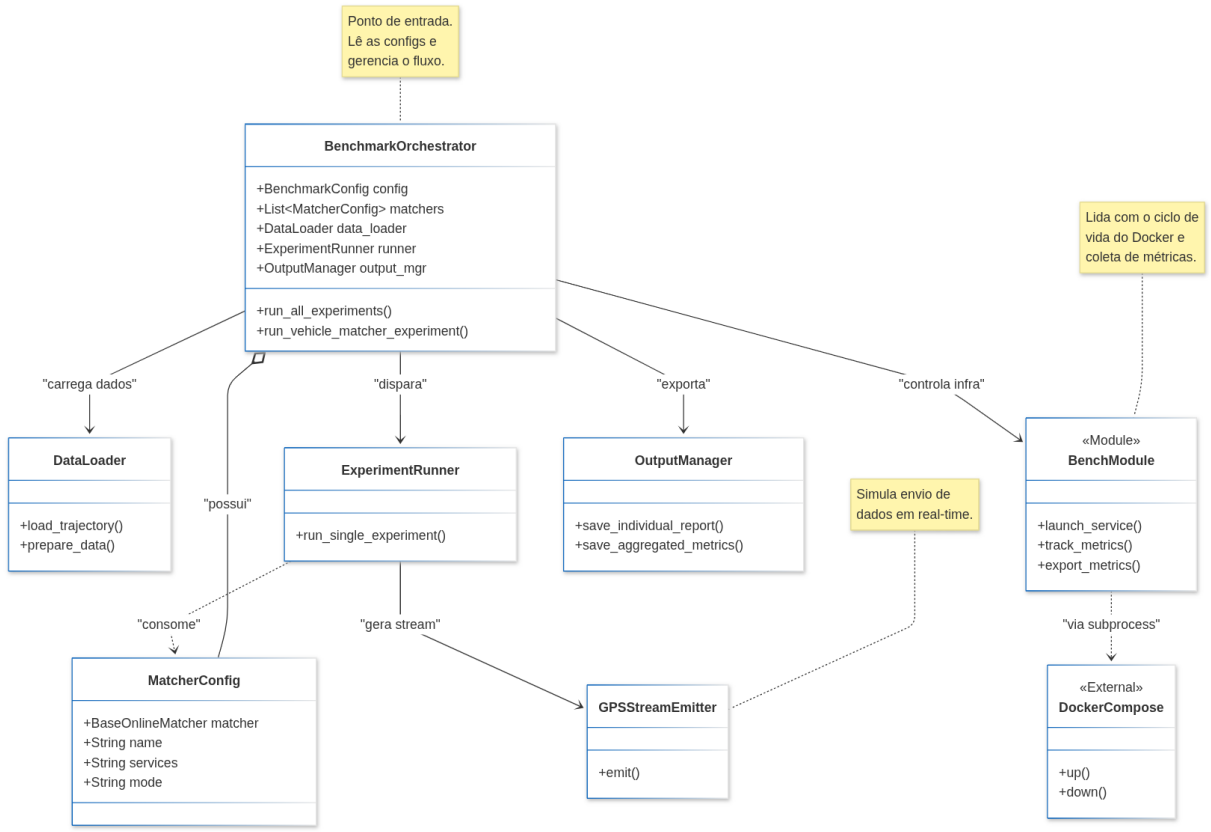
Ao término de cada experimento, o orquestrador consulta o Prometheus para coletar métricas de infraestrutura no lado do servidor, filtrando as séries temporais pelos identificadores associados à execução. Em seguida, o serviço de *map matching* é encerrado, garantindo isolamento lógico entre experimentos consecutivos. Esse procedimento é repetido sistematicamente para todos os SUTs e para todas as combinações da matriz de configuração experimental.

Do ponto de vista estrutural, o orquestrador foi implementado de forma modular, com responsabilidades bem delimitadas entre seus componentes internos. Essa organização teve como objetivo facilitar a extensibilidade, a rastreabilidade dos experimentos e a separação clara entre a lógica de controle, a execução e a persistência de resultados. A Figura 17 apresenta a arquitetura interna do orquestrador, destacando seus principais componentes e suas interações.

O `BenchmarkOrchestrator` atua como ponto de entrada do sistema, sendo responsável por carregar as configurações experimentais, iterar sobre os SUTs e coordenar o fluxo global de execução. O carregamento e a preparação dos dados são delegados ao componente `DataLoader`, enquanto a execução lógica de cada experimento é conduzida pelo `ExperimentRunner`, que utiliza o `GPSStreamEmitter` para simular o fluxo temporal de dados GNSS. O gerenciamento da infraestrutura de execução, incluindo o ciclo de vida dos contêineres e a inicialização dos coletores de métricas, é abstraído pelo módulo `BenchModule`, que interage com o Docker por meio de comandos automatizados. Por fim, a persistência dos resultados individuais e agregados é centralizada no `OutputManager`, assegurando organização e rastreabilidade dos artefatos gerados.

O fluxo de coleta, integração e agregação de métricas adotado neste trabalho é ilustrado na Figura 18 e pode ser descrito de forma sequencial conforme segue:

Figura 17 – Componentes principais do *benchmark harness* no lado do cliente.



Fonte: Elaborado pelo autor.

I. **Execução do experimento**: o SUT é iniciado em um contêiner dedicado e processa a carga de trabalho definida, recebendo pontos GNSS simulados pelo emissor configurado pelo orquestrador.
II. **Coleta de métricas ponta a ponta**: durante a execução, a biblioteca `mmlib` coleta métricas *black-box* no lado do cliente, incluindo latência e vazão, bem como os resultados do *map matching* necessários para o cálculo das métricas de acurácia.
III. **Instrumentação da infraestrutura**: em paralelo, o cAdvisor monitora continuamente o uso de recursos do contêiner do SUT, expondo métricas de CPU, memória e rede.
IV. **Coleta de métricas no lado do servidor**: o Prometheus coleta periodicamente as métricas de infraestrutura, que ao final de cada experimento são consultadas pelo *harness* por meio de consultas filtradas pelo identificador do experimento.
V. **Agregação e correlação**: as métricas de ponta a ponta, as métricas de acurácia e as métricas de infraestrutura são agregadas em um único processo, sendo correlacionadas por meio do identificador do experimento.
VI. **Persistência dos resultados**: os dados agregados são persistidos em relatórios conso-

lidados, que constituem a base para a análise quantitativa apresentada no capítulo de resultados.

Ao término da campanha experimental, o orquestrador encerrou a pilha de observabilidade e consolidou todas as métricas em relatórios finais. Essa automação foi fundamental para reduzir a variabilidade operacional, assegurar a rastreabilidade e garantir que cada resultado estivesse diretamente vinculado a uma configuração experimental explícita e a um identificador único.

4.4 Adaptação das ferramentas *offline* para execução *online*

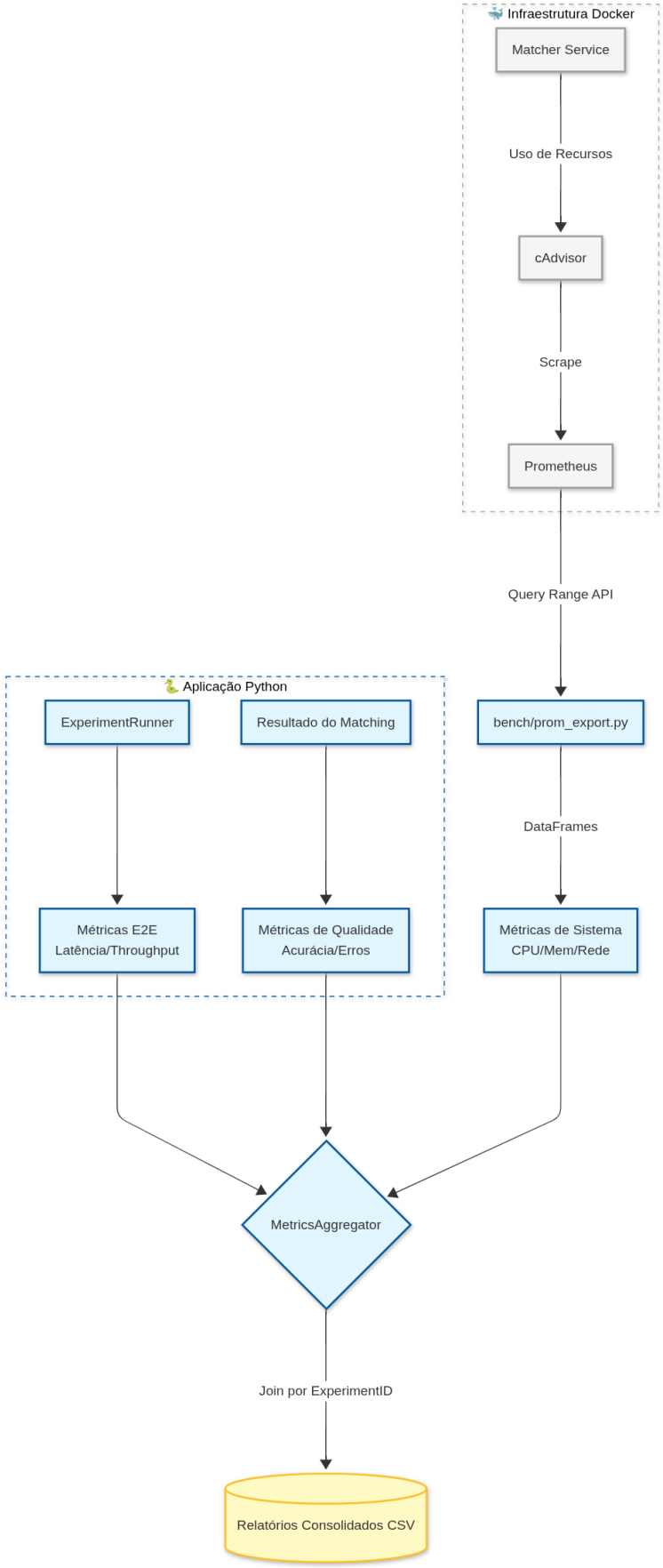
Uma parcela relevante das ferramentas OSS selecionadas opera nativamente no paradigma *offline*, isto é, assume a disponibilidade de uma trajetória completa como entrada e produz um resultado apenas ao final do processamento. Entretanto, o objetivo deste trabalho é avaliar o comportamento dessas soluções em cenários *online*, nos quais as observações GNSS chegam de forma incremental. Para compatibilizar esses dois modelos de execução sem alterar a lógica algorítmica interna de cada SUT, foi implementada uma camada de adaptação no lado do cliente, baseada em lotes, viabilizando uma simulação controlada do processamento de forma incremental.

A estratégia adotada consiste em agrupar o fluxo de pontos recebidos em lotes (*batches*) de tamanho fixo (*batch size*, bs). Sempre que um lote é completado, o processamento é acionado por meio do *matcher* originalmente *offline*. Essa abordagem preserva integralmente o comportamento do algoritmo dentro de cada lote, ao mesmo tempo em que permite ao *benchmark harness* observar resultados intermediários e coletar métricas de latência e vazão em um regime incremental.

Do ponto de vista de implementação, essa adaptação foi encapsulada no componente `BatchesOnlineMatcher`. Esse componente implementa a interface `OnlineBaseMatcher` e recebe, em seu construtor, uma instância que segue a interface *offline* `BaseMatcher`. Dessa forma, o `BatchesOnlineMatcher` atua como um adaptador entre os dois paradigmas, aplicando explicitamente o padrão *Adapter*. Essa solução torna ferramentas que expõem apenas interfaces síncronas e em lote compatíveis com a `mmlib`, permitindo sua execução no contexto *online* do *benchmark* sem modificações no algoritmo original.

A Figura 19 sintetiza o fluxo operacional da adaptação. Ao iniciar a execução, a aplicação cliente invoca `match_stream(points)` e passa a fornecer pontos individualmente

Figura 18 – Fluxo de coleta, integração e agregação de métricas no *benchmark harness*.



Fonte: Elaborado pelo autor.

ao adaptador. Cada ponto recebido é armazenado em um *buffer* local. Quando o número de observações acumuladas atinge *bs*, o adaptador dispara o processamento do lote de forma assíncrona por meio de um `ThreadPoolExecutor`. Essa decisão é necessária para evitar o bloqueio do laço assíncrono e para manter a capacidade de instrumentação de métricas *black-box* no lado do cliente, incluindo tempos de espera e custos de comunicação.

Uma vez concluído o processamento de um lote, o resultado *offline* é convertido em uma instância de `OnlineMatchResult` e incorporado a uma geometria acumulada mantida pelo adaptador. Essa estratégia de atualização incremental possibilita que a aplicação consumidora obtenha estimativas parciais ao longo do tempo, por meio de sucessivos *yields*, mesmo quando o algoritmo subjacente não opera em regime estritamente ponto a ponto. Em seguida, o *buffer* é reinicializado e o procedimento é reiterado para os pontos subsequentes do fluxo de dados.

