



AGATA – Arquitetura para Gerenciamento Automático de Tarefas de Aprendizado Federado

Guilherme A. Thomaz¹, Fernando Dias de M. Silva¹, Lucas Airam C. de Souza^{1,2},
Luís Henrique M. K. Costa¹ e Miguel Elias M. Campista¹

¹Universidade Federal do Rio de Janeiro, GTA/DEL-Poli/PEE-COPPE, Brasil

²École Polytechnique, INRIA Saclay, França

{guiaraujo, fernandodias, airam, luish, miguel}@gta.ufrj.br,

Resumo. *O aprendizado federado protege a privacidade ao efetuar o treinamento do modelo nos dispositivos dos usuários. Entretanto, o operador em nuvem não possui acesso aos dispositivos, que estão muitas vezes indisponíveis, para experimentar com diferentes modelos e hiperparâmetros. Este trabalho propõe uma ferramenta que resolve esses problemas ao oferecer funcionalidades para a inicialização, finalização e troca automática de múltiplas tarefas de aprendizado federado. A implementação da AGATA é compatível com ferramentas populares, como o Flower, e se diferencia de outros sistemas por automatizar a troca de hiperparâmetros e códigos.*

1. Introdução

O aprendizado federado (*Federated Learning* – FL) mantém o treinamento de modelos de aprendizado de máquina nos dispositivos dos usuários, enquanto apenas os parâmetros são transmitidos e agregados em um servidor central, garantindo privacidade [McMahan et al. 2017]. No entanto, uma vez que o operador perde o acesso direto às máquinas que executam os códigos, se torna desafiador adaptar de forma dinâmica os modelos e hiperparâmetros. Ademais, alguns clientes podem possuir poucos ou nenhum dado, poder de processamento insuficiente, já estarem ocupados com outras tarefas ou não estarem disponíveis por problemas de conectividade. Em um cenário de desenvolvimento e implantação contínuo de aprendizado federado, é necessário um sistema que automatize em tempo real a disponibilização e a alteração das tarefas para os clientes apropriados [Yang et al. 2023]. Este trabalho propõe a AGATA - Arquitetura para Gerenciamento Automático de Tarefas de Aprendizado Federado - que simplifica o trabalho de cientistas de dados ao contribuir com as seguintes funcionalidades:

- a criação e finalização de múltiplas tarefas de treinamento federado no lado do servidor, utilizando ferramentas comerciais como PyTorch e Flower [Contributors 2025, Beutel et al. 2020]. A ferramenta Flower foi selecionada como base por ser a mais adotada;
- a descoberta de tarefas no lado do cliente, com transferência Over-the-Air (OTA), instalação e inicialização automática de códigos;
- o suporte a critérios de seleção de clientes que mantém a privacidade, podendo incluir recursos disponíveis no dispositivo, nível de contribuição para o aprendizado e qualidade da conexão; e

- a implementação de lógicas de automação para modificar a execução de tarefas. Critérios de seleção e hiperparâmetros de uma tarefa podem ser modificados quando uma condição pré-programada é atingida, sem intervenção de um operador humano. Esta funcionalidade não é explorada por outras ferramentas.

O restante do trabalho está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A Seção 3 detalha a arquitetura da ferramenta e a implementação das suas funcionalidades. A Seção 4 descreve a disponibilização de códigos, documentação, artefatos e experimentos. Seção 5 conclui finalmente o trabalho e apresenta direções para trabalhos futuros.

