Middleware de Inferência de Contexto baseado em Aprendizado Federado para IoT no Computing Continuum

Henrique de S. Santana[1], **Thais R. M. B. Silva**[1,3], **Fabrício A. Silva**[1,3], **Linnyer B. R. Aylon**[2,3]

[1] Universidade Federal de Viçosa (UFV) / Campus Florestal – MG

[2]Universidade Estadual de Maringá (UEM) – PR

[3]Manna Team

{henrique.s.santana,thais.braga,fabricio.asilva}@ufv.br, lbruiz@uem.br

Abstract. *The growth of the Internet of Things (IoT) increases data collection about environments, driving context-aware systems that rely on reasoning to extract relevant information. Although machine learning is promising for this task, challenges remain regarding resource usage and data privacy. Paradigms such as federated learning and fog computing enable the decentralization of processing and data, but their integration is complex, motivating the use of middleware solutions. Thus, this work proposes Micelio, a context-aware IoT middleware focused on reasoning through federated learning, supporting cloud, fog, and edge computing. A case study on trash classification for smart cities is presented, evaluated in a simulation environment, and showing efficient resource usage and accuracy superior to the baseline.*

Resumo. *O crescimento da Internet das Coisas (IoT) amplia a coleta de dados sobre os ambientes, impulsionando sistemas cientes de contexto que dependem de inferência para extrair informações relevantes. Embora o aprendizado de máquina seja promissor nessa tarefa, há desafios relacionados ao uso de recursos e privacidade. Paradigmas como aprendizado federado e computação em névoa permitem descentralizar processamento e dados, mas sua integração é complexa, motivando o uso de middlewares. Então, este trabalho propõe o Micelio, um middleware IoT ciente de contexto com foco em inferência via aprendizado federado, com suporte à computação em nuvem, névoa e borda. É apresentado um estudo de caso em classificação de lixo para cidades inteligentes, avaliado num ambiente simulação, e demonstrando o uso eficiente de recursos pelo middleware, e acurácia superior à linha de base.*

1. Introdução

A expansão da Internet das Coisas (IoT) promete um futuro com cada vez mais dispositivos diferentes captando dados sobre os ambientes físicos e atuando sobre eles. Esse volume de dados e heterogeneidade de tecnologias fornece tanto insumos quanto desafios para sistemas cientes de contexto, que precisam coletar, modelar, inferir e distribuir dados contextuais [Perera et al. 2013], os quais serão utilizados na implementação e/ou adaptação de um ou mais de seus serviços.

A inferência, em particular, consiste em derivar e/ou sintetizar novos contextos mais significativos a partir de contextos brutos. Se por um lado ela pode proporcionar uma melhor seleção de serviços, mais personalizados, por outro lado ela é uma tarefa muitas vezes de realização complexa, que requer um investimento em consumo de recursos normalmente escassos em equipamentos IoT.

O contínuo aumento da quantidade de objetos inteligentes coletando uma grande quantidade de dados sobre pessoas, lugares e objetos para variadas aplicações IoT não só potencializou o interesse e adoção da inferência de contextos para diferentes cenários, como possibilitou a utilização de modelos de aprendizado de máquina para este fim. Além de benefícios como a dispensa de se estabelecer regras lógicas a priori e a descoberta de mais e melhores contextos inferidos, o uso dos modelos de aprendizado, muitas vezes, pode diminuir a necessidade de espaço para o armazenamento de dados. Por outro lado, surgem questões sobre como e onde os modelos poderão ser treinados e utilizados, bem como preocupações quanto ao uso ético e seguro de dados.

Buscando mitigar algumas das principais preocupações quanto ao uso de aprendizado de máquina para a inferência de contextos em cenários IoT, dois novos paradigmas podem ser utilizados: o aprendizado federado (AF) [Baccour et al. 2022] e o continuum computacional [Al-Dulaimy et al. 2024]. O primeiro possibilita a construção de modelos de aprendizado menores e mais localizados, que podem ser posteriormente unidos, diminuindo ou eliminando a necessidade de trafegar os dados utilizados entre componentes da aplicação. Já o segundo inclui a possibilidade de utilização da computação em borda, névoa e nuvem para processamento da tarefa de inferência de contextos.

No entanto, integrar todas esses dados, dispositivos e técnicas não é trivial, em especial para os projetistas das aplicações IoT, que desejam apenas fazer uso dos contextos inferidos. Uma solução comumente utilizada neste caso é a adoção de camadas de abstração, tais como middlewares, para facilitar o desenvolvimento dos sistemas cientes de contexto IoT e particularmente o uso da inferência [Razzaque et al. 2015].

Diante desse cenário, este trabalho apresenta um middleware, chamado Micelio (**Mi**ddleware for **c**ontext r**e**asoning through federated **l**earning in the **Io**T computing continuum) com o objetivo de equipar aplicações cientes de contexto na IoT com recursos de inferência, empregando esses avanços científicos de forma mais transparente quanto possível para seus desenvolvedores. Em especial para o cenário de cidades inteligentes, em que objetos inteligentes e usuários estão espalhados em uma ampla área geográfica e em grande quantidade, torna-se interessante o uso da computação em névoa e do AF.

Para validar a proposta, é apresentado um estudo de caso utilizando o Micelio no desenvolvimento de uma aplicação de lixeiras inteligentes, e testado por meio de simulação. Os principais resultados mostraram que o uso dos recursos de rede foram satisfatórios, com tráfego controlado de dados e treinamento de modelos em tempo hábil. Além disso, os modelos finais usados na inferência de contexto obtiveram acurácia razoável, em 61% no melhor caso, acima do trabalho usado como comparação.

O restante do trabalho está organizado da seguinte forma. Na Seção 2 são apresentados os principais trabalhos relacionados encontrados na literatura. A Seção 3 apresenta o middleware proposto, descrevendo sua arquitetura e principais implementações. Na Seção 4 é descrito o estudo de caso, exemplificando o uso do middleware para uma

aplicação, e resultados obtidos através de uma simulação. Por fim, as conclusões e possíveis trabalhos futuros podem ser encontrados na Seção 5.