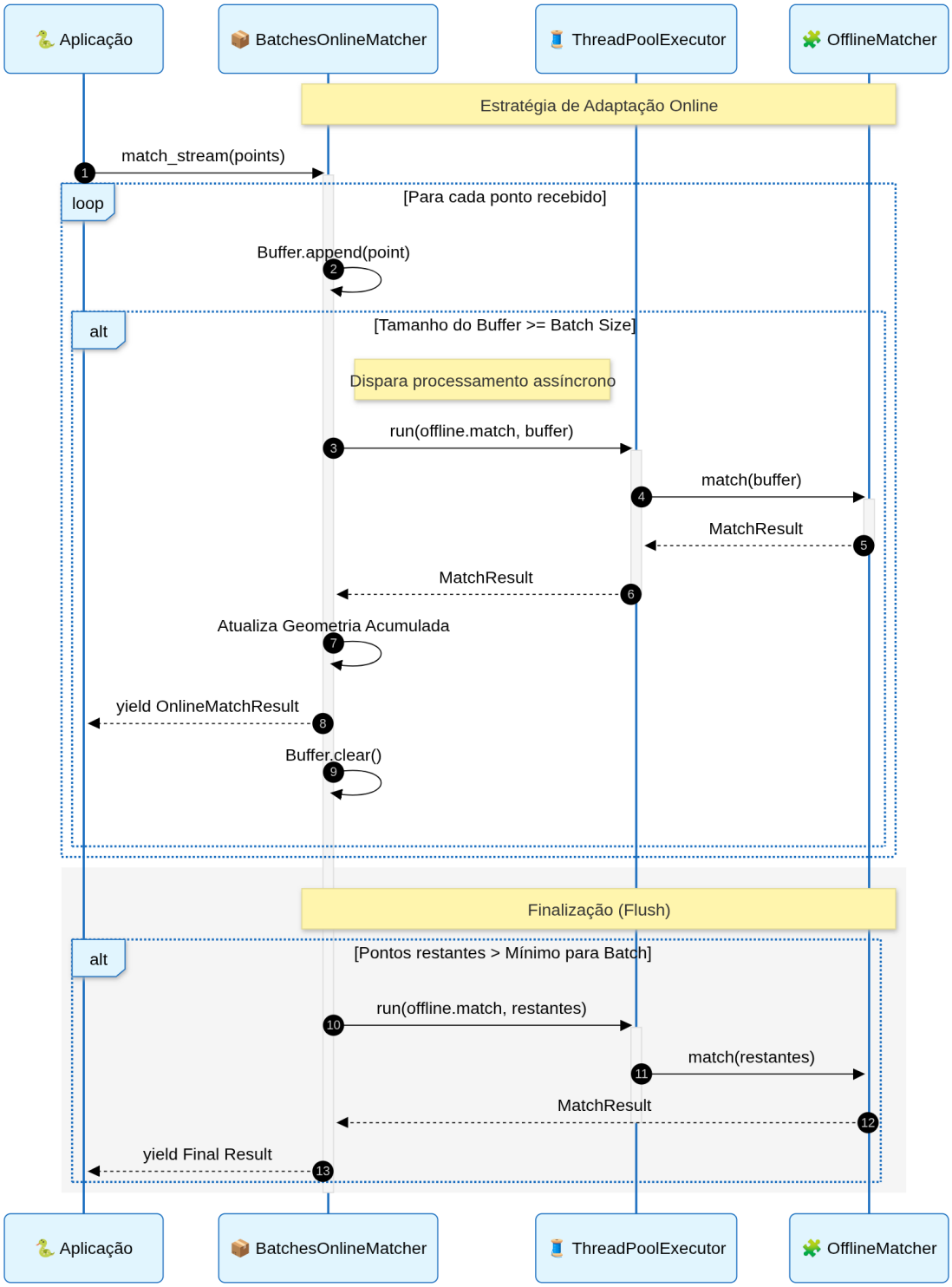
Ao final do fluxo de observações, o adaptador executa uma etapa de finalização (*flush*). Caso existam pontos residuais no *buffer* e a quantidade acumulada seja suficiente para produzir um resultado válido, o último micro-lote é processado e emitido como resultado final. Essa etapa evita o descarte de observações que não completariam um lote integral e reduz a discrepância entre a trajetória processada e a trajetória originalmente emitida pelo *publisher*.

Essa adaptação por lotes foi aplicada de forma isonômica a todas as ferramentas *offline* incluídas no portfólio, utilizando os mesmos valores de *bs* definidos na subseção 4.2.1. Dessa forma, as diferenças observadas entre os SUTs refletem predominantemente as características algorítmicas e arquiteturais das ferramentas avaliadas, e não particularidades do mecanismo de integração. Além disso, a decisão de manter o processamento *offline* em lotes controlados reforça a validade interna do *benchmark*, pois preserva o comportamento do algoritmo original e limita a intervenção do *harness* à organização do fluxo de entrada e à padronização da emissão de resultados.

4.5 Definição do Ambiente Experimental e Execução do *Benchmark*

A *benchmark suite* definida neste trabalho foi executada sistematicamente sobre todos os sistemas sob teste (SUTs) selecionados, já adequados. Cada trajetória definida no *workload* amostrado foi processada sob todas as combinações estabelecidas na matriz de configuração experimental, totalizando até 240 execuções por ferramenta nos cenários com processamento em lotes. Essa execução sistemática permitiu avaliar o comportamento das ferramentas sob diferentes regimes de processamento *online*.

Figura 19 – Fluxo de execução do adaptador `BatchesOnlineMatcher` para simular processamento *online* via lotes.



Fonte: Elaborado pelo autor.

Nesse cenário, todos os experimentos foram conduzidos em um ambiente computacional controlado, com o objetivo de garantir a reprodutibilidade dos resultados e a comparabilidade entre os SUTs. As execuções foram realizadas na seguinte infraestrutura:

- **Host**: Acer Predator PT316-51s;
- **Processador (CPU)**: Intel® Core™ i7-12700H (12ª geração, 14 núcleos, 20 *threads*, frequência máxima de 4.6 GHz);
- **Memória RAM**: 32 GB DDR5;
- **GPU**: NVIDIA GeForce RTX 3060 Mobile e Intel Iris Xe Graphics (não utilizadas nas execuções);
- **Armazenamento**: SSD NVMe M.2 Gen 4 de 512 GB;
- **Sistema Operacional**: Kubuntu 24.04.3 LTS (x86_64);
- **Kernel**: Linux 6.8.0-90-generic.

Além disso, ressalta-se que a alocação de recursos computacionais não foi artificialmente restringida durante a execução do *benchmark*, permitindo que os SUTs explorassem o potencial completo do ambiente disponível. Com o objetivo de minimizar interferências externas, as execuções foram conduzidas em um ambiente dedicado, no qual apenas os processos essenciais ao sistema operacional e ao *benchmark* permanecem ativos.

4.6 Análise e Avaliação dos Resultados

A análise e a avaliação dos resultados obtidos a partir da execução do *benchmark* foram conduzidas de forma sistemática, tomando como referência as questões de pesquisa definidas na seção 1.2. Essas questões estão alinhadas aos objetivos deste trabalho e aos critérios de qualidade de *benchmarking* discutidos na subseção 2.4.1, orientando tanto a seleção das métricas quanto a organização da análise experimental.

Em particular, a avaliação foi estruturada para responder às quatro dimensões investigadas pelas questões de pesquisa: a qualidade de correspondência das ferramentas em função da taxa de amostragem, a eficiência temporal no paradigma *online*, a eficiência computacional associada à qualidade dos resultados e a capacidade de processamento contínuo sob diferentes regimes de amostragem.

Essa abordagem permitiu estabelecer uma análise integrada e comparável entre as ferramentas avaliadas, evidenciando de forma objetiva os compromissos (*trade-offs*) entre acurácia, latência, uso de recursos computacionais e capacidade de processamento, fornecendo o

embasamento necessário para a resposta final às QPs deste trabalho.

4.7 Ameaças à validade

As principais ameaças à validade deste estudo são organizadas conforme a tipologia de estudos experimentais abordada na revisão de FELDT Robert; MAGAZINIUS (2010), abrangendo validade interna, externa, de construto e de conclusão.

No que se refere à **validade interna**, buscou-se reduzir vieses por meio da automação integral do ciclo de vida dos experimentos, do isolamento por contêiner e da coleta padronizada de métricas via `mmlib`. Apesar desses cuidados, ainda podem ocorrer variações residuais de carga do sistema operacional e efeitos de aquecimento (*warm-up*), que potencialmente influenciam métricas temporais e de recursos.

Quanto à **validade externa**, a principal limitação é que a carga de trabalho se baseia em trajetórias sintéticas de uma única região (O'Hare, Chicago), o que pode restringir a generalização dos resultados para outras malhas viárias. Em contrapartida, a seleção por quartis foi adotada para preservar a diversidade de comprimentos e de complexidade estrutural das trajetórias, ampliando o espectro de cenários considerados.

No caso da **validade de construto**, embora as métricas estejam definidas na fundamentação teórica, sua operacionalização depende da disponibilidade de identificadores de arestas do OSM e da normalização dos resultados entre ferramentas. Para mitigar essa ameaça, foram feitas apenas adaptações mínimas, descritas nos SUTs, preservando ao máximo o comportamento algorítmico original.

Por fim, quanto à **validade de conclusão**, a comparação entre as ferramentas baseia-se em múltiplas execuções com diferentes configurações de *sr* e *bs*, permitindo observar tendências consistentes em vários cenários experimentais. Ainda assim, as conclusões devem ser interpretadas à luz do ambiente computacional controlado descrito na seção 4.5, que define o contexto e os limites de validade dos resultados.

4.8 Documentação e Disponibilização

Em consonância com as boas práticas para pesquisa computacional reprodutível estabelecidas por Sandve *et al.* (2013), este trabalho adotou como princípio a documentação sistemática, o versionamento rigoroso e a disponibilização pública de todos os artefatos utilizados

na condução dos experimentos. Esse cuidado visa assegurar a reprodutibilidade, a verificabilidade e a extensibilidade dos resultados apresentados.

Todos os artefatos de *software*, configurações experimentais e conjuntos de dados empregados foram devidamente documentados e disponibilizados em repositórios públicos versionados. A Tabela 6 organiza esses artefatos em dois grupos principais: (i) os componentes desenvolvidos no escopo deste trabalho, que compõem o *benchmark harness* e sua infraestrutura de execução, e (ii) as ferramentas de *map matching* avaliadas experimentalmente, incluindo versões adaptadas quando necessário. Em todos os casos, são indicados os repositórios públicos correspondentes e os *commits* exatos utilizados durante a execução do *benchmark*.

Tabela 6 – Artefatos de software, repositórios e versões utilizadas

Artefato	**Repositório**	**Commit (v.)**
Artefatos desenvolvidos neste trabalho		
Benchmark harness	https://github.com/JoseEdSouza/map-matching	`45a3ebf`
Conjunto de dados sintético	https://github.com/JoseEdSouza/map-matching	`45a3ebf`
Biblioteca `mmlib`	https://github.com/JoseEdSouza/mmlib	`f57a132`
Ferramentas de map matching avaliadas		
Barefoot (adaptado)	https://github.com/JoseEdSouza/barefoot	`53eec69`
Graphium (importação)	https://github.com/JoseEdSouza/graphium	`54d1a86`
GraphiumMM (adaptado)	https://github.com/JoseEdSouza/graphium-neo4j	`b465f6a`
OSRM	https://github.com/Project-OSRM/osrm-backend	`c98537b`
GraphHopper	https://github.com/graphhopper/graphhopper	`88335ad`
GraphHopper (Docker)	https://github.com/IsraelHikingMap/graphhopper-docker	`c3d6b8d`

Fonte: Elaborada pelo autor.

As imagens Docker utilizadas nos experimentos foram construídas diretamente a partir dos *commits* apresentados na Tabela 6, assegurando correspondência exata entre o código-fonte versionado e os binários executados. O conjunto de dados sintético gerado, bem como os metadados necessários para a reprodução dos experimentos, foi disponibilizado juntamente com o repositório do *benchmark harness*[18,19].

Todo o código-fonte desenvolvido no escopo deste trabalho foi versionado utilizando o sistema de controle de versões Git[20] e disponibilizado publicamente por meio da plataforma GitHub. Esse material inclui o *benchmark harness*, os *scripts* de geração do conjunto de dados sintético, as rotinas de execução automatizada dos experimentos e os procedimentos de processamento e análise dos resultados. A biblioteca `mmlib`, embora mantida em repositório

[18] https://github.com/JoseEdSouza/map-matching/tree/45a3ebf/dataset
[19] https://github.com/JoseEdSouza/map-matching/blob/45a3ebf/run_benchmarking.py
[20] https://git-scm.com/

separado por razões de modularidade e reutilização, integra o mesmo ecossistema experimental e segue o mesmo processo de versionamento e documentação.

Por fim, todas as dependências do projeto foram explicitamente declaradas, gerenciadas e versionadas por meio do gerenciador de ambientes UV[21], garantindo consistência no ambiente de execução e reduzindo variações decorrentes de diferenças de configuração entre sistemas.

[21] https://docs.astral.sh/uv/

5 CONSTRUÇÃO DO CONJUNTO DE DADOS SINTÉTICO COM GROUND TRUTH PARA AVALIAÇÃO DE *MAP MATCHING*

Este capítulo descreve o processo de construção de um conjunto de dados sintético de trajetórias com *ground truth* exato, utilizado como carga de trabalho no *benchmark* de *map matching* apresentado neste trabalho. A criação desse conjunto de dados foi motivada pela escassez de bases públicas que fornecem trajetórias reais com correspondência exata conhecida à malha viária, condição essencial para a avaliação quantitativa da acurácia.

A metodologia adotada é inspirada nos trabalhos de Erdmann e Ebendt (2014) e Guastella *et al.* (2025), que demonstram a viabilidade do uso de simuladores de mobilidade urbana para a geração de trajetórias sintéticas controladas. Ao empregar um simulador, é possível manter conhecimento completo da trajetória real percorrida sobre a rede viária, ao mesmo tempo em que se emulam imperfeições típicas de sensores GNSS por meio de ruído artificial. A Figura 20 ilustra esse processo.

O conjunto de dados descrito neste capítulo é um artefato metodológico que sustenta a execução do *benchmark*, permitindo a comparação justa e reprodutível das ferramentas avaliadas. As seções a seguir detalham o processo de obtenção da malha viária (seção 5.1), a geração das trajetórias (seção 5.2), o pós-processamento dos dados (seção 5.3), a validação preliminar de sua adequação ao objetivo proposto (seção 5.4) e, finalmente, a síntese do capítulo (seção 5.5).

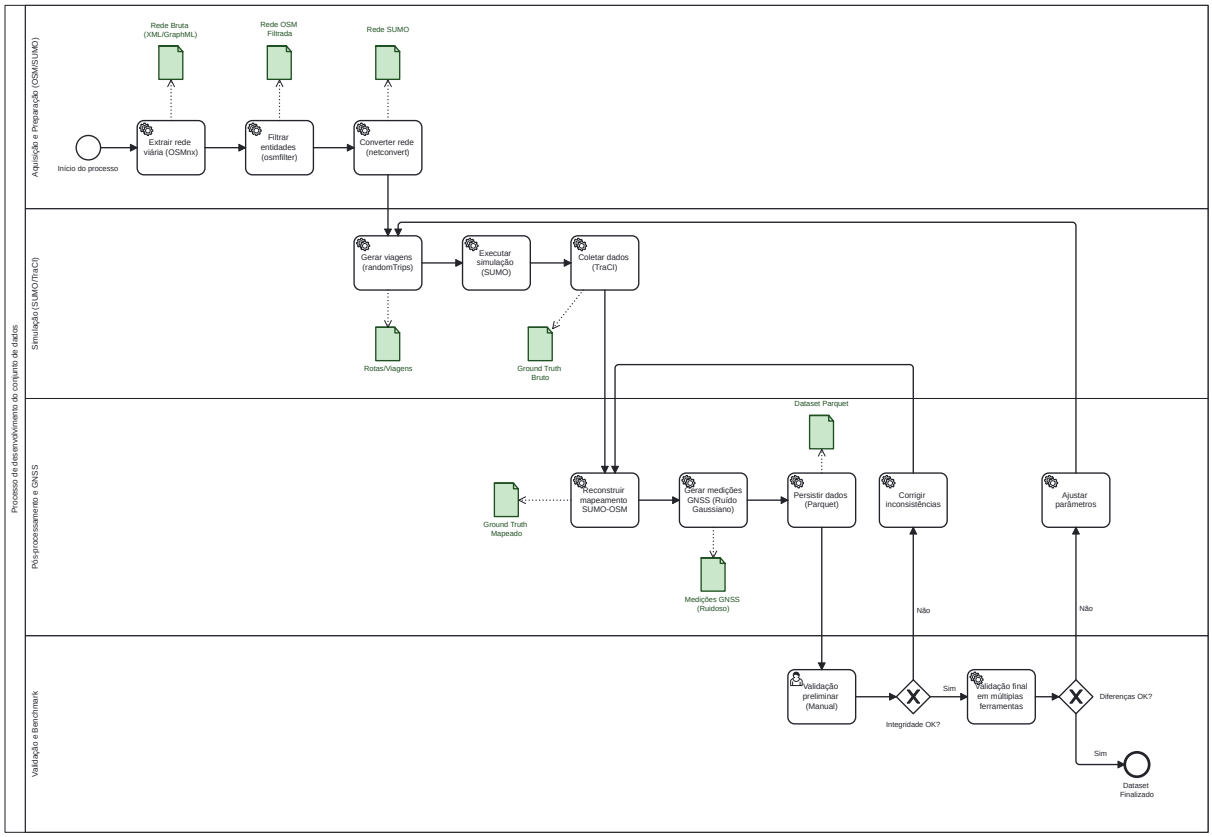
5.1 Obtenção e preparação da malha viária

A seleção da malha viária constituiu o primeiro passo na construção do conjunto de dados. A escolha da localidade foi guiada pela necessidade de criar cenários representativos e desafiadores para algoritmos de *map matching*. Conforme discutido por Chao *et al.* (2020), uma rede adequada deve conter tanto regiões densas e topologicamente complexas quanto trechos longos e contínuos, que tendem a gerar medições GNSS mais espaçadas.