2. Trabalhos Relacionados

Dada a necessidade de organizar e integrar a heterogeneidade de dados e dispositivos inerente da IoT, diferentes middlewares e frameworks cientes de contexto já foram propostos na literatura. Em cada um desses trabalhos, são demonstradas melhorias nas diferentes etapas do ciclo de vida de contexto – coleta, modelagem, inferência e distribuição [Perera et al. 2013]. Dessas etapas, destacamos os middlewares que abordam a inferência ou raciocínio de contexto, que é onde se encaixa o objetivo deste trabalho.

O Senz [Zhang et al. 2017] é um middleware ciente de contexto focado no reconhecimento de atividade de usuários de dispositivos móveis. Os autores combinaram a coleta de contextos dos sensores de cada dispositivo com informações públicas sobre pontos de interesse e eventos próximos. O middleware agrega esses dados numa arquitetura de nuvem para treinar diferentes modelos como redes neurais e árvores de decisão.

Em [Mahieu et al. 2019], os autores propõem um modelo ontológico e um middleware para a construção de serviços voltados para a interação entre humanos e robôs no cenário da IoT. O middleware permite que uma aplicação faça consultas de contexto usando SPARQL, com a execução de regras semânticas sobre a ontologia, e integra essas consultas com atuadores robóticos. De forma semelhante, em [Symeonaki et al. 2020] também é apresentado um middleware operando com uma modelagem ontológica e uma arquitetura de nuvem, porém voltado para o cenário da agricultura inteligente. Foram utilizadas regras semânticas como mecanismo de inferência, além de técnicas de aprendizado de máquina para refinar a etapa de coleta de contexto, com destaque para o K-Means.

Os autores de [Michalakis et al. 2021] apresentam outro middleware ciente de contexto para IoT, usando uma modelagem ontológica de alto nível e abordagens híbridas de inferência de contexto. Essas abordagens incluem lógica difusa, regras semânticas e modelos de aprendizado de máquina supervisionados e não supervisionados. Enquanto a proposta do middleware é de uso geral, os autores apresentam um estudo de caso no qual o middleware é empregado numa aplicação para espaços culturais.

O SIM [Elkhodr et al. 2024] é um middleware IoT semântico integrado com soluções de blockchain e inteligência artificial (IA) para diferentes propósitos. Destaca-se o uso da IA nas etapas de anotação automática de eventos coletados pelo módulo IoT, análise e inferência em consultas semânticas, e análise de feedback de usuários.

De forma semelhante à maioria dos trabalhos supracitados, o Micelio também opera com um modelo ontológico extensível pela aplicação que o utiliza, e conta com recursos da computação em nuvem. No entanto, ao passo que a IA aparece de diferentes formas em cada trabalho, não foram encontrados middlewares IoT que integram AF com apoio da computação em névoa em suas estratégias de inferência de contexto. Logo, este trabalho pretende cobrir essa lacuna.

3. Arquitetura do middleware

O Micelio é um middleware IoT ciente de contexto, sendo assim, seu papel é fornecer às aplicações da Internet das Coisas facilidades para lidar com as principais tarefas do

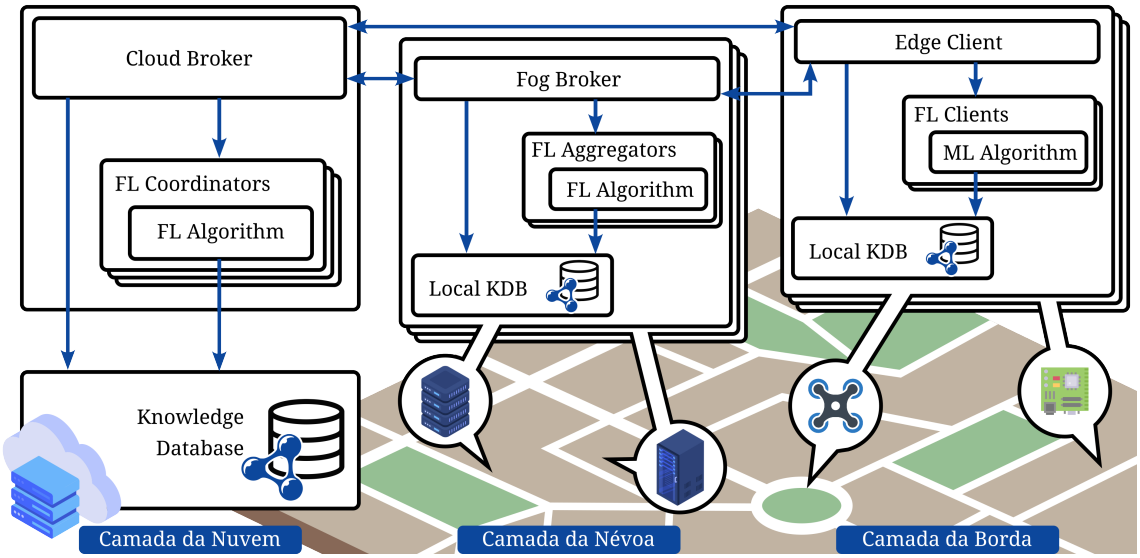


Figura 1. Visão geral da arquitetura interna do middleware.

gerenciamento de contexto: coleta, modelagem, inferência e distribuição. Para a modelagem, o middleware funciona com uma ontologia, possuindo definições próprias para entender e trabalhar em conjunto com o modelo da aplicação. Para a coleta e distribuição, há métodos tanto para registrar contextos quanto para consultá-los, sem que a aplicação se preocupe com as conexões envolvidas nessas operações. E por fim, o destaque da proposta está na inferência, que além de contar com o suporte à inferência ontológica, o middleware é capaz de treinar modelos de IA, usando AF, de tal forma que a aplicação precise apenas indicar quais contextos gostaria de inferir, e quais contextos podem ser usados como insumo para essa inferência.

Ao longo de todo o trabalho, para facilitar a compreensão da modelagem ontológica, será utilizada a notação da linguagem Turtle para referenciar elementos dessa ontologia, com o auxílio de prefixos recorrentes relacionados aos padrões da Web semântica, como `rdf:` e `rdfs:` para as definições do Resource Description Framework (RDF), `owl:` para a Web Ontology Language (OWL2), e o prefixo `mcl:` para indicar as definições do próprio modelo do middleware.

3.1. Camadas e entidades

Para atingir seu objetivo, a arquitetura interna do middleware é organizada em três camadas, correspondendo a três níveis hierárquicos comumente abordados no continuum computacional: nuvem (*cloud*), névoa (*fog*) e borda (*edge*). A Figura 1 ilustra a visão geral da arquitetura e suas entidades.

Na nuvem, onde há amplos recursos computacionais, situam-se o *Cloud Broker*, a *Knowledge Database* e os *Federated Learning Coordinators*. O *Cloud Broker* serve como ponto de entrada para a aplicação, oferecendo operações como a consulta de contexto e configuração das tarefas de inferência, além de atuar como ponto centralizador de repasse de mensagens entre demais entidades. A *Knowledge Database* é responsável pela persistência do modelo ontológico completo e dos dados contextuais, sendo a visão global dos contextos da aplicação. Por último, para cada tarefa de aprendizado em execução em

dado momento, há um *FL Coordinator* instanciado, responsável por coordenar a ação de cada uma das entidades nas outras camadas.

Para a camada da névoa, sua definição varia na literatura, mas consideramos neste trabalho uma noção geral de que consiste em recursos computacionais distribuídos em localizações geograficamente mais próximas dos usuários [Yousefpour et al. 2019]. Nessa camada situa-se então os *Fog Brokers* e os *Federated Learning Aggregators*. Em cada nó da névoa, há um *FL Aggregator* para cada tarefa de aprendizado em execução, responsável por fazer as agregações das atualizações de modelo, enquanto o *Fog Broker* do respectivo nó serve para direcionar mensagens vindas das outras camadas para o *FL Aggregator* correspondente.

Na camada da borda, há os *Edge Clients* e os *Federated Learning Clients*. O *Edge Client* de cada nó provê uma interface para a aplicação IoT ser capaz de coletar contextos, enquanto fica disponível em plano de fundo para iniciar tarefas de aprendizado, instanciando um *FL Client* quando solicitado por um *FL Coordinator*. Cada *FL Client* é responsável então por treinar modelos locais e avaliá-los.

Tanto na camada da borda quanto da névoa, há também em cada nó uma *Local Knowledge Database*, que contém apenas os contextos coletados pelos próprio nó, ou pelos nós conectados. Além disso, cada *Edge Client* e *Fog Broker*, ao inicializarem, devem se apresentar para o *Cloud Broker* informando quais contextos são capazes de coletar, além de outras informações descritas adiante.

3.2. Protocolos de comunicação

Quanto aos protocolos usados para habilitar a comunicação entre essas entidades, foi empregada uma combinação do Constrained Application Protocol (CoAP) [Shelby et al. 2014] com o formato Concise Binary Object Representation (CBOR) [Bormann and Hoffman 2020] para a representação externa de dados. Essa escolha foi motivada pelo cenário de IoT em que o middleware opera, considerando que são tecnologias semelhantes a outras mais comuns na Web em questão de utilização, porém com menos bytes usados em cada mensagem – o CoAP em relação ao HTTP, e o CBOR em relação ao JSON.

3.3. Modelo ontológico

Dentre as opções de modelagem de contexto, a abordagem de ontologia foi escolhida pela sua expressividade e extensibilidade. Seu objetivo é prover ao middleware e à aplicação que o utiliza uma formalização comum dos dados e metadados persistidos, de forma que o middleware possa se integrar e entender o modelo específico da aplicação, ao mesmo tempo em que organiza suas próprias funcionalidades. A Figura 2 ilustra as principais definições da modelagem interna do middleware.

No centro do modelo está o `mcl:Context`, para representar as classes de contexto. Cada classe de contexto deve ser anotada com as definições de `mcl:WithAttribute` para que o middleware conheça quais propriedades estarão presentes numa instância de contexto (`mcl:onProperty`), e se é uma propriedade chave, isto é, que identifica unicamente aquela instância (`mcl:isKey`). Ainda em relação às classes de contexto, há também a propriedade `mcl:visibility`, cujo valor pode ser `mcl:Public` ou `mcl:Private`. Uma classe de contexto pública indica que suas

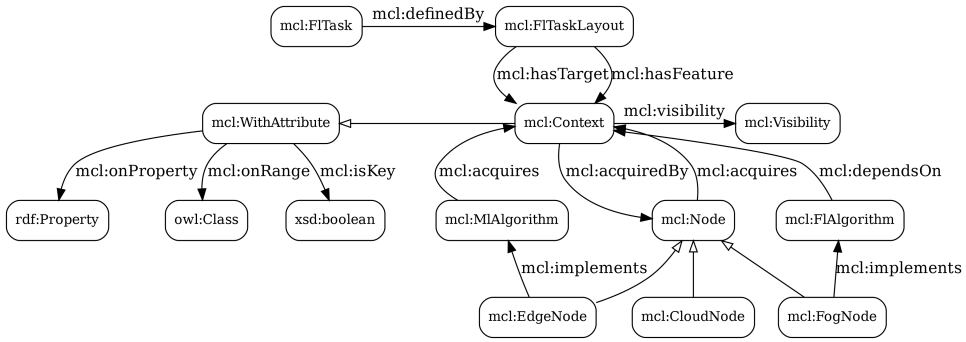
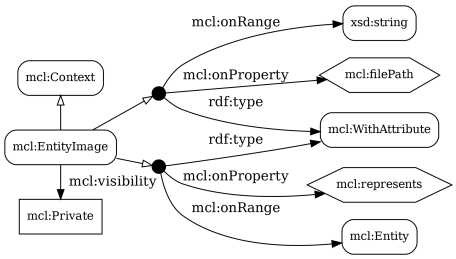
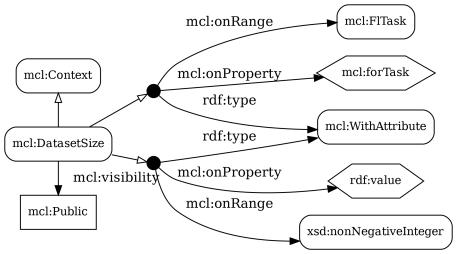


Figura 2. Visão geral do modelo ontológico do middleware.