Com base nesses critérios, foi selecionada a área comunitária 76 de Chicago (O'Hare), EUA. A extração da rede viária foi realizada com a biblioteca **OSMnx**[1], utilizando a consulta "O'Hare, Chicago, Illinois, USA" e o filtro do tipo *drive*, de modo a incluir apenas vias adequadas ao tráfego veicular. A malha resultante foi armazenada nos formatos XML e GraphML para uso nas etapas subsequentes.

[1] https://github.com/gboeing/osmnx

Figura 20 – Processo de desenvolvimento do conjunto de dados



Fonte: Elaborado pelo autor.

Após a extração, a rede foi filtrada com a ferramenta **osmfilter**[2], selecionando apenas entidades do tipo *highway*. Em seguida, a conversão para o formato compatível com o SUMO foi realizada por meio da ferramenta **Netconvert**[3], com parâmetros ajustados para preservar a fidelidade topológica da rede original. Esses ajustes visaram evitar a criação excessiva de junções automáticas e garantir a rastreabilidade entre os identificadores internos do simulador e os identificadores das arestas do OSM.

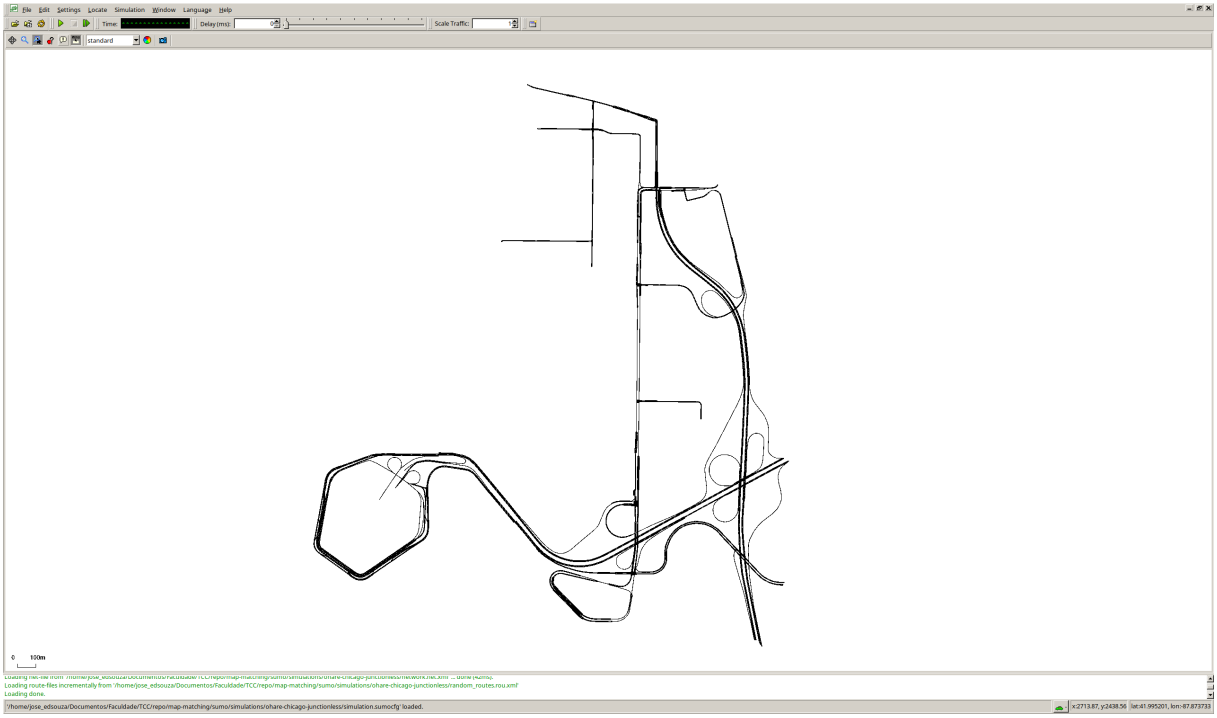
A Figura 21 apresenta a malha viária após o processo de conversão.

5.2 Geração de trajetórias sintéticas

Com a malha viária preparada, o tráfego sintético foi gerado utilizando a ferramenta **randomTrips.py**[4] do SUMO. As viagens foram geradas com a semente aleatória 42, duração total de uma hora e inserção de veículos a cada dois segundos, seguindo uma distribuição binomial que introduz variação não uniforme no fluxo. Foi definida uma distância mínima de

[2] https://wiki.openstreetmap.org/wiki/Osmfilter
[3] https://sumo.dlr.de/docs/netconvert.html
[4] https://sumo.dlr.de/docs/Tools/Trip.html

Figura 21 – Rede de ruas da região de O'Hare carregada no SUMO



Fonte: Elaborado pelo autor.

500 metros por viagem.

As rotas foram calculadas com base no menor caminho entre a origem e o destino, resultando em um total de 1759 trajetórias distintas. A simulação foi executada com um passo temporal de 0,5 segundos, permitindo alta resolução temporal na coleta das observações.

Durante a execução, os dados foram coletados dinamicamente por meio da interface **TraCI**[5]. Para cada veículo e a cada passo de simulação, foram registrados o identificador do veículo, o tempo, a posição exata e a via em que se encontrava, constituindo a base do *ground truth*.

5.3 Pós-processamento e geração das medições GNSS

Os dados brutos coletados passaram por um processo de pós-processamento com dois objetivos principais. O primeiro foi mapear as vias internas geradas pelo SUMO para as arestas correspondentes do grafo do OSM, garantindo consistência topológica. O segundo foi gerar medições GNSS ruidosas a partir do *ground truth*.

A reconstrução das correspondências entre vias internas e arestas externas foi realizada por meio de um procedimento recursivo que identifica os nós de entrada e saída das junções

[5] https://sumo.dlr.de/docs/TraCI.html

automáticas e os associa às arestas correspondentes no grafo original. Lacunas topológicas foram corrigidas por meio do cálculo de caminhos mínimos entre segmentos desconectados.

Em seguida, foi adicionado ruído gaussiano bidimensional às posições de cada ponto, com média zero, e desvio padrão de cinco metros ($\mu = 0m, \sigma = 5m$), conforme abordagens amplamente adotadas na literatura (Erdmann; Ebendt, 2014). Os conjuntos de dados finais, correspondentes ao *ground truth* e às medições GNSS ruidosas, foram armazenados em formato Parquet.

5.4 Validação e adequação ao *benchmark*

A validação do conjunto de dados foi conduzida em duas etapas complementares em diferentes momentos da pesquisa. A primeira validação foi preliminar à adequação funcional de todas as ferramentas e focou em verificar a integridade topológica e a viabilidade prática do conjunto de dados em experimentos de *map matching*. Já a segunda etapa, por sua vez, foi executada em um contexto pós-adequação funcional [6], essa validação focou em confirmar a adequação do conjunto como carga de trabalho do *benchmarking*, buscando avaliar sua capacidade de expor comportamentos distintos entre as ferramentas sob as mesmas condições experimentais.

Na primeira validação, foi conduzida uma validação qualitativa por inspeção manual. Uma amostra aleatória de dez trajetórias foi processada por um algoritmo de *map matching* de referência, o **GraphHopper**[7], e os resultados foram comparados visualmente com o *ground truth*. Essa verificação buscou identificar lacunas, descontinuidades e inconsistências evidentes entre as rotas inferidas e as rotas reais da simulação.

Tabela 7 – Análise qualitativa dos resultados do *map matching* em uma amostra de trajetórias.

Classificação	**Quantidade**	**Descrição**
Correspondência Perfeita	3	A trajetória recuperada foi idêntica ao *ground truth*.
Desvios Mínimos	3	A rota principal foi corretamente identificada, com pequenos desvios pontuais.
Desvios Significativos	4	A rota inferida apresentou desvios desnecessários ou escolhas subótimas.

Fonte: Elaborada pelo autor.

Os resultados apresentados na Tabela 7 indicaram que o conjunto de dados contém

[6] neste momento da pesquisa, todas as ferramentas já haviam sido adequadas e tinham um cliente escrito na `mmlib`

[7] https://github.com/graphhopper/graphhopper

trajetórias de diferentes níveis de dificuldade, incluindo casos bem condicionados e casos mais desafiadores, frequentemente associados a junções complexas e múltiplos níveis de vias.

Na segunda validação, observou-se que parte das trajetórias selecionadas resulta em correspondências consistentes e equivalentes entre os SUTs, caracterizando cenários simples. Em contraste, houve trajetórias nas quais algumas ferramentas apresentaram desvios desnecessários ou quebras de correspondência, enquanto outras mantiveram correspondências corretas e completas. Essa heterogeneidade é desejável no contexto de *benchmarking*, pois indica que a carga de trabalho inclui cenários multivariados capazes de revelar diferenças de robustez, acurácia e comportamento incremental entre as soluções avaliadas.

Dessa forma, a validação em duas etapas confere confiança de que o conjunto de dados é topologicamente consistente, viável para experimentação e adequado para sustentar uma avaliação comparativa das ferramentas no contexto do *benchmark* proposto.

5.5 Síntese

Este capítulo apresentou o processo de construção de um conjunto de dados sintético com *ground truth* exato, baseado em simulação de mobilidade urbana. O conjunto resultante oferece diversidade estrutural e níveis variados de dificuldade, atendendo aos requisitos necessários para a avaliação sistemática de ferramentas de *map matching*. A adequação do conjunto foi verificada em duas etapas, incluindo uma validação qualitativa preliminar e uma verificação posterior no contexto do *benchmark*, o que reforça sua utilidade como carga de trabalho para a comparação experimental apresentada nos capítulos subsequentes.

6 RESULTADOS

Este capítulo apresenta e analisa os resultados obtidos a partir da execução do *benchmarking* de *map matching*, seguindo a metodologia descrita no Capítulo 4. A análise é organizada de forma gradual, com o propósito de caracterizar o comportamento das ferramentas avaliadas sob diferentes taxas de amostragem e de responder às questões de pesquisa estabelecidas neste trabalho. Primeiramente, a seção 6.1 oferece uma visão geral dos experimentos realizados, detalhando o volume de execuções, as ferramentas consideradas e as configurações adotadas. Na sequência, a seção 6.2 examina a qualidade da correspondência produzida pelas ferramentas em função da taxa de amostragem, com base nas métricas de acurácia, Taxa de erro e $NKError$. A seção 6.3 analisa a eficiência temporal no paradigma *online*, por meio da avaliação da latência média de processamento de ponta a ponta. Já na seção 6.4, investiga-se o custo computacional das ferramentas, considerando o uso médio de CPU e de memória, bem como sua relação com a qualidade dos resultados alcançados. Por fim, a seção 6.5 examina a capacidade de processamento contínuo das ferramentas por meio da análise de vazão. O capítulo é encerrado com as seções 6.6 e 6.7, que, respectivamente, integram os principais achados experimentais e fornecem respostas diretas às questões de pesquisa definidas na seção 1.2, ressaltando implicações práticas para a escolha de ferramentas em aplicações reais.

6.1 Visão Geral dos Experimentos Executados

A suíte de *benchmarking* definida na seção 4.2 foi executada integralmente sobre o conjunto das 4 ferramentas selecionadas. Ela compreende 20 trajetórias amostradas de forma estratificada, avaliadas em 4 taxas de amostragem. Para sistemas com processamento incremental em lotes, foram considerados 3 tamanhos de lote adicionais, enquanto ferramentas estritamente ponto a ponto foram avaliadas apenas quanto à taxa de amostragem.

A combinação desses fatores resultou em 800 execuções planejadas, distribuídas entre os 4 SUTs conforme suas capacidades operacionais. Essas execuções foram conduzidas de forma automatizada pelo *benchmark harness*, respeitando as configurações e a carga de trabalho definidas na metodologia.

Durante a execução do *benchmarking*, observou-se que uma pequena fração das instâncias planejadas não foi concluída com sucesso devido a falhas internas em algumas das ferramentas avaliadas. Para mitigar efeitos efêmeros e falhas ocasionais de inicialização, o

orquestrador de experimentos adotou uma política de até três tentativas automáticas para cada execução. Em cada tentativa, o contêiner da ferramenta era integralmente recriado e o cenário experimental reinicializado, assegurando condições limpas e controladas.

Quando uma execução persistia em falhar após essas tentativas, ela era classificada como não concluída e excluída da análise subsequente. Considerando que todas as ferramentas foram submetidas ao mesmo processo de orquestração, à mesma suíte de testes e às mesmas condições experimentais, tais falhas foram interpretadas como parte do comportamento empírico observado das próprias ferramentas no contexto do *benchmark*.

Dessa forma, os resultados apresentados neste capítulo baseiam-se exclusivamente nas execuções concluídas com sucesso. Do total de 800 execuções inicialmente planejadas, 790 foram consideradas válidas para análise, correspondendo a uma taxa de sucesso global de 98,75%. A distribuição detalhada das execuções planejadas, concluídas e descartadas por ferramenta é apresentada na Tabela 8.