(a) Definição da classe de imagem capturada sobre uma entidade.



(b) Definição da classe de tamanho de conjunto de dados.

Figura 3. Exemplos de definição de atributos para duas classes de contexto.

instâncias serão distribuídas do nó coletor até a nuvem, enquanto instâncias de uma classe de contexto privada ficam armazenadas apenas no próprio nó que as coletam, sendo usadas apenas para o treinamento local de modelos de aprendizado.

A Figura 3 traz exemplos dessas definições para classes de contexto. Na Figura 3a, há a classe `mcl:EntityImage`, que registra imagens capturadas para representar alguma entidade, sendo assim uma classe de contexto privada. Seus atributos são `mcl:filePath` para indicar onde o arquivo da imagem está salvo, e `mcl:represents` para indicar qual entidade está presente na imagem coletada. Já na Figura 3b, temos a classe de contexto pública `mcl:DatasetSize`, que representa o tamanho total do conjunto de dados de um cliente – registrado no atributo `rdf:value` – para uma tarefa de aprendizado específica – indicada em `mcl:forTask`.

Sempre que um contexto é coletado, o middleware registra quem o coletou (`mcl:acquiredBy`) e quando (`mcl:acquiredAt`). A entidade responsável pela coleta é uma instância de algum tipo de nó, a depender da camada onde se situa (`mcl:EdgeNode`, `mcl:FogNode` ou `mcl:CloudNode`). Como descrito na Seção 3.1, quando um nó se apresenta para o middleware, o mesmo informa quais classes de contexto é capaz de coletar, o que fica registrado pela propriedade `mcl:acquires`. As demais definições, relacionadas a AF, são abordadas na Seção 3.4.

Para acessar efetivamente a *Knowledge Database*, o middleware utiliza uma mesma interface em todas as camadas. Essa interface depende da linguagem SPARQL, e seus métodos são baseados em cada uma das formas de consulta da linguagem: `SELECT`, `CONSTRUCT` e `ASK` para leitura, e `INSERT`/`DELETE` para escrita de dados. A lingua-

gem SPARQL foi escolhida por ser um padrão consolidado, com suporte na maioria dos bancos de dados de grafos, assim como a SQL está para bancos relacionais.

No caso das *Local Knowledge Databases* das camadas da névoa e da borda, a implementação dessa interface usa um banco de dados de grafos embutido, o Oxigraph [Yokochi and Thalhath 2023]. Já na camada da nuvem, é usada por padrão uma implementação baseada no Apache Jena Fuseki, por se tratar de uma solução *open source* com suporte às funcionalidades de inferência da OWL2. A aplicação pode, no entanto, usar implementações diferentes na camada da nuvem.

3.4. Inferência de contexto via aprendizado federado

A principal contribuição do Micelio está em sua capacidade de inferência de contexto, implementada através de AF. O uso do aprendizado de máquina, de forma geral, já era indicado como uma possível estratégia de inferência, adequada para situações em que não se conhece previamente uma relação lógica clara entre os contextos brutos e os contextos a ser inferidos [Perera et al. 2013]. Pensando em melhor atender aplicações IoT cujos objetos inteligentes e usuários estão em grande número e distribuídos por um espaço amplo, como é o caso de aplicações em cidades inteligentes, optou-se pelo AF apoiado pela computação em névoa. Com isso, o treinamento de modelos não depende do envio de todos os dados brutos para um servidor central, apenas das atualizações de pesos dos modelos. Assim, economiza-se recursos de rede e armazenamento, além de preservar a privacidade de parte desses dados.

Para realizar essa implementação, foram analisados diferentes trabalhos na literatura com propostas de algoritmos de AF utilizando a camada da névoa. Foi identificado que parte significativa de suas contribuições tratam de melhorar decisões de uso de recursos do sistema, ou dos dados disponíveis em cada cliente.

A exemplo, o algoritmo apresentado em [Saha et al. 2020] trabalha com múltiplos nós da névoa associados a um subconjunto dos clientes, realizando agregações locais entre esses clientes. Periodicamente, um desses agregadores, ou o próprio coordenador na nuvem, é selecionado para realizar uma agregação global. Há uma decisão envolvida para escolher quais clientes serão servidos por quais agregadores baseada na localização de cada nó, priorizando nós mais próximos, enquanto que a escolha do agregador global se baseia na carga de trabalho de cada nó e no atraso de comunicação entre eles, priorizando nós com menor carga e menor atraso. O mesmo vale para [Tripathy et al. 2024]. O algoritmo de [Hou et al. 2024] segue decisões semelhantes, considerando também o gasto energético de cada nó ao selecionar agregadores, priorizando minimizar esse gasto.

Já em [Fu et al. 2023], é apresentada uma revisão sistemática focada em estratégias de seleção de clientes para treinamento em AF. Ao passo que muitos trabalhos selecionam todos os nós clientes, ou uma fração aleatória deles, nesse estudo são elencados critérios de utilidade dos clientes. Esses critérios podem ser estatísticos, como a quantidade de amostras do conjunto de dados de cada cliente, o valor agregado da função de perda ou erro sob cada amostra, ou a diferença média entre pesos do modelo entre a iteração corrente e a anterior. Há também critérios do sistema, como o atraso de comunicação, a capacidade computacional, ou energia disponível.

Todos esses critérios podem ser modelados como contextos, seja contextos sobre os conjuntos de dados, sobre cada atualização de modelo, ou sobre os próprios nós partici-

pantes do processo de aprendizado. Foi também identificada uma sequência de operações comuns à maioria dos trabalhos analisados, cujos objetivos são os mesmos, mudando apenas os critérios contextuais subjacentes. Essas operações, chamadas aqui de estratégias contextuais, são listadas a seguir:

- **Mapeamento de nós:** dada uma lista de clientes compatíveis com o algoritmo, indica quais clientes da borda serão servidos por quais agregadores da névoa.
- **Seleção de clientes para treinamento:** dada a lista de clientes servidos por um agregador, indica quais clientes participarão do treinamento na iteração corrente.
- **Condição de agregação global:** indica se haverá agregação global na iteração corrente.
- **Seleção de agregador global:** dada a lista de agregadores, indica qual deles será o agregador global da iteração corrente, ou se o próprio coordenador na nuvem fará a agregação.
- **Agregação de modelos**: dada a lista de atualizações de modelo obtidas pelos clientes ou agregadas localmente, retorna o novo modelo agregado.
- **Seleção de clientes para avaliação:** dada a lista de clientes servidos por um agregador, indica quais clientes participarão da avaliação do modelo na iteração corrente.
- **Agregação de avaliação:** agrega as métricas de avaliação coletadas pelos clientes.
- **Condição de parada:** indica quando o algoritmo de aprendizado se encerra.

Assim, de forma semelhante a como bibliotecas e frameworks de AF funcionam (ex.: Flower [Beutel et al. 2020]), o Micelio é capaz de abarcar múltiplas implementações de algoritmos de aprendizado, desde que sigam essas estratégias contextuais. Na prática, o middleware conta com duas interfaces que trabalham em conjunto, o *Machine Learning Algorithm* para o treinamento e avaliação de modelos locais, e o *Federated Learning Algorithm* que reúne cada uma das estratégias contextuais. Todos os métodos dessas interfaces têm acesso à *Knowledge Database*, de forma que o *ML Algorithm* pode registrar quaisquer contextos sobre seu conjunto de dados e métricas de avaliação, enquanto o *FL Algorithm* pode consultar esses e outros contextos públicos da aplicação como um todo.

Atualmente, o middleware conta com uma implementação padrão de cada uma dessas interfaces. Para o *FL Algorithm*, há uma implementação baseada em [Saha et al. 2020], com o mapeamento de nós baseado na proximidade, agregações globais periódicas, e seleção do agregador global com base em sua carga de trabalho. Para o *ML Algorithm*, há um classificador de imagens genérico, baseado num modelo pré-treinado do ResNet-18 [He et al. 2016] e disponibilizado pela biblioteca PyTorch [Paszke et al. 2019].

Para colocar esses algoritmos em execução, são necessárias informações complementares, presentes na modelagem ontológica, como mostrado na Figura 2. Primeiramente, um `mcl:FlTaskLayout` é responsável por definir qual classe de contexto é o *target* e dos modelos da ser treinados (`mcl:hasTarget`) e qual classe possui as *features* (`mcl:hasFeature`). As propriedades chave em comum entre as classes identificam cada amostra, enquanto as demais propriedades são de fato usadas como insumo para treinamento para o *ML Algorithm*. Naturalmente, é esperado que a classe *target* seja marcada como pública, e a classe *feature* como privada. A propriedade `mcl:acquires` para os *ML Algorithms* serve indicar quais classes de contexto o algoritmo é capaz de coletar durante sua execução (ex.: tamanho do conjunto de dados de treinamento, métricas

de avaliação). Por outro lado, a propriedade `mcl:dependsOn` para os *FL Algorithms* indica quais classes de contexto são necessárias para executar suas estratégias contextuais.

Todas essas informações são avaliadas antes de qualquer algoritmo ser executado, durante a inicialização de uma tarefa de inferência. Quando a aplicação solicita o início de uma tarefa, são informados qual *FL Task Layout*, qual *FL Algorithm* e qual *ML Algorithm* serão usados, além de parâmetros arbitrários a ser passados para cada algoritmo. Com isso, o middleware é capaz de listar todos os clientes compatíveis com todos os requisitos: coletar os contextos usados como *feature* e *target*, implementar o *ML Algorithm* solicitado, e coletar todos os contextos requisitados pelo *FL Algorithm*. Se a tarefa for iniciada com sucesso, sua execução é registrada como uma instância da classe `mcl:FlTask`, para que a aplicação possa acompanhar seu andamento.

Esses e os demais passos da execução do AF são ilustrados em alto nível pelo diagrama da Figura 4. Cada uma das estratégias contextuais supracitadas podem ser identificadas por chamadas de método ao *FL Algorithm*. Nota-se também que, apesar de haver apenas um *FL Algorithm* no diagrama, na realidade, há uma instância em execução para cada agregador na névoa, e uma para o coordenador na nuvem. De forma semelhante, também há uma instância do *ML Algorithm* em execução em cada cliente.

Ao fim da tarefa, cada cliente tem acesso ao modelo global. Em posse desse modelo, o middleware o armazena e o aciona periodicamente para inferir novos contextos, com base em contextos coletados após o período de treinamento. Por se tratarem de contextos públicos, o middleware depois os envia para a nuvem, permitindo que a aplicação execute novas consultas contando com o resultado da inferência.

4. Estudo de caso em cidades inteligentes

Para validar a implementação do middleware, foi projetado um estudo de caso com base em uma aplicação relacionada a categoria de cidades inteligentes, pensando em um sistema de coleta de lixo. O objetivo da aplicação é ser capaz de detectar o tipo de lixo contido em cada uma de suas lixeiras inteligentes através do reconhecimento de imagem.