Tabela 8 – Execuções planejadas, concluídas e descartadas por ferramenta

Ferramenta	**Planejadas**	**Concluídas**	**Descartadas**	**Sucesso (%)**
GraphHopper	240	240	0	100,0%
GraphiumMM	240	239	1	99,6%
OSRM	240	231	9	96,2%
Barefoot	80	80	0	100,0%
Total	**800**	**790**	**10**	**98,7%**

Fonte: Elaborada pelo autor.

6.1.1 Considerações sobre inicialização e variantes de execução

Durante a análise experimental dos resultados, observou-se que determinadas ferramentas apresentam comportamento fortemente influenciado pelo contexto inicial de observações disponíveis no início do processamento. Esse efeito é particularmente relevante em algoritmos de *map matching online* baseados em janelas ou em processamento incremental por lotes, nos quais a qualidade da correspondência pode depender da existência de um segmento inicial suficientemente informativo.

No trabalho de Rehrl *et al.* (2018), os autores destacam que o algoritmo empregado pelo GraphiumMM depende de um contexto inicial maior para estabelecer corretamente a correspondência entre pontos GNSS e segmentos da malha viária. Em termos práticos, o algoritmo tenta identificar um segmento compatível para cada ponto e, na ausência de uma

correspondência plausível, posterga a decisão até que informações adicionais estejam disponíveis.

Embora o GraphiumMM seja uma ferramenta nativamente orientada ao paradigma *online* do ponto de vista algorítmico, sua implementação prática adota uma estratégia incremental baseada em lotes de pontos. Especificamente, a lógica de acumulação dos pontos, bem como o controle do tamanho da janela ou do lote, é realizada no lado do cliente da ferramenta. A cada requisição HTTP, o cliente envia um conjunto de pontos juntamente com o identificador da última aresta estimada, permitindo que o algoritmo execute o processamento incremental internamente.

No contexto deste *benchmark*, o GraphiumMM é executado por meio de um cliente implementado como *matcher* na biblioteca `mmlib`, responsável por gerenciar explicitamente o *buffer* de pontos e a lógica de envio em lotes. Essa característica torna o comportamento experimental do GraphiumMM comparável ao das ferramentas originalmente *offline* que foram adaptadas para execução incremental por meio de processamento em lotes. Por essa razão, o GraphiumMM é avaliado neste trabalho também em função do tamanho do lote (*batch size*), de forma consistente com as demais soluções avaliadas sob esse regime.

Durante testes experimentais preliminares, foi observado que a execução inicial de um lote maior, contendo 30 pontos, resulta em maior estabilidade e melhor acurácia na identificação do segmento inicial, independentemente do tamanho de lote adotado posteriormente. Esse comportamento caracteriza um fenômeno de inicialização a frio (*cold start*), no qual o algoritmo necessita de um contexto inicial mais amplo para convergir para uma solução consistente.

Em razão disso, a configuração padrão do cliente do GraphiumMM adotada neste trabalho inclui explicitamente essa fase inicial de *cold start*. Como consequência, as métricas agregadas de latência da ferramenta incorporam o custo adicional dessa inicialização, refletindo o comportamento esperado em uma aplicação real que inicia o processamento sem conhecimento prévio do estado da trajetória.

Com o objetivo de isolar e analisar separadamente o impacto dessa fase inicial sobre as métricas temporais, foi considerada uma variante adicional de execução, denominada *GraphiumMM (warm)*. Nessa variante, os custos associados ao *cold start* são desconsiderados na agregação das métricas, permitindo avaliar o desempenho do algoritmo em regime estacionário, após a estabilização do contexto inicial. Ambas as variantes são analisadas nas seções subsequentes, oferecendo uma visão mais precisa do impacto da inicialização sobre o desempenho e a

eficiência temporal da ferramenta.

6.2 Qualidade de Correspondência em Função da Taxa de Amostragem

Para responder à QP1, os resultados de cada execução válida do *benchmark* foram processados para calcular as métricas de qualidade definidas na suíte de *benchmark* (subseção 4.2.3) e apresentadas no referencial teórico (subseção 2.1.6), a partir da comparação entre as sequências de arestas inferidas e o *ground truth*. Em seguida, para cada ferramenta e taxa de amostragem (*sample rate*), os valores de F_1-score, taxa de erro e $NKError$ foram agregados pela média, considerando todas as execuções com a mesma configuração experimental.

A Tabela 9 apresenta os valores médios dessas métricas por ferramenta e por taxa de amostragem, fornecendo uma visão quantitativa da qualidade de correspondência. Para facilitar a comparação e evidenciar tendências, esses resultados também são mostrados graficamente: a Figura 22 ilustra a variação do F_1-score médio em função da taxa de amostragem, e a Figura 23 mostra o comportamento do erro médio de Newson e Krumm nas mesmas condições. Em conjunto, a tabela e as figuras permitem observar como a redução da densidade de pontos impacta de forma distinta as ferramentas avaliadas, revelando diferentes níveis de robustez à esparsidade das trajetórias.

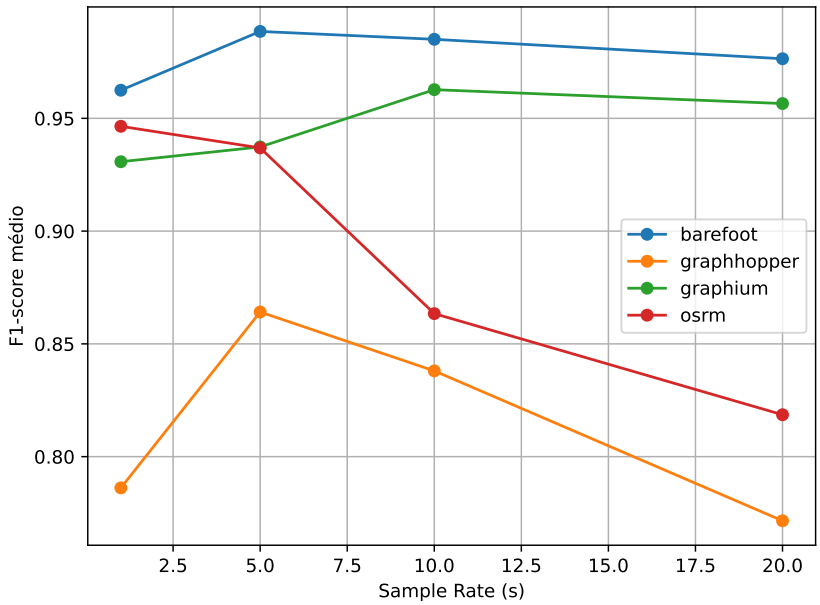
Tabela 9 – Qualidade de correspondência das ferramentas ao variar a taxa de amostragem

Ferramenta	Taxa (s)	F1-score médio	ErrorRate médio	Erro NK médio
Barefoot	1	0.962	0.038	0.078
	5	**0.989**	**0.011**	**0.023**
	10	0.985	0.015	0.029
	20	0.976	0.024	0.045
GraphiumMM	1	0.931	0.069	0.139
	5	0.937	0.063	0.121
	10	0.963	0.037	0.069
	20	0.957	0.043	0.080
OSRM	1	0.946	0.054	0.114
	5	0.937	0.063	0.120
	10	0.863	0.137	0.233
	20	0.819	0.181	0.294
GraphHopper	1	0.786	0.214	0.489
	5	0.864	0.136	0.284
	10	0.838	0.162	0.314
	20	0.772	0.228	0.434

Fonte: Elaborada pelo autor.

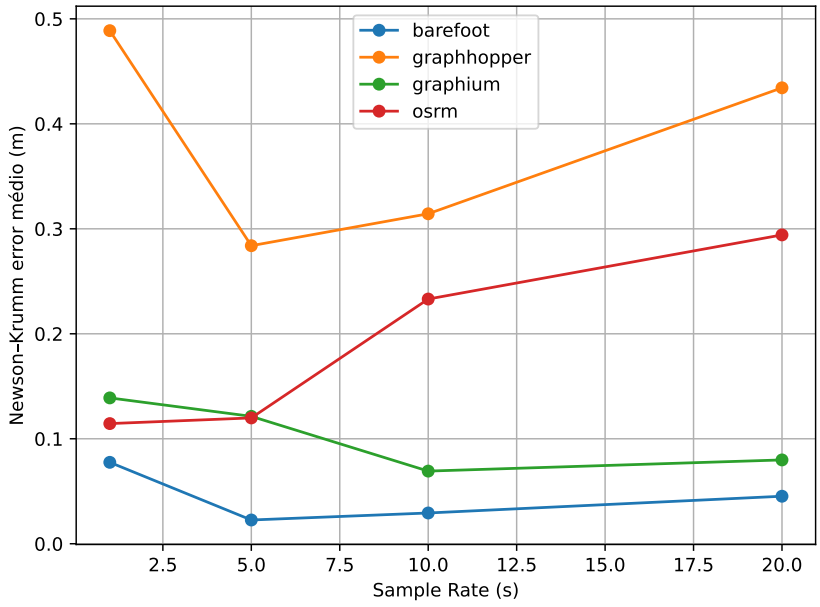
Nesse contexto, o Barefoot demonstrou a maior resiliência entre os avaliados. Seu

Figura 22 – F1-score médio por ferramenta variando a taxa de amostragem



Fonte: Elaborado pelo autor.

Figura 23 – Erro médio de correspondência por ferramenta variando a taxa de amostragem



Fonte: Elaborado pelo autor.

F1-score médio manteve-se consistentemente acima de $0,96$, mesmo sob condições de alta esparsidade. Essa estabilidade é ratificada pelos baixos índices do $NKError$ e pela taxa de erro, o que aponta para uma elevada precisão na reconstrução geométrica da trajetória, independentemente do intervalo temporal entre as observações GNSS.

Já o GraphiumMM apresentou um equilíbrio competitivo, com F1-score oscilando entre $0,93$ e $0,96$. Notavelmente, em taxas intermediárias, seus desvios espaciais aproximam-se dos resultados obtidos pelo Barefoot. Contudo, o algoritmo demonstra uma leve queda de desempenho conforme as lacunas entre pontos aumentam, embora ainda supere as ferramentas restantes em cenários críticos de baixa frequência.

Em contrapartida, GraphHopper e OSRM mostraram-se mais sensíveis à perda de resolução dos dados, sendo o GraphHopper a ferramenta que registrou os maiores valores de $NKError$ em todos os cenários, indicando um distanciamento recorrente entre o percurso inferido e o *ground truth*. Já o OSRM, embora eficaz em altas frequências (1s e 5s), sofre uma degradação acentuada à medida que os dados se tornam menos densos, resultando em uma queda drástica na acurácia e no aumento expressivo dos erros espaciais.

Em suma, os resultados sugerem que as ferramentas projetadas sob o paradigma *online* tendem a lidar melhor com a variação na taxa de amostragem, especialmente no que diz respeito à precisão espacial. Esses achados respondem à QP1, confirmando que a arquitetura do algoritmo influencia diretamente a tolerância à descontinuidade dos sinais.

6.3 Eficiência Temporal

Esta seção analisa a eficiência temporal das ferramentas avaliadas, respondendo à QP2, que investiga como a latência média de processamento varia em função da taxa de amostragem das trajetórias no paradigma *online*. A análise considera métricas temporais de ponta a ponta e métricas de latência por etapa, ambas mensuradas no lado do cliente, conforme definido na metodologia (subseção 4.2.3).

A latência foi calculada como o tempo transcorrido entre o envio de um ponto ou lote de pontos GNSS pelo *benchmark harness* e o recebimento do resultado correspondente do SUT. Essa métrica reflete o custo temporal percebido por uma aplicação consumidora, englobando serialização, comunicação, processamento interno e tempo de espera pela resposta. De forma complementar, a latência por etapa permite isolar o custo médio de processamento incremental, sendo particularmente relevante para comparar ferramentas que operam com diferentes

estratégias.

Os resultados de latência média em função da taxa de amostragem são apresentados na Tabela 10 e ilustrados na Figura 24. Observa-se um comportamento consistente entre todas as ferramentas, no qual a latência cresce à medida que a taxa de amostragem se torna mais esparsa. Esse efeito é esperado, pois intervalos maiores entre observações aumentam a ambiguidade espacial e topológica, elevando o custo de inferência por ponto.

Tabela 10 – Latência média de processamento variando a taxa de amostragem

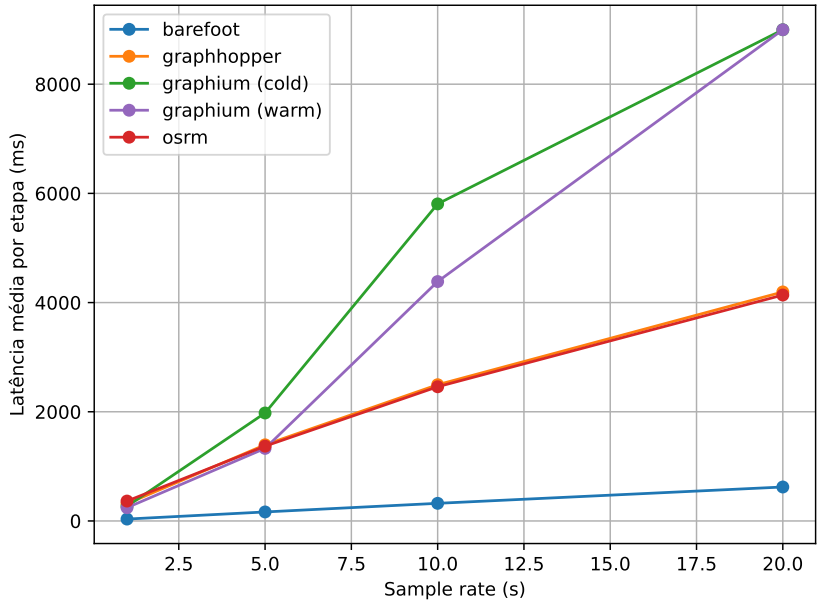
Ferramenta	Variante	Taxa (s)	Latência (ms)	Exec.
Barefoot	–	1	**34,62**	20
	–	5	165,09	20
	–	10	323,02	20
	–	20	622,04	20
GraphHopper	–	1	315,50	60
	–	5	1.394,59	60
	–	10	2.495,68	60
	–	20	4.194,82	60
GraphiumMM	*Cold*	1	268,71	59
		5	1.977,63	60
		10	5.807,31	60
		20	8.997,62	60
	Warm	1	243,30	59
		5	1.329,03	60
		10	4.384,96	60
		20	8.997,62	60
OSRM	–	1	364,60	51
	–	5	1.370,04	60
	–	10	2.458,18	60
	–	20	4.138,27	60

Fonte: Elaborada pelo autor.