Essa aplicação foi implementada no simulador de rede NS-3[1], num ambiente contendo uma quantidade variada de nós computacionais em cada camada do continuum computacional. O ambiente simulado considera um espaço 5 km x 5 km. Cada caso de execução contém 5 nós de usuários da aplicação, N_E nós da borda, representando as lixeiras com algumas imagens de lixo rotuladas e não rotuladas, 5 nós da névoa, e um nó da nuvem. Todos os nós são situados dentro desse espaço de forma estática e aleatória, com exceção do nó da nuvem, que é situado longe do espaço que representaria a cidade.

A topologia de rede da simulação é gerada através da ferramenta BRITE [Medina et al. 2001], que simula uma infraestrutura de roteadores de forma verossímil. Os nós de aplicação são então conectados a essa infraestrutura em redes locais distintas. Os nós da borda e dos usuários são agrupados aleatoriamente em redes Wi-Fi 802.11n, com no máximo 4 nós em cada uma dessas redes. Os nós da névoa e da nuvem são conectados diretamente a um roteador, usando uma conexão de 1 Gbps de taxa de dados porém 100 ms de atraso para a nuvem – simulando uma maior distância –,

[1] `https://www.nsnam.org/`

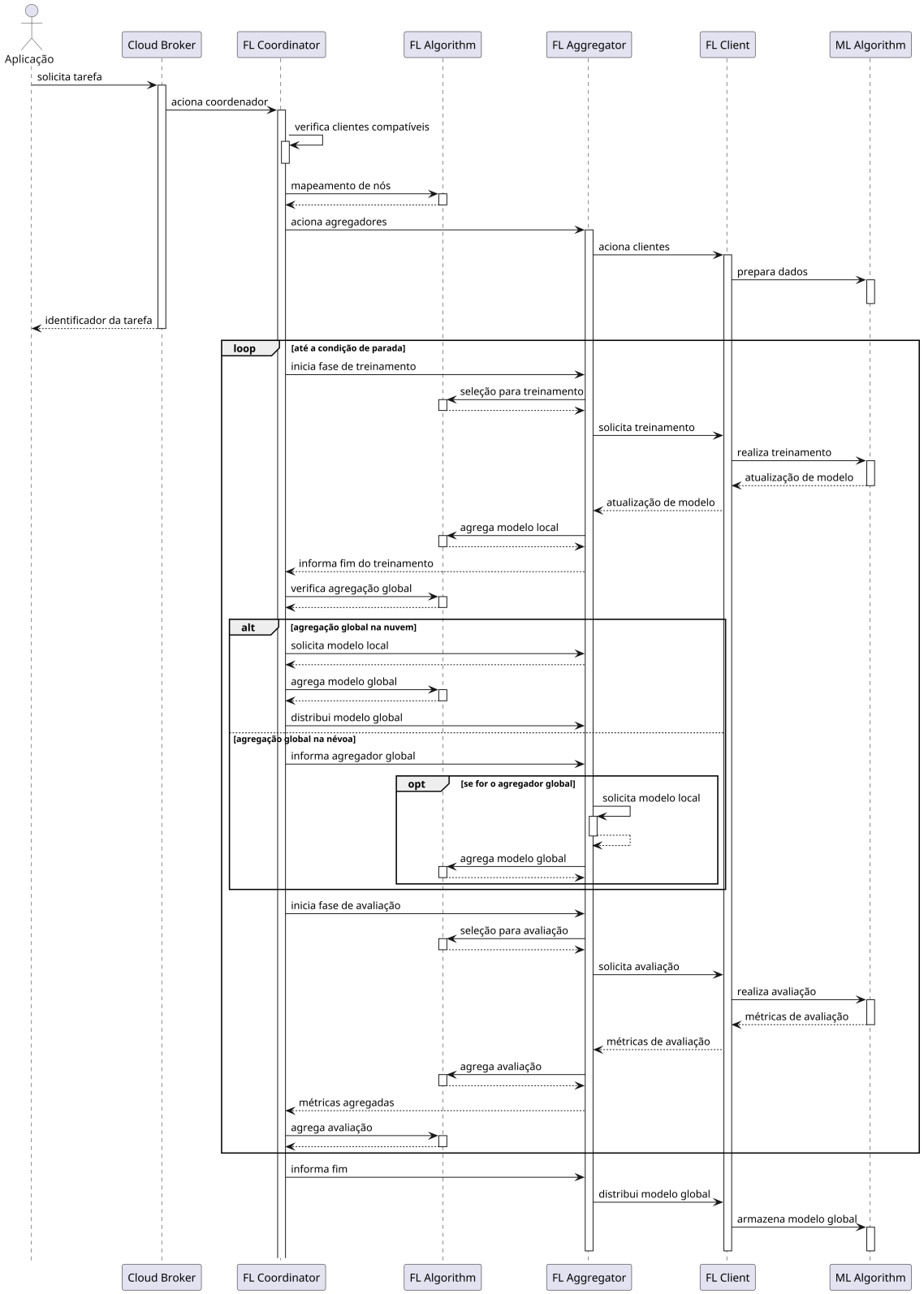
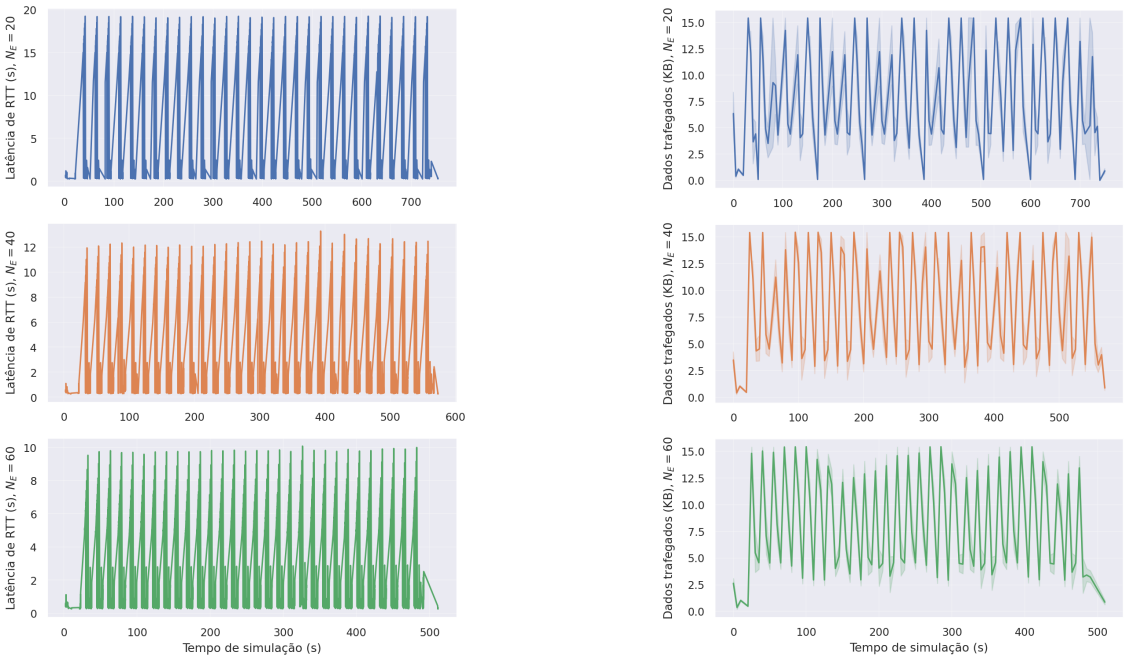