O Barefoot apresentou os menores valores de latência em todos os cenários avaliados. Mesmo nas taxas de amostragem mais baixas, a latência média permaneceu em patamares significativamente inferiores aos das demais ferramentas. Esse resultado é coerente com sua arquitetura nativamente orientada ao paradigma *online*, que mantém estado contínuo e utiliza comunicação baseada em *sockets*, reduzindo o *overhead* de interação.

O GraphHopper e o OSRM exibiram comportamentos muito semelhantes entre si ao longo de toda a faixa de taxas de amostragem. Em cenários de alta frequência de coleta, ambas apresentaram latências moderadas, que cresceram de forma progressiva à medida que os dados se tornaram mais esparsos. Ainda assim, o GraphHopper apresentou valores sistematicamente superiores aos do OSRM, sobretudo nas taxas de amostragem mais baixas. Esse comportamento

Figura 24 – Comparação de latência por ferramenta ao variar a taxa de amostragem



Fonte: Elaborado pelo autor.

reflete o custo adicional de seu fluxo de processamento originalmente *offline*, quando adaptado para execução incremental, enquanto o OSRM se beneficia de uma implementação mais enxuta do ponto de vista computacional.

O GraphiumMM apresentou os maiores valores de latência média entre todas as ferramentas, especialmente em cenários de maior esparsidade. Esse comportamento está diretamente associado à dependência de um contexto inicial mais amplo para a estabilização do algoritmo, conforme discutido na subseção 6.1.1. Essa característica motivou a avaliação de duas variantes experimentais: GraphiumMM (*cold*), que inclui o custo completo de inicialização, e GraphiumMM (*warm*), na qual o impacto do *cold start* é desconsiderado.

A comparação entre essas variantes evidencia de forma clara o impacto do contexto inicial no custo temporal. Conforme apresentado na Tabela 10, a variante GraphiumMM (*warm*) apresenta reduções substanciais de latência em relação à variante *cold*, especialmente nas taxas de amostragem intermediárias. Apesar disso, mesmo na configuração *warm*, os valores de latência permanecem superiores aos observados para Barefoot, OSRM e GraphHopper, indicando que uma parte relevante do custo temporal está associada à própria lógica de processamento incremental do GraphiumMM, e não apenas à sua fase de inicialização.

Essa diferença também se manifesta na análise da latência em função do tamanho do lote, apresentada na Tabela 11 e na Figura 25. Para todas as ferramentas, o aumento do *batch size* resulta em um crescimento consistente da latência média por etapa. No caso do GraphiumMM,

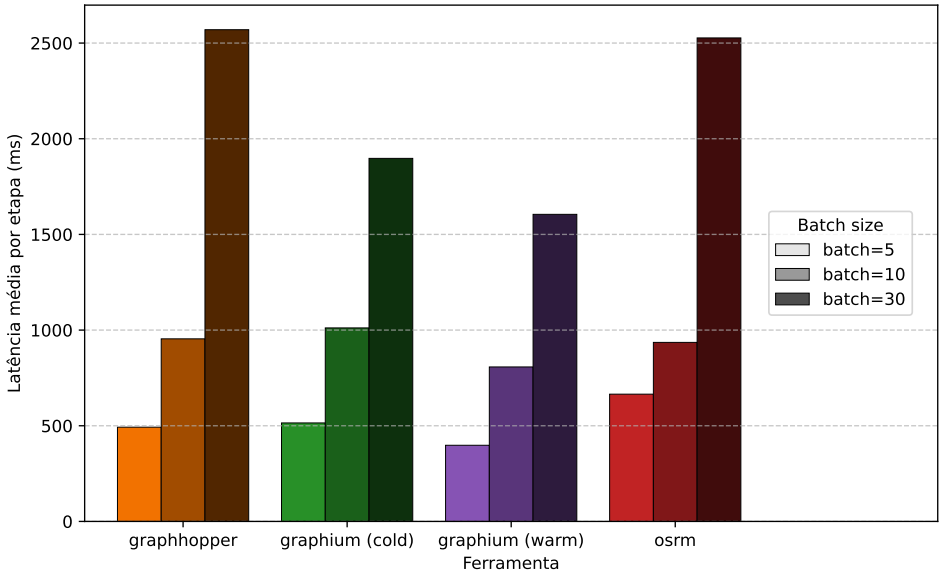
a variante *warm* apresenta latências sistematicamente inferiores à variante *cold* para todos os tamanhos de lote avaliados, confirmando que o *cold start* exerce um impacto significativo no custo temporal. Ainda assim, mesmo na configuração *warm*, observa-se que o crescimento da latência com lotes maiores permanece mais acentuado do que nas demais ferramentas, sugerindo maior complexidade computacional por etapa.

Tabela 11 – Latência média de processamento variando o tamanho do lote

Ferramenta	Lote (Batch)	Latência (ms)	Exec.
GraphHopper	5	492,59	80
	10	954,26	80
	30	2.569,84	80
GraphiumMM	5	514,75	79
	10	1.011,40	80
	30	1.897,43	80
GraphiumMM (*warm*)	5	**398,06**	79
	10	**807,38**	80
	30	**1.604,94**	80
OSRM	5	665,15	71
	10	935,73	80
	30	2.526,91	80

Fonte: Elaborada pelo autor.

Figura 25 – Latência média por etapa (ms) para as ferramentas e tamanhos de batch



Fonte: Elaborado pelo autor.

Por fim, o tempo total de execução ponta a ponta, apresentado na Tabela 12 e na Figura 26, permite avaliar o efeito acumulado dessas latências ao longo de trajetórias completas. O Barefoot e o OSRM apresentaram os menores tempos totais médios, com variação limitada em

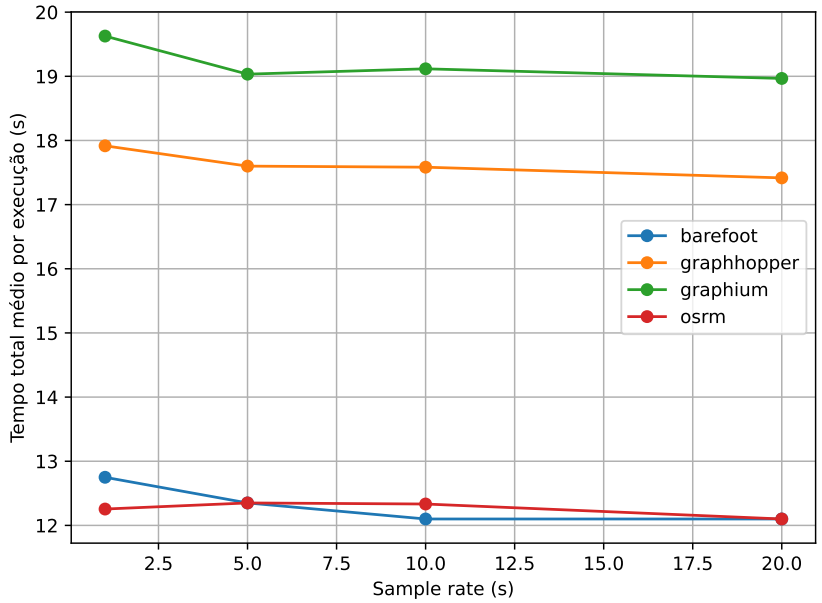
função da taxa de amostragem. O GraphHopper apresentou os maiores tempos totais, refletindo o acúmulo de latências por etapa ao longo da execução. O GraphiumMM situou-se em uma faixa intermediária, com tempos totais próximos aos do GraphHopper, ainda que motivados por fatores distintos, especialmente o custo de inicialização e a complexidade do processamento incremental.

Tabela 12 – Tempo médio de execução ponta a ponta variando a taxa de amostragem

Ferramenta	Taxa (s)	Tempo E2E (s)	Exec.
Barefoot	1, 5, 10, 20	$\approx$ 12,3	80
OSRM	1	12,25	51
	5	12,35	60
	10	12,33	60
	20	**12,10**	60
GraphHopper	1	17,92	60
	5	17,60	60
	10	17,58	60
	20	17,42	60
GraphiumMM	1	19,63	59
	5	19,03	60
	10	19,12	60
	20	18,97	60

Fonte: Elaborada pelo autor.

Figura 26 – Tempo de execução das ferramentas ao variar a taxa de amostragem



Fonte: Elaborado pelo autor.

Em síntese, os resultados indicam que ferramentas projetadas nativamente para o paradigma *online* apresentam maior eficiência temporal e maior previsibilidade frente à

variação da taxa de amostragem. Em contrapartida, soluções originalmente *offline*, embora funcionais quando adaptadas, demonstram maior sensibilidade à esparsidade dos dados. No caso específico do GraphiumMM, a análise das variantes *cold* e *warm* evidencia que o contexto inicial desempenha um papel central no custo temporal, mas não explica integralmente as latências elevadas observadas, apontando para um custo intrínseco mais alto de sua estratégia incremental.

6.4 Eficiência Computacional

Esta seção analisa a eficiência computacional das ferramentas em resposta à QP3, investigando a correlação entre o custo de hardware e a qualidade da correspondência sob diferentes regimes de amostragem. A análise combina o consumo de CPU e memória no servidor com as métricas de acurácia, permitindo identificar o compromisso (*trade-off*) entre exigência computacional e precisão espacial.

Os dados agregados de consumo de recursos computacionais apresentados nesta seção foram obtidos a partir da instrumentação *server-side* descrita na subseção 4.3.2, na qual métricas de CPU e memória foram coletadas continuamente durante a execução de cada experimento. Essas métricas foram depois agregadas por ferramenta e taxa de amostragem, resultando nos valores médios e de pico da Tabela 13.

A evolução do consumo médio de CPU e de memória em função da taxa de amostragem é detalhada, respectivamente, nas Figuras 27 e 28, permitindo uma análise comparativa do comportamento das ferramentas sob diferentes níveis de densidade amostral. De forma geral, observa-se que a diminuição da densidade amostral atenua a carga sobre o processador, uma vez que o volume de observações processadas por unidade de tempo se torna significativamente menor. Esse comportamento é consistente com o modelo de execução incremental adotado, no qual cada ponto GNSS implica uma etapa adicional de inferência e atualização do estado interno do algoritmo.

O OSRM posiciona-se como um *outlier* em eficiência. Em todos os cenários, seu custo computacional foi quase zero, com uso de CPU na escala de milésimos de porcentagem e um consumo de memória que decresceu de 11 MB para meros 3 MB em trajetórias esparsas. Esse desempenho excepcional é reflexo da natureza de sua implementação em C++[1] e do uso de estruturas de dados altamente otimizadas. Contudo, embora seja a solução mais econômica, sua acurácia sofre uma degradação severa sob baixa frequência de amostragem, o que limita sua

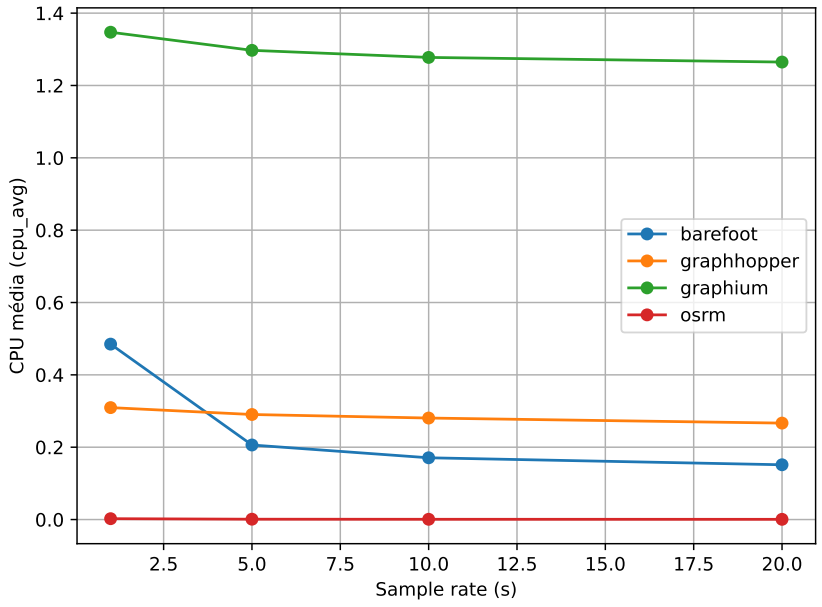
[1] https://isocpp.org/

Tabela 13 – Consumo de recursos computacionais das ferramentas variando a taxa de amostragem

Ferramenta	Taxa (s)	CPU Média (%)	Mem. Média (MB)	CPU Total (s)	Mem. Pico (MB)
Barefoot	1	0,48	308,17	6,49	448,66
	5	0,21	174,97	2,49	219,31
	10	0,17	151,10	2,00	199,83
	20	0,15	125,33	1,79	173,11
GraphHopper	1	0,31	331,23	5,15	375,37
	5	0,29	328,49	4,70	370,43
	10	0,28	325,51	4,55	367,57
	20	0,27	320,58	4,27	361,64
GraphiumMM	1	1,35	1.444,64	25,66	1.487,76
	5	1,30	1.447,35	24,00	1.487,02
	10	1,28	1.448,58	23,70	1.487,85
	20	1,26	1.449,32	23,19	1.488,28
OSRM	1	0,002	10,91	0,03	14,97
	5	0,001	4,85	0,01	6,47
	10	0,001	3,50	0,01	4,46
	20	**0,001**	**2,78**	**0,01**	**3,31**

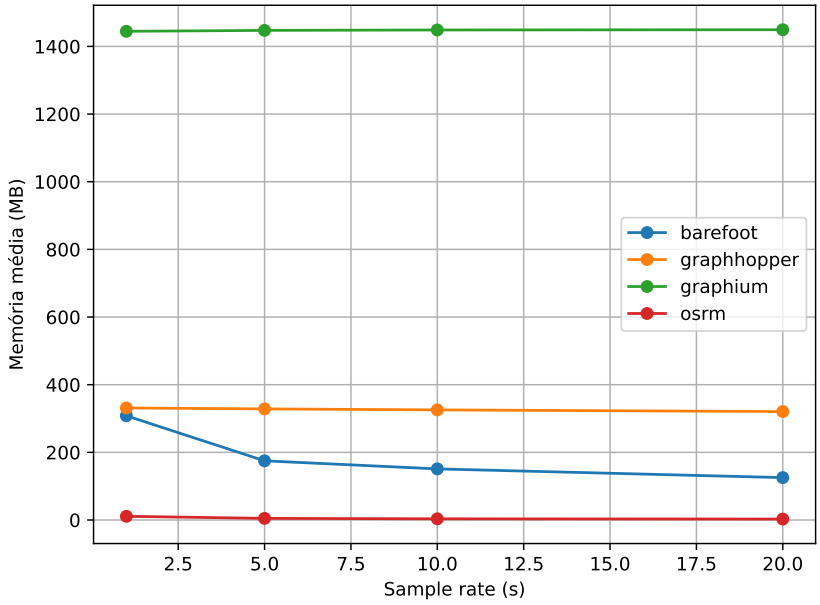
Fonte: Elaborada pelo autor.

Figura 27 – Uso médio de CPU variando a taxa de amostragem



Fonte: Elaborado pelo autor.

Figura 28 – Uso médio de memória variando a taxa de amostragem



Fonte: Elaborado pelo autor.

aplicabilidade a contextos de alta densidade, onde o baixo custo de infraestrutura é a restrição primária.

O Barefoot, por sua vez, estabelece a fronteira da eficiência prática deste *benchmark*. Com um consumo de CPU variando entre $0,48\%$ e $0,15\%$ e uma demanda de memória entre 308 MB e 125 MB, a ferramenta entrega os maiores índices de F1-score e os menores erros espaciais. Esse equilíbrio demonstra que a arquitetura do Barefoot utiliza os recursos com parcimônia, convertendo o consumo computacional em precisão de forma mais eficaz do que seus concorrentes e consolidando-se como a escolha ideal para sistemas de missão crítica.

Em contrapartida, o GraphiumMM apresentou a maior carga operacional, especialmente em termos de memória, mantendo-se estagnado em patamares de $1,44$ GB. Tal comportamento é um subproduto de sua arquitetura: ao operar como um *plugin* do Neo4j, o consumo reflete o *overhead*[2] do SGBD e da *Java Virtual Machine* / Máquina Virtual Java (JVM), independentemente da carga de trabalho. A despeito de oferecer qualidade competitiva, o GraphiumMM exige uma infraestrutura robusta, tornando-o menos viável para dispositivos de borda ou ambientes com recursos restritos.

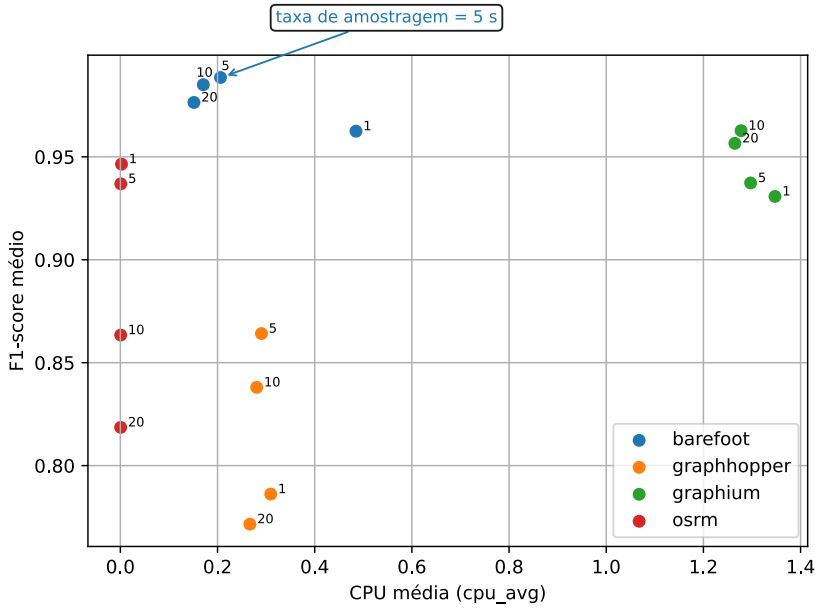
O GraphHopper apresentou um perfil de consumo intermediário (≈ 320 MB), mas com um retorno de investimento qualitativo inferior. Diferente do Barefoot, o custo computacional do GraphHopper não se traduz em acurácia proporcional, resultando em F1-scores

[2] Excesso de recursos consumidos para realizar uma tarefa específica ou gerenciar um sistema.

mais baixos e erros espaciais elevados. No contexto deste trabalho, a ferramenta demonstrou um balanço desfavorável, falhando em se destacar tanto em economia de recursos quanto em precisão de inferência.

A Figura 29 sintetiza esse panorama. O gráfico de dispersão revela claramente os regimes de operação: o OSRM como a opção mais econômica, o Barefoot como o melhor compromisso entre custo e benefício e o GraphiumMM como uma alternativa de alto custo. Assim, a escolha da ferramenta depende das restrições do ambiente: onde há pouca memória ou pouca CPU, prevalece o OSRM, já em casos nos quais a precisão é prioritária, o Barefoot torna-se soberano.

Figura 29 – Qualidade (F1) vs custo (CPU)



Fonte: Elaborado pelo autor.

6.5 Capacidade de Processamento Contínuo

Esta seção analisa a capacidade de processamento contínuo das ferramentas avaliadas, respondendo à QP4, que investiga como a vazão média varia em função da taxa de amostragem das trajetórias. A análise considera duas métricas complementares de vazão: a vazão em pontos por segundo (PPS), que reflete a capacidade de ingestão de observações GNSS, e a vazão espacial em quilômetros por segundo (km/s), que expressa a quantidade de trajetória efetivamente processada ao longo do tempo.

Os valores agregados de vazão em função da taxa de amostragem são apresentados

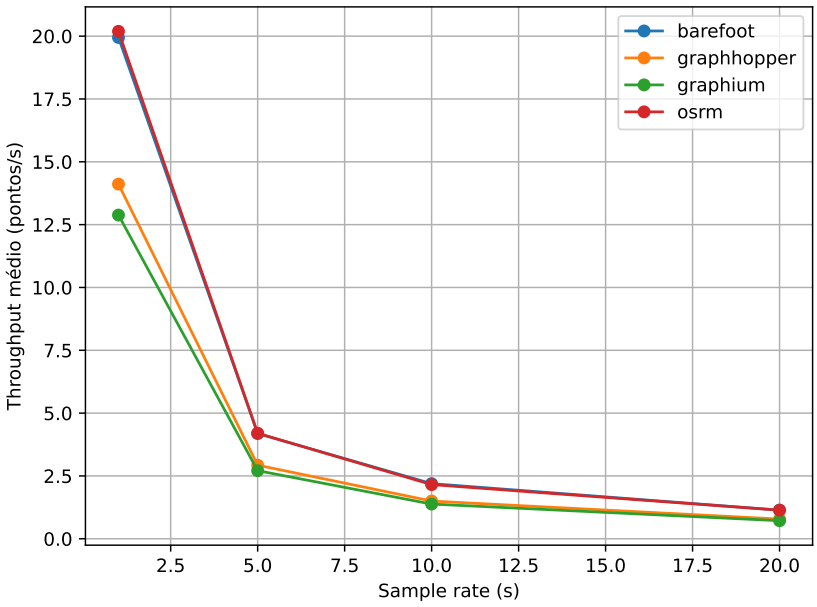
na Tabela 14, enquanto sua visualização gráfica é mostrada na Figura 30 e na Figura 31. Observa-se, de forma consistente entre todas as ferramentas, que a vazão expressa em PPS decresce quase proporcionalmente à medida que a taxa de amostragem aumenta. Esse comportamento é esperado, uma vez que intervalos amostrais mais longos implicam menor número de pontos emitidos por unidade de tempo, reduzindo naturalmente o volume de processamento contínuo.

Tabela 14 – Vazão de processamento em função da taxa de amostragem

Ferramenta	Taxa (s)	PPS Médio	PPS P95	Vazão (km/s)
Barefoot	1	19,95	23,43	0,126
	5	4,19	4,88	0,129
	10	2,19	2,50	0,132
	20	1,14	1,27	0,133
OSRM	1	**20,19**	**23,85**	**0,189**
	5	4,20	4,96	0,189
	10	2,16	2,52	0,189
	20	1,14	1,32	0,204
GraphHopper	1	14,11	17,83	0,089
	5	2,93	3,59	0,091
	10	1,50	1,84	0,091
	20	0,78	0,96	0,091
GraphiumMM	1	12,88	16,36	0,083
	5	2,71	3,21	0,083
	10	1,38	1,66	0,083
	20	0,72	0,86	0,083

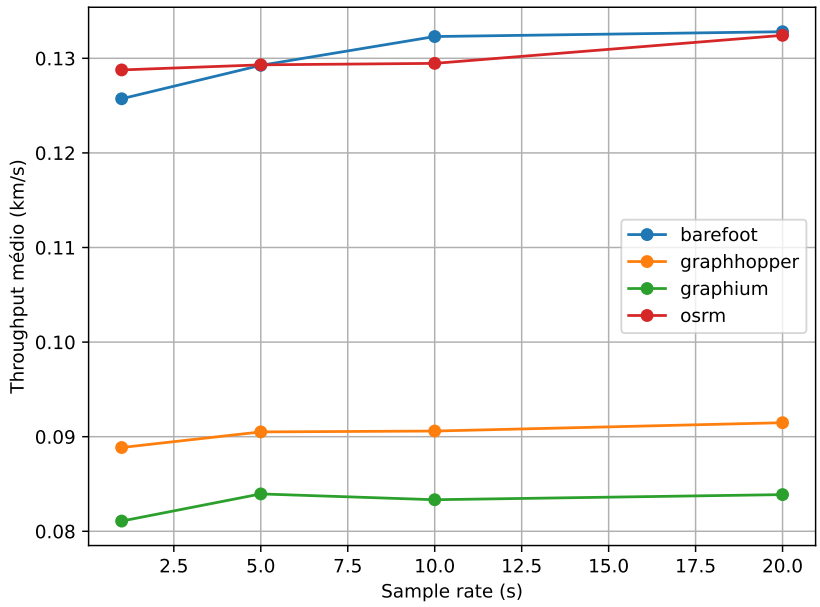
Fonte: Elaborada pelo autor.

Figura 30 – Vazão média (pontos por segundo) em função da taxa de amostragem



Fonte: Elaborado pelo autor.

Figura 31 – Vazão média (km/s) em função da taxa de amostragem



Fonte: Elaborado pelo autor.

Em contraste, a vazão espacial em km/s apresenta um comportamento notavelmente mais estável. Em particular, o OSRM mantém valores praticamente constantes em torno de 0,19–0,20 km/s em todas as taxas de amostragem avaliadas, enquanto o Barefoot permanece estável na faixa de aproximadamente 0,13 km/s. Esse resultado indica que, para essas ferramentas, o custo computacional dominante não está associado à recepção dos pontos individualmente, mas sim à exploração do espaço de estados no grafo viário. Em outras palavras, mesmo com dados mais esparsos, essas soluções mantêm uma velocidade quase constante de varredura espacial da rede.

O GraphHopper e o GraphiumMM, por sua vez, apresentam vazão espacial inferior, em torno de 0,09 km/s e 0,083 km/s, respectivamente. Embora também exibam estabilidade em km/s, esses valores indicam uma capacidade de processamento espacial cerca de 40% menor em comparação ao OSRM, reforçando o *trade-off* observado nas análises de eficiência computacional e temporal.

Além das médias, foi considerado o percentil 95 (P95) da vazão em PPS, também apresentado na Tabela 14. A proximidade entre os valores médios e o P95 sugere um comportamento estável, com baixa variabilidade e ausência de picos significativos de degradação. Essa característica é particularmente desejável em sistemas *online*, pois indica uma capacidade sustentada de processamento contínuo sob diferentes regimes de amostragem.

A influência do processamento em lote foi avaliada separadamente, conforme apre-

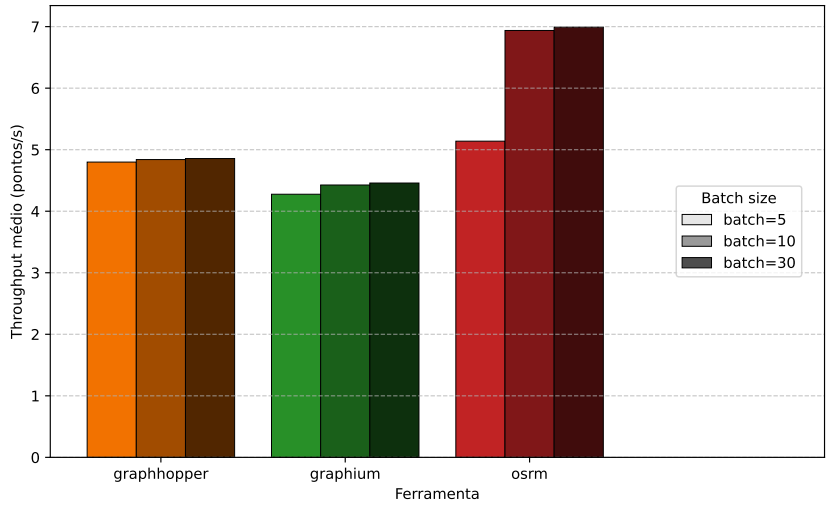
sentado na Tabela 15, na Figura 32 e na Figura 33. Observa-se que o OSRM se beneficia de forma clara do aumento do tamanho do lote, apresentando um ganho significativo de vazão em PPS ao passar de lotes de 5 para 10 pontos, seguido de estabilização em tamanhos maiores. Esse resultado aponta para um ganho de escala moderado, em que o custo fixo de processamento é distribuído de forma mais eficiente à medida que o tamanho dos lotes aumenta.

Tabela 15 – Vazão média de processamento por ferramenta variando o tamanho do lote

Ferramenta	**Tamanho do Lote (*Batch Size*)**	**PPS Médio**	**PPS P95**	**Vazão (km/s)**
OSRM	5	5,14	21,09	0,131
	10	6,94	22,34	0,129
	30	**6,99**	**22,66**	**0,130**
GraphHopper	5	4,80	16,40	0,090
	10	4,84	16,40	0,091
	30	4,86	16,77	0,090
GraphiumMM	5	4,28	15,26	0,083
	10	4,43	14,94	0,083
	30	4,46	15,63	0,083

Fonte: Elaborada pelo autor.

Figura 32 – Vazão média (pontos por segundo) em função do tamanho do lote

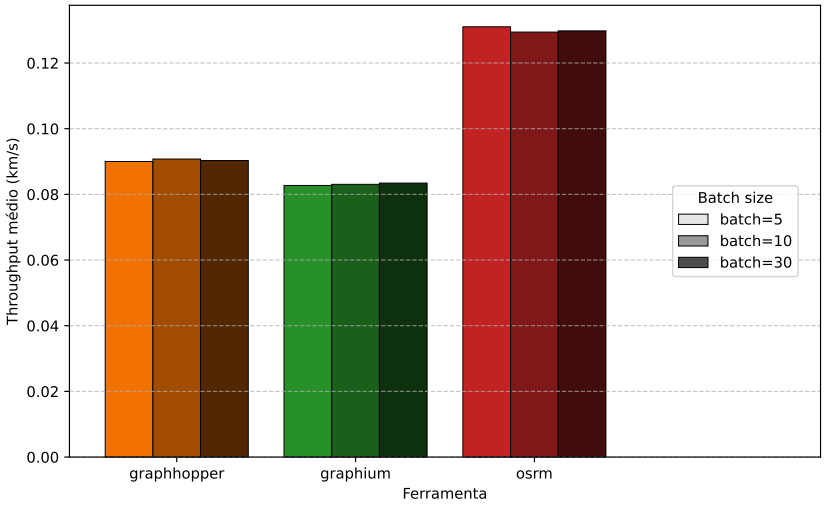


Fonte: Elaborado pelo autor.