Figura 4. Principais operações e trocas de mensagem durante o aprendizado federado.



(a) Latência de RTT sobre o tempo.

(b) Bytes trafegados sobre o tempo.

Figura 5. Métricas de rede coletadas ao longo do tempo de simulação.

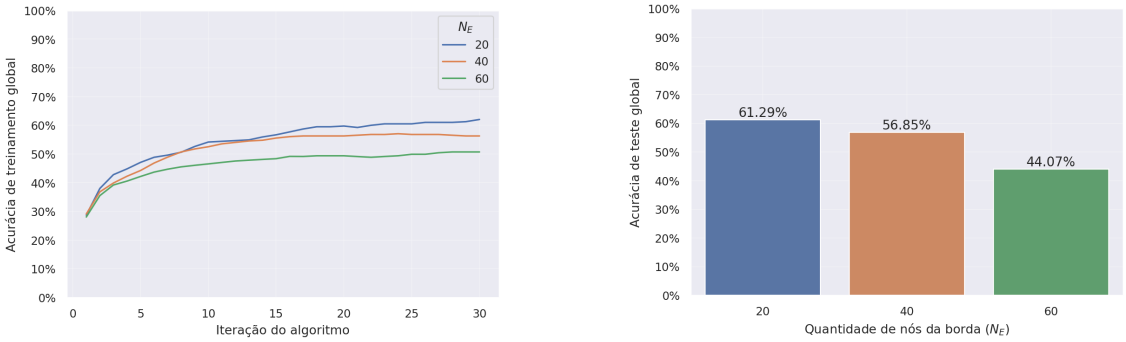
e 100 Mbps de taxa de dados com 5 ns de atraso para a névoa – por estarem posicionados ao lado dos respectivos roteadores.

Sobre a aplicação configurada, foram utilizadas as implementações padrões do *FL Algorithm* e *ML Algorithm*, sendo o *ML Algorithm* configurado para classificar entre 6 categorias de lixo encontradas no conjunto de dados do TrashNet [Yang and Thung 2016]: "papel", "papelão", "plástico", "vidro", "metal" e "outros". Esse mesmo conjunto de dados foi utilizado no treinamento, contendo 2527 imagens de lixo rotuladas. Suas amostras distribuídas aleatoriamente entre os N_E nós da borda. Foi utilizada uma fração de 20% para testes, de forma que o middleware não tem acesso à qual seria a categoria real dessas amostras, e uma fração de 20% durante o treinamento para validação.

Os usuários da aplicação são configurados na simulação para realizar consultas sobre a quantidade total de cada categoria de lixo em uma das lixeiras após a inicialização dos demais nós. Depois das consultas de contexto, um desses nós solicita o treinamento da tarefa de classificação do lixo. A aplicação aguarda o fim do treinamento, e realiza as mesmas consultas do início da simulação, para observar as novas instâncias de lixo que foram classificadas. Toda a implementação do middleware e do estudo de caso encontra-se disponível num repositório online[2].

Com isso, foram coletadas as métricas de latência de RTT (Figura 5a), quantidade de bytes trafegados (Figura 5b), e acurácia de treinamento do modelo de classificação (Figura 6a) ao longo do tempo de simulação. É possível observar como a quantidade de dados contidos em cada nó afetou cada uma dessas métricas.

[2]`https://github.com/NESPEDUFV/micelio`



(a) Acurácia de treinamento dos modelos globais a cada iteração.

(b) Acurácia de teste do modelo global final.

Figura 6. Métricas de acurácia dos modelos.

No início da simulação, quando os *Edge Clients* estão se registrando para o *Cloud Broker*, é o momento em que há o maior fluxo de bytes, pois é quando todos os contextos públicos estão sendo enviados. Apesar disso, a latência nesse momento inicial não sofre grande impacto. Já durante o treinamento dos modelos, cada iteração do algoritmo produz picos bem definidos tanto para a latência quanto para os bytes trafegados. A latência, devido ao tempo necessário para treinamento, e os bytes, devido aos pesos dos modelos. Como se trata de um mesmo tipo de modelo, o tamanho total dos pesos independe da quantidade de dados usados para ajustá-los. Por fim, na acurácia é possível ver como os dados mais espalhados dificultam na convergência dos modelos. Quanto mais nós foram introduzidos, mais a acurácia se reduziu ao final das 30 iterações de cada execução: enquanto em $N_E = 20$ a acurácia de treinamento chegou a 62%, com $N_E = 60$ o valor final foi de 50,7%. Como mostra a Figura 6b, a acurácia de teste atingiu valores semelhantes aos seus respectivos casos de teste, com 61,2% no melhor caso.