Em contraste, o GraphHopper e o GraphiumMM exibem curvas de vazão praticamente planas em função do tamanho do lote. Independentemente de o processamento ocorrer com 5, 10 ou 30 pontos por lote, a vazão em PPS permanece quase constante. Esse padrão sugere uma saturação da vazão, na qual o gargalo está associado à lógica interna do algoritmo ou à sobrecarga da plataforma de execução, e não à forma de ingestão dos dados.

Em síntese, os resultados indicam que o OSRM apresenta a maior capacidade de

Figura 33 – Vazão média (km/s) por ferramenta variando o lote



Fonte: Elaborado pelo autor.

processamento contínuo, tanto em termos de PPS quanto de vazão espacial, combinando alta eficiência e estabilidade. O Barefoot também se destaca positivamente, com vazão consistente e comportamento previsível. Já o GraphHopper e o GraphiumMM demonstram menor capacidade de processamento espacial e pouca sensibilidade a estratégias de processamento em lote, o que pode limitar sua escalabilidade em cenários de alto volume e múltiplos fluxos simultâneos.

6.6 Síntese dos Resultados

Esta seção consolida os principais achados do *benchmark* e responde diretamente às questões de pesquisa. A análise conjunta das métricas de qualidade, latência, uso de recursos e vazão mostra que as ferramentas não competem em apenas um critério, pois cada uma ocupa uma posição específica no panorama de opções, com vantagens nítidas em contextos determinados.

I. **Qualidade de correspondência (QP1).** A robustez à esparsidade foi um divisor de águas. O Barefoot apresentou a melhor acurácia global e permaneceu estável mesmo com amostragem a cada 20 s, com altos valores de F1-score e baixos erros espaciais. O GraphiumMM sustentou desempenho competitivo, sobretudo em taxas intermediárias, e mostrou boa capacidade de recuperação geométrica quando o contexto já estava estabelecido. Em contraste, OSRM e GraphHopper foram mais sensíveis à perda de resolução, com degradação consistente da qualidade conforme a amostragem se tornava mais esparsa.

II. **Eficiência temporal (QP2).** Em latência percebida pelo cliente, o Barefoot foi a referência. Ele respondeu mais rápido em todas as taxas, refletindo uma arquitetura nativamente *online*

implementada de forma assíncrona usando ZeroMQ com estado contínuo e baixo custo de interação. O OSRM e o GraphHopper seguiram um comportamento semelhante, apresentando maior latência quando a amostragem é menos densa, o que evidencia o custo de inferência sob maior incerteza. O GraphiumMM teve as maiores latências, e a comparação entre as variantes confirmou que o *cold start* pesa, mas não explica tudo. Mesmo em regime *warm*, o custo incremental por etapa permanece elevado.

III. **Eficiência computacional (QP3).** O OSRM foi o campeão em economia. Seu consumo de CPU e memória ficou muito abaixo dos demais, o que o torna atraente quando a infraestrutura é o fator limitante, desde que a aplicação opere com alta densidade de observações. O Barefoot entregou o melhor equilíbrio entre custo e benefício, convertendo um consumo moderado em alta precisão. O GraphiumMM impôs o maior custo, sobretudo em memória, devido ao efeito estrutural da sua execução como *plugin* no Neo4j e ao *overhead* da JVM. O GraphHopper ocupou uma zona intermediária de consumo, mas sem retorno proporcional em qualidade no cenário avaliado.

IV. **Capacidade de processamento contínuo (QP4).** Em vazão, o OSRM liderou tanto em PPS quanto em km/s e foi a ferramenta que mais se beneficiou da adaptação de processamento em lote, indicando boa amortização de custo fixo por requisição. O Barefoot apresentou vazão estável e previsível, com comportamento consistente em diferentes taxas de amostragem. GraphHopper e GraphiumMM exibiram sinais de saturação, com vazão quase invariável ao variar o tamanho do lote e menor velocidade espacial, o que limita a escalabilidade quando há múltiplos fluxos concorrentes.

Em conjunto, os resultados mostram que não há uma solução universalmente superior, mas perfis distintos que atendem a diferentes requisitos. Dessa forma, essa análise integrada fornece subsídios concretos para a escolha informada de ferramentas de *map matching* em aplicações reais.

6.7 Discussão dos Resultados

Além das respostas às questões de pesquisa apresentadas na seção 6.6, os resultados permitem discutir as implicações práticas da adoção, considerando não apenas a qualidade e o desempenho, mas também o esforço de integração e operação em um ambiente de produção. Em aplicações reais, a ferramenta escolhida precisa atender simultaneamente aos requisitos de acurácia, latência, custo de infraestrutura, vazão e complexidade de implantação, o que introduz

um eixo adicional de decisão que não é capturado apenas por métricas numéricas.

Nesse contexto, a subseção 6.7.1 traduz os achados experimentais em diretrizes objetivas de escolha, sintetizando quando cada ferramenta tende a ser mais adequada. Em seguida, a subseção 6.7.2 complementa essa leitura ao discutir aspectos de implementação e prontidão para produção, destacando fatores que afetam diretamente o custo de adoção, como o modelo de comunicação, dependências de infraestrutura e exigências operacionais.

6.7.1 Diretrizes práticas de escolha

Com base nos achados apresentados, a seleção das ferramentas pode ser resumida em diretrizes objetivas orientadas por restrições de operação:

- **Priorize o Barefoot** quando a aplicação exigir alta acurácia, mesmo com trajetórias esparsas, e quando baixa latência for um requisito central em processamento *online*, mantendo uma vazão elevada com consumo de recursos aceitável.
- **Priorize o OSRM** quando o fator limitante for a infraestrutura e a simplicidade de adoção, sobretudo em cenários com alta densidade amostral, nos quais seu custo computacional mínimo e sua alta vazão favorecem a execução em escala.
- **Considere o GraphiumMM** quando a meta for boa qualidade e houver disponibilidade de memória e tolerância a uma maior latência, especialmente em cenários em que a integração com uma base de dados orientada a grafos seja desejável.
- **Evite o GraphHopper como primeira opção** no cenário avaliado, pois ele não apresentou vantagem dominante em qualidade, latência, custo computacional ou vazão quando comparado às alternativas.

6.7.2 Considerações da implementação e prontidão para produção

Os resultados também evidenciam diferenças relevantes no caminho de adoção. Do ponto de vista da engenharia, existe um compromisso entre desempenho e complexidade operacional, em que soluções com melhor custo-benefício podem demandar maior esforço de implantação e manutenção.

O Barefoot apresenta o melhor equilíbrio geral entre qualidade, latência, consumo de recursos e vazão no cenário avaliado. Entretanto, sua adoção tende a ser mais trabalhosa do que as alternativas baseadas em HTTP, pois o sistema é distribuído e envolve múltiplos componentes de servidor. Além disso, a comunicação é realizada via *sockets* e conexão com ZeroMQ, e

não por uma interface HTTP tradicional. Esse detalhe aumenta o esforço de integração e observabilidade, pois exige bibliotecas específicas no cliente, um modelo de troca de mensagens diferente e cuidados adicionais na operação. Em contrapartida, essa escolha arquitetural reduz *o overhead* de comunicação e tende a favorecer o desempenho em cenários com múltiplos veículos simultâneos, o que é compatível com a baixa latência e a alta vazão observadas no *benchmark*.

O OSRM, por outro lado, combina baixo custo computacional com alta vazão e oferece uma experiência de integração mais direta. A interface HTTP torna o consumo do serviço simples e amplamente compatível com arquiteturas modernas, e a adaptação *online* , por meio de lógica incremental no cliente, segue o procedimento descrito na metodologia. Na prática, isso reduz barreiras à adoção e facilita a instrumentação e a implantação. Embora sua qualidade se degrade em amostragens mais esparsas, ela se mantém atrativa quando a restrição principal é a infraestrutura e quando o regime de coleta é denso, o que a posiciona como uma alternativa operacionalmente eficiente em cenários de produção com foco em custo.

O GraphHopper compartilha a mesma vantagem de integração via HTTP e, portanto, apresenta um caminho de adoção semelhante ao do OSRM, com lógica de lotes implementada no lado do cliente. Contudo, no recorte experimental avaliado, seu consumo e desempenho não se traduziram em ganhos competitivos de acurácia ou eficiência temporal, o que enfraquece sua atratividade como escolha padrão, apesar da facilidade de integração.

Já **o GraphiumMM** exige uma integração um pouco mais específica, pois a lógica de janela e de *buffering* é conduzida no cliente, e o algoritmo apresenta sensibilidade ao contexto inicial. Ainda assim, a comunicação permanece baseada em HTTP, o que mantém a integração acessível. Seu principal diferencial está no modelo de implantação como *plugin* do Neo4j. Embora isso eleve o consumo de memória e imponha um custo de infraestrutura mais alto, essa arquitetura traz duas implicações práticas. Primeiro, reduz os custos de movimentação interna de dados quando a aplicação já mantém informações e operações no ecossistema do banco, pois parte do processamento ocorre no mesmo ambiente lógico do grafo. Segundo, devido à natureza da sua implementação, a ferramenta herda todos os mecanismos de escalabilidade, administração e operação do Neo4j, o que pode ser vantajoso em cenários com alta concorrência e necessidade de integração estreita com consultas e serviços baseados em grafos.

Em síntese, a análise de implementação complementa as métricas experimentais ao tornar explícito o custo de adoção. O Barefoot destaca-se como uma solução de melhor custo-benefício sob a ótica de desempenho, porém, à custa de uma complexidade operacional mais

elevada. O OSRM maximiza a simplicidade e a eficiência da infraestrutura, com uma melhor adequação a cenários de alta taxa de amostragem. O GraphiumMM oferece uma alternativa competitiva quando a integração com um banco de grafos é estratégica e o custo de memória é aceitável. Assim, a escolha final depende tanto de restrições quantitativas quanto da maturidade do *stack* de produção e do esforço de engenharia disponível.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho propôs, implementou e aplicou um *benchmark* reprodutível para a avaliação de ferramentas de *map matching* de código aberto sob o paradigma *online*. A motivação central foi reduzir a lacuna entre as métricas teóricas reportadas na literatura e as demandas práticas de engenharia de software, oferecendo um processo sistemático para mensurar a qualidade da correspondência, a eficiência temporal e o custo computacional. As seções a seguir são organizadas da seguinte forma: a seção 7.1 sintetiza as principais contribuições técnicas e científicas resultantes deste estudo. Em seguida, a seção 7.2 apresenta as considerações finais e o posicionamento geral dos achados. Por fim, a seção 7.3 descreve direções de continuidade.

7.1 Contribuições

A realização deste estudo resultou em quatro contribuições principais para a área de processamento de trajetórias e engenharia de sistemas:

I. **Suíte Experimental e conjunto de dados sintético:** A criação de um conjunto de trajetórias com *ground truth* rigoroso, permitindo a avaliação da robustez à esparsidade de forma controlada e isolada de ruídos externos.
II. **Harness de Benchmarking Automatizado:** O desenvolvimento de uma infraestrutura baseada em contêineres que orquestra execuções, monitora o hardware e aplica políticas de tolerância a falhas, garantindo a reprodutibilidade dos experimentos em diferentes ambientes.
III. **Biblioteca `mmlib`:** A implementação de uma camada de abstração que padroniza a interface das ferramentas e viabiliza a execução incremental de algoritmos originalmente não nativos ao fluxo *online*, facilitando a extensão deste estudo para novos softwares.
IV. **Caracterização Comparativa Multidimensional:** A produção de um quadro comparativo inédito que correlaciona acurácia espacial com métricas de vazão (*throughput*) e latência, oferecendo evidências empíricas para apoiar decisões de arquitetura em sistemas de monitoramento*online*.

7.2 Considerações Finais

Os experimentos realizados evidenciaram perfis operacionais distintos e confirmaram que a escolha da ferramenta ideal é condicionada às restrições do ambiente. O **Barefoot**

consolidou-se como a solução de maior acurácia e melhor equilíbrio entre precisão e custo. Por outro lado, o **OSRM**, embora nativamente *offline*, demonstrou-se um *outlier* em eficiência de hardware e vazão quando funcionando em modo de micro-lotes, sendo a escolha natural para cenários de alta densidade de dados e restrição de recursos.

Logo, em conjunto, os resultados indicam que não existe uma solução que seja superior em todos os casos, mas sim diferentes perfis capazes de atender a requisitos distintos. Assim, essa análise conjunta fornece subsídios concretos para a escolha informada de ferramentas de *map matching* em aplicações reais.

Além disso, ressalta-se que a arquitetura modular introduzida pela `mmlib` garante que este trabalho não seja um estudo estático, mas uma base extensível para a incorporação contínua de novos algoritmos e cenários de carga.

7.3 Trabalhos Futuros

Como desdobramentos desta pesquisa e visando a evolução da maturidade dos testes em sistemas de *map matching*, propõem-se os seguintes trabalhos:

- **Análise Estatística de Significância:** Investigar as correlações entre *o batch size* e a acurácia por meio de testes de hipóteses e cálculo de tamanho de efeito, quantificando de forma estatística os limites de cada ferramenta.
- **Expansão de Cenários e Conjuntos de Dados:** Validar o *framework* com conjuntos de dados reais e amplamente citados, como os de Melbourne e Seattle, para avaliar como o ruído de sensores reais (e não apenas a esparsidade) afeta o desempenho temporal.
- **Simulação de Alta Concorrência:** Ampliar o portfólio de métricas para incluir cenários de estresse com múltiplas trajetórias simultâneas, simulando cargas de produção de cidades inteligentes para medir a degradação da latência sob disputa de recursos.

REFERÊNCIAS

BARTZ-BEIELSTEIN, T.; DOERR, C.; BERG, D. van den; BOSSEK, J.; CHANDRASEKARAN, S.; EFTIMOV, T.; FISCHBACH, A.; KERSCHKE, P.; CAVA, W. L.; LOPEZ-IBANEZ, M.; MALAN, K. M.; MOORE, J. H.; NAUJOKS, B.; ORZECHOWSKI, P.; VOLZ, V.; WAGNER, M.; WEISE, T. **Benchmarking in Optimization: Best Practice and Open Issues**. arXiv, 2020. ADS Bibcode: 2020arXiv200703488B. Disponível em: https://ui.adsabs.harvard.edu/abs/2020arXiv200703488B. Acesso em: 02 jul. 2025.

BERNSTEIN, D.; KORNHAUSER, A. **An Introduction to Map Matching for Personal Navigation Assistants**. [*S. l.*], 1996. Disponível em: https://rosap.ntl.bts.gov/view/dot/38257. Acesso em: 18 jul. 2025.

BORGES, H.; VALENTE, M. T. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. **Journal of Systems and Software**, v. 146, p. 112–129, dez. 2018. ISSN 01641212. ArXiv:1811.07643 [cs]. Disponível em: http://arxiv.org/abs/1811.07643. Acesso em: 12 jan. 2026.

CHAO, P.; XU, Y.; HUA, W.; ZHOU, X. **A Survey on Map-Matching Algorithms**. arXiv, 2020. ArXiv:1910.13065 [cs]. Disponível em: http://arxiv.org/abs/1910.13065. Acesso em: 05 jul. 2025.

DOYLE, A. C. **Um estudo em vermelho**. [*S. l.*]: Domínio Público, 1887.

ERDMANN, J.; EBENDT, R. Benchmarking Trajectory Matchers with SUMO. In: CARVALHO, A.; MANUEL, J.; BRAGANCA, C. (Ed.). **Proceedings[...]**. Porto, Portugal: Eurosis-ETI, 2014. p. 226–230. ISBN 978-90-77381-86-1. Disponível em: https://elib.dlr.de/93676/. Acesso em: 16 jan. 2026.

FELDT ROBERT; MAGAZINIUS, A. **Validity threats in empirical software engineering research**: an initial survey. [S. l.]: [s. n.], 2010. 379 p. p.

FU, X.; ZHANG, J.; ZHANG, Y. An Online Map Matching Algorithm Based on Second-Order Hidden Markov Model. **Journal of Advanced Transportation**, v. 2021, n. 1, p. 9993860, 2021. ISSN 2042-3195. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/9993860. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/9993860. Acesso em: 23 jul. 2025.

GOH, C.; DAUWELS, J.; MITROVIC, N.; ASIF, M. T.; ORAN, A.; JAILLET, P. Online map-matching based on Hidden Markov model for real-time traffic sensing applications. In: INTERNATIONAL IEEE CONFERENCE ON INTELLIGENT TRANSPORTATION SYSTEMS, 15. **Proceedings [...]**. 2012. p. 776–781. ISSN: 2153-0017. Disponível em: https://ieeexplore.ieee.org/abstract/document/6338627. Acesso em: 3 jul. 2025.

GOOGLE LLC. **Planos de preços e custos de API**. [S. l.]: Google Cloud, 2025. Disponível em: https://mapsplatform.google.com/intl/pt-BR_br/pricing/. Acesso em: 7 jul. 2025.

GOOGLE LLC. **Waze: aplicativo de navegação por GPS**. [S. l.]: Google, 2025. Disponível em: https://www.waze.com. Acesso em: 2 jul. 2025.

GRAPHHOPPER. **Pricing**: Graphhopper directions api. [S. l.]: GraphHopper, 2025. Disponível em: https://www.graphhopper.com/pricing/. Acesso em: 7 jul. 2025.

GROVES, P. D.; JIANG, Z.; WANG, L.; ZIEBART, M. K. Intelligent Urban Positioning using Multi-Constellation GNSS with 3D Mapping and NLOS Signal Detection. In: INTERNATIONAL TECHNICAL MEETING OF THE SATELLITE DIVISION OF THE INSTITUTE OF NAVIGATION (ION GNSS 2012), 25. **Proceedings[...]**. 2012. p. 458–472. Disponível em: http://www.ion.org/publications/abstract.cfm?jp=p&articleID=10262. Acesso em: 05 jul. 2025.

GUASTELLA, D. A.; MONTERO-PORRAS, E.; MORALES-HERNáNDEZ, A.; BONTEMPI, G. **Traffic Modeling with SUMO: a Tutorial**. [*S. l.*]: arXiv, 2025.

HASHEMI, M.; KARIMI, H. A. A critical review of real-time map-matching algorithms: Current issues and future directions. **Computers, Environment and Urban Systems**, v. 48, p. 153–165, nov. 2014. ISSN 0198-9715. Disponível em: https://www.sciencedirect.com/science/article/pii/S0198971514000908. Acesso em: 02 jul. 2025.

HASSELBRING, W. Benchmarking as Empirical Standard in Software Engineering Research. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 25. **Proceedings[...]**. New York, NY, USA: Association for Computing Machinery, 2021. (EASE '21), p. 365–372. ISBN 978-1-4503-9053-8. Disponível em: https://dl.acm.org/doi/10.1145/3463274.3463361. Acesso em: 10 jan. 2026.

JIN, Z.; KIM, J.; YEO, H.; CHOI, S. Transformer-based Map Matching Model with Limited Ground-Truth Data using Transfer-Learning Approach. **Transportation Research Part C: Emerging Technologies**, v. 140, p. 103668, jul. 2022. ISSN 0968-090X. ArXiv:2108.00439 [cs]. Disponível em: http://arxiv.org/abs/2108.00439. Acesso em: 23 jul. 2025.

KEMPINSKA, K.; DAVIES, T.; SHAWE-TAYLOR, J. **Probabilistic map-matching using particle filters**. arXiv, 2016. ArXiv:1611.09706 [stat]. Disponível em: http://arxiv.org/abs/1611.09706. Acesso em: 23 jul. 2025.

KHAN, M. E.; KHAN, F. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. **International Journal of Advanced Computer Science and Applications (IJACSA)**, v. 3, n. 6, jul. 2012. ISSN 2156-5570. Publisher: The Science and Information (SAI) Organization Limited. Disponível em: https://thesai.org/Publications/ViewPaper?Volume=3&Issue=6&Code=IJACSA&SerialNo=3. Acesso em: 10 jan. 2026.

KISTOWSKI, J. v.; ARNOLD, J. A.; HUPPLER, K.; LANGE, K.-D.; HENNING, J. L.; CAO, P. How to Build a Benchmark. In: ACM/SPEC INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, 6. **Proceedings[...]**. New York, NY, USA: Association for Computing Machinery, 2015. (ICPE '15), p. 333–336. ISBN 978-1-4503-3248-4. Disponível em: https://dl.acm.org/doi/10.1145/2668930.2688819. Acesso em: 10 jan. 2026.

KUBIčKA, M.; CELA, A.; MOULIN, P.; MOUNIER, H.; NICULESCU, S. Dataset for testing and training of map-matching algorithms. In: IEEE INTELLIGENT VEHICLES SYMPOSIUM, IV. **Proceedings[...]**. 2015. p. 1088–1093. ISSN: 1931-0587. Disponível em: https://ieeexplore.ieee.org/abstract/document/7225829. Acesso em: 2 jul. 2025.

LI, H.; KULIK, L.; RAMAMOHANARAO, K. Spatio-temporal trajectory simplification for inferring travel paths. In: ACM SIGSPATIAL INTERNATIONAL CONFERENCE ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 22. **Proceedings[...]**. New York,

NY, USA: Association for Computing Machinery, 2014. (SIGSPATIAL '14), p. 63–72. ISBN 978-1-4503-3131-9. Disponível em: https://doi.org/10.1145/2666310.2666409. Acesso em: 21 jul. 2025.

LOU, Y.; ZHANG, C.; XIE, X.; ZHENG, Y.; WANG, W.; HUANG, Y. Map-matching for low-sampling-rate gps trajectories. In: ACM SIGSPATIAL CONFERENCE ON ADVANCES IN GEOGRAPHICAL INFORMATION SYSTEMS, 18. **Proceedings[...]**. 2009. Disponível em: https://www.microsoft.com/en-us/research/publication/map-matching-for-low-sampling-rate-gps-trajectories/. Acesso em: 23 jul. 2025.

MARON, C. A. F.; FERNANDES, L. G. Uma Suíte de Benchmarks Parametrizáveis para o Domínio de Processamento de Stream em Sistemas Multi-Core. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIãO SUL (ERAD-RS). **Anais [...]**. SBC, 2018. ISSN: 2595-4164. Disponível em: https://sol.sbc.org.br/index.php/eradrs/article/view/4723. Acesso em: 10 jan. 2026.

MATTHEIS, S.; AL-ZAHID, K. K.; ENGELMANN, B.; HILDISCH, A.; HOLDER, S.; LAZAREVYCH, O.; MOHR, D.; SEDLMEIER, F.; ZINCK, R. Putting the car on the map: A scalable map matching system for the open source community. In: JAHRESTAGUNG DER GESELLSCHAFT FÜR INFORMATIK, BIG DATA-KOMPLEXITÄT MEISTERN, INFORMATIK, 44. **Proceedings[...]**. Gesellschaft für Informatik e.V., 2014. p. 2109–2119. ISBN 978-3-88579-626-8. Disponível em: https://dl.gi.de/items/ef2f8aa9-adae-44e4-8854-de0f5b30ce5c. Acesso em: 6 jul. 2025.

MOHAMMADI, S.; SMYTH, A. W. **NLP-enabled Trajectory Map-matching in Urban Road Networks using a Transformer-based Encoder-decoder**. arXiv, 2025. ArXiv:2404.12460 [cs] version: 4. Disponível em: http://arxiv.org/abs/2404.12460. Acesso em: 17 jul. 2025.

NEWSON, P.; KRUMM, J. Hidden Markov map matching through noise and sparseness. In: ACM SIGSPATIAL INTERNATIONAL CONFERENCE ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 17. **Proceedings [...]**. New York, NY, USA: Association for Computing Machinery, 2009. (GIS '09), p. 336–343. ISBN 978-1-60558-649-6. Disponível em: https://doi.org/10.1145/1653771.1653818. Acesso em: 2 jun. 2025.

PISSARDINI, R. de S.; OLIVEIRA, R. H. de; VAZ, J. A.; FILHO, F. G. V. de A.; JUNIOR, E. S. da F. O problema do posicionamento para transporte terrestre no ambiente urbano. **Revista Brasileira de Geomática**, v. 5, n. 3, p. 380–403, 2017.

QUDDUS, M. A.; OCHIENG, W. Y.; NOLAND, R. B. Current map-matching algorithms for transport applications: State-of-the art and future research directions. **Transportation Research Part C: Emerging Technologies**, v. 15, n. 5, p. 312–328, 2007. ISSN 0968-090X. Disponível em: https://www.sciencedirect.com/science/article/pii/S0968090X07000265. Acesso em: 20 jul. 2025.

RABINER, L. A tutorial on hidden Markov models and selected applications in speech recognition. **IEEE**, v. 77, n. 2, p. 257–286, fev. 1989. ISSN 1558-2256. Disponível em: https://ieeexplore.ieee.org/document/18626. Acesso em: 7 jul. 2025.

REHRL, K.; GRöCHENIG, S.; WIMMER, M. Optimization and Evaluation of a High-Performance Open-Source Map-Matching Implementation. In: MANSOURIAN, A.; PILESJö, P.; HARRIE, L.; LAMMEREN, R. van (Ed.). **Geospatial Technologies for All**. Cham: Springer International Publishing, 2018. p. 251–270. ISBN 978-3-319-78208-9.

ROSS, S. Chapter 2 - random variables. In: ROSS, S. (Ed.). **Introduction to Probability Models (Eleventh Edition)**. Eleventh edition. Boston: Academic Press, 2014. p. 21–91. ISBN 978-0-12-407948-9. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780124079489000025. Acesso em: 7 jul. 2025.

ROSS, S. Chapter 4 - markov chains. In: ROSS, S. (Ed.). **Introduction to Probability Models (Eleventh Edition)**. Eleventh edition. Boston: Academic Press, 2014. p. 183–276. ISBN 978-0-12-407948-9. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780124079489000049. Acesso em: 7 jul. 2025.

SAKI, S.; HAGEN, T. A Practical Guide to an Open-Source Map-Matching Approach for Big GPS Data. **SN Computer Science**, v. 3, n. 5, p. 415, ago. 2022. ISSN 2661-8907. Disponível em: https://doi.org/10.1007/s42979-022-01340-5. Acesso em: 6 jul. 2025.

SANDVE, G. K.; NEKRUTENKO, A.; TAYLOR, J.; HOVIG, E. Ten Simple Rules for Reproducible Computational Research. **PLOS Computational Biology**, v. 9, n. 10, p. 1–4, out. 2013. Publisher: Public Library of Science. Disponível em: https://doi.org/10.1371/journal.pcbi.1003285. Acesso em: 6 jul. 2025.

UBER TECHNOLOGIES INC. **Uber: plataforma de mobilidade urbana**. [S. l.]: Uber, 2025. Disponível em: https://www.uber.com. Acesso em: 2 jul. 2025.

U.S. GOVERNMENT. **GPS Accuracy**. [S. l.]: GPS.gov, 2025. Disponível em: https://www.gps.gov/systems/gps/performance/accuracy/. Acesso em: 2 jul. 2025.

VITERBI, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. **IEEE Transactions on Information Theory**, v. 13, n. 2, p. 260–269, abr. 1967. ISSN 1557-9654. Disponível em: https://ieeexplore.ieee.org/abstract/document/1054010. Acesso em: 3 jul. 2025.

WHITE, C. E.; BERNSTEIN, D.; KORNHAUSER, A. L. Some map matching algorithms for personal navigation assistants. **Transportation Research Part C: Emerging Technologies**, v. 8, n. 1, p. 91–108, fev. 2000. ISSN 0968-090X. Disponível em: https://www.sciencedirect.com/science/article/pii/S0968090X00000267. Acesso em: 4 jul. 2025.

WÖLTCHE, A. Open source map matching with markov decision processes: A new method and a detailed benchmark with existing approaches. v. 27, n. 7, p. 1959–1991, 2023. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.13107. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1111/tgis.13107.

YUAN, J.; ZHENG, Y.; ZHANG, C.; XIE, W.; XIE, X.; SUN, G.; HUANG, Y. T-drive: driving directions based on taxi trajectories. In: SIGSPATIAL INTERNATIONAL CONFERENCE ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 18. **Proceedings [...]**. New York, NY, USA: Association for Computing Machinery, 2010. (GIS '10), p. 99–108. ISBN 9781450304283. Disponível em: https://doi.org/10.1145/1869790.1869807. Acesso em: 18 jul. 2025.

ZHANG, G.; HSU, L.-T. Performance assessment of GNSS diffraction models in urban areas. **NAVIGATION**, v. 68, n. 2, p. 369–389, 2021. ISSN 2161-4296. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/navi.417. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/navi.417. Acesso em: 3 jul. 2025.

ZHENG, Y.; XIE, X. Learning travel recommendations from user-generated gps traces. **ACM Trans. Intell. Syst. Technol.**, Association for Computing Machinery, New York, NY, USA, v. 2, n. 1, jan. 2011. ISSN 2157-6904. Disponível em: https://doi.org/10.1145/1889681.1889683. Acesso em: 18 jul. 2025.