Esses resultados sugerem dois pontos principais. Em termos de uso da rede, o middleware foi capaz de oferecer suas funcionalidades sem causar sobrecargas. E em termos do aprendizado, apesar da acurácia não ter atingido níveis satisfatórios para uma aplicação real, no trabalho original no qual o conjunto de dados foi apresentado, já havia sido identificada a dificuldade em encontrar hiperparâmetros adequados para obter bons resultados nesse tipo de algoritmo envolvendo redes neurais [Yang and Thung 2016]. Contudo, a acurácia de teste obtida superou o valor encontrado no trabalho de referência, que variou de 12% a 37% a depender da categoria de lixo, o que demonstra potencial de melhorias.

Assim, o uso do middleware ainda se mantém válido para facilitar a validação de diferentes hiperparâmetros, ou mesmo permitir outras implementações de algoritmo, mantendo abstraídos os detalhes de comunicação do AF no continuum computacional.

5. Conclusão e trabalhos futuros

Neste trabalho, foi apresentado o projeto e implementação do Micelio, um middleware de inferência de contexto através de AF para IoT utilizando o continuum computacional. Sua arquitetura é genérica para diferentes aplicações, sendo capaz de se integrar com diferentes domínios de contexto através de ontologia, além de permitir a extensão de suas funcionalidades através de novas implementações de suas abstrações chave. A arquitetura de AF é compatível com três camadas hierárquicas de computação – nuvem,

névoa e borda –, sendo possível representar diferentes algoritmos propostos na literatura. As implementações foram validadas num estudo de caso para uma aplicação de cidades inteligentes, envolvendo classificação de lixo por imagem. Os resultados obtidos por simulação sugerem um uso adequado dos recursos de rede, com potencial para melhorias dos modelos treinados. Trabalhos futuros incluem a validação de novas aplicações, além da implementação de outras opções de algoritmos para compor o catálogo padrão do middleware.

Agradecimentos

Os autores agradecem o apoio da FAPEMIG (Projeto APQ-03126-18), CNPq (Processo 404922/2023-6) e Manna (Manna Team, Fundação Araucária, Softex e CNPq).

Referências

Al-Dulaimy, A., Jansen, M., Johansson, B., Trivedi, A., Iosup, A., Ashjaei, M., Galletta, A., Kimovski, D., Prodan, R., Tserpes, K., et al. (2024). The computing continuum: From iot to the cloud. *Internet of Things*, 27:101272.

Baccour, E., Mhaisen, N., Abdellatif, A. A., Erbad, A., Mohamed, A., Hamdi, M., and Guizani, M. (2022). Pervasive ai for iot applications: A survey on resource-efficient distributed artificial intelligence. *IEEE Communications Surveys & Tutorials*, 24(4):2366–2418.

Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Kwing, H. L., Parcollet, T., Gusmão, P. P. d., and Lane, N. D. (2020). Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.

Bormann, C. and Hoffman, P. (2020). Rfc 8949 concise binary object representation (cbor). *Internet Engineering Task Force (IETF)*.

Elkhodr, M., Khan, S., and Gide, E. (2024). A novel semantic iot middleware for secure data management: Blockchain and ai-driven context awareness. *Future Internet*, 16(1).

Fu, L., Zhang, H., Gao, G., Zhang, M., and Liu, X. (2023). Client selection in federated learning: Principles, challenges, and opportunities. *IEEE Internet of Things Journal*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hou, W., Wen, H., Zhang, N., Lei, W., Lin, H., Han, Z., and Liu, Q. (2024). Adaptive training and aggregation for federated learning in multi-tier computing networks. *IEEE Transactions on Mobile Computing*, 23(5):4376 – 4388.

Mahieu, C., Ongenae, F., De Backere, F., Bonte, P., De Turck, F., and Simoens, P. (2019). Semantics-based platform for context-aware and personalized robot interaction in the internet of robotic things. *Journal of Systems and Software*, 149:138 – 157.

Medina, A., Lakhina, A., Matta, I., and Byers, J. (2001). Brite: An approach to universal topology generation. In *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 346–353. IEEE.

Michalakis, K., Christodoulou, Y., Caridakis, G., Voutos, Y., and Mylonas, P. (2021). A context-aware middleware for context modeling and reasoning: A case-study in smart cultural spaces. *Applied Sciences*, 11(13).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2013). Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1):414–454.

Razzaque, M. A., Milojevic-Jevric, M., Palade, A., and Clarke, S. (2015). Middleware for internet of things: a survey. *IEEE Internet of things journal*, 3(1):70–95.

Saha, R., Misra, S., and Deb, P. K. (2020). Fogfl: Fog-assisted federated learning for resource-constrained iot devices. *IEEE Internet of Things Journal*, 8(10):8456–8463.

Shelby, Z., Hartke, K., and Bormann, C. (2014). Rfc 7252: The constrained application protocol (coap).

Symeonaki, E., Arvanitis, K., and Piromalis, D. (2020). A context-aware middleware cloud approach for integrating precision farming facilities into the iot toward agriculture 4.0. *Applied Sciences (Switzerland)*, 10(3).

Tripathy, S. S., Bebortta, S., Chowdhary, C. L., Mukherjee, T., Kim, S., Shafi, J., and Ijaz, M. F. (2024). Fedhealthfog: A federated learning-enabled approach towards healthcare analytics over fog computing platform. *Heliyon*, 10(5).

Yang, M. and Thung, G. (2016). Classification of trash for recyclability status. *CS229 project report*, 2016(1):3.

Yokochi, M. and Thalhath, N. (2023). Evaluating oxigraph server as a triple store for small and medium-sized datasets. *BioHackrXiv doi: https://doi. org/10.37044/osf. io/yru4b*, 29.

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., and Jue, J. P. (2019). All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330.

Zhang, H., Huang, T., Liu, Y., Zhu, S., Gui, G., and Chi, Y. (2017). Senz: A context awareness middleware system used in mobile devices. In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pages 1–7. IEEE.