2. Trabalhos Relacionados

Os desafios para o gerenciamento de tarefas de aprendizado federado vêm sendo investigados por diversos trabalhos na literatura. Moon *et al.* implementam a plataforma FedOps para gerenciar o desenvolvimento contínuo de modelos para um cenário de aprendizado federado [Moon et al. 2024]. Wang *et al.* desenvolvem o FLINT (*Federated Learning Integration*), que auxilia na solução de problemas relacionados à limitação de recursos computacionais e à disponibilidade dos dispositivos [Wang et al. 2023]. Ambas as ferramentas não oferecem nenhum mecanismo para inicializar múltiplas tarefas e transferir os códigos para o cliente. A FLIP (*Federated Learning Interactive Platform*) [Galende et al. 2024] oferece camadas acima do Flower para gerenciar tarefas a partir de contêineres Docker. Entretanto, a proposta assume que o nó central de gerenciamento sempre possui acesso aos recursos dos clientes. Yang *et al.* propõem uma ferramenta chamada FLScalizer para fazer a integração e desenvolvimento contínuos de modelos em um ambiente federado com o Git, para facilitar a incorporação de modificações nos clientes e servidor [Yang et al. 2023]. O trabalho não discute técnicas de automatização de troca de tarefas e de seleção de clientes.

Diferente dos trabalhos anteriores, a AGATA gerencia múltiplas tarefas, automatizando a mudança das tarefas e a transferência dos códigos para os clientes mais apropriados. Ferramentas como o Flower não são suficientes em produção, pois elas assumem que os clientes já possuem acesso aos códigos, e que os critérios de seleção de clientes e hiperparâmetros já foram definidos. Neste sentido, a AGATA torna o aprendizado federado viável em cenários onde há necessidade de testar diferentes tarefas e ajustar parâmetros em tempo real.

3. AGATA – Arquitetura e Funcionalidades

A arquitetura utiliza microsserviços para implementar diferentes funcionalidades, conforme o diagrama da Figura 1.

3.1. Broker e RPC

A comunicação entre microsserviços em um mesmo ambiente é intermediada por um *broker*. Cada microsserviço reserva algumas filas no *broker* por onde recebe mensagens, ao mesmo tempo que se podem publicar em uma fila, que será consumida por outro microsserviço. Essa abordagem, conhecida como Publicador/Assinante (*publish-subscribe*), permite comunicação assíncrona, ou seja, o microsserviço interrompido não perde as mensagens destinadas a ele, uma vez que, ao ser reiniciado, consome as mensagens que ficaram enfileiradas no *broker*, aumentando a tolerância a interrupções dos

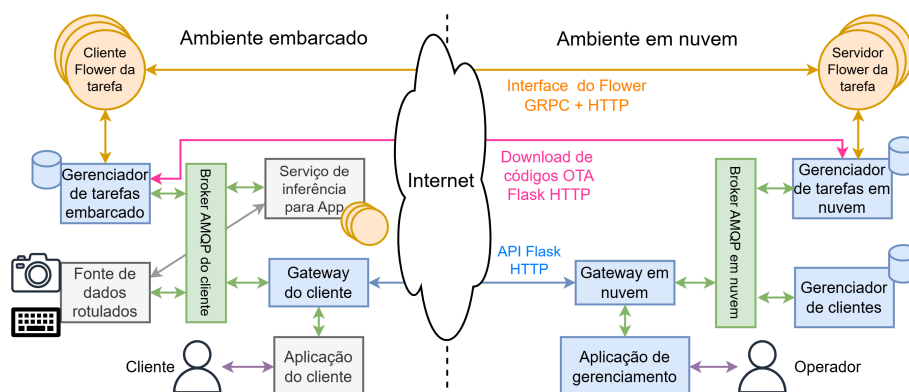


Figura 1. Os retângulos azuis ilustram os microserviços implementados. As setas representam as interfaces de comunicação. O ambiente embarcado é um dispositivo que se comunica com o servidor em nuvem pela Internet

microserviços. A ferramenta escolhida para o *broker* foi o RabbitMQ, o protocolo selecionado foi o AMQP (*Advanced Messaging Queuing Protocol*) e a biblioteca Python para interação com o *broker* é denominada Pika [Pivotal Software, Inc. 2025]. O objetivo da troca de mensagens é habilitar chamadas de procedimentos remotas (*Remote Procedure Calls – RPCs*), ou seja, um microserviço pode invocar a execução de uma função em outro microserviço. Cada fila que um microserviço consome corresponde a uma função que ele executa e as mensagens recebidas nessa fila contêm os argumentos da função. O microserviço chamador cria uma fila no *broker* apenas para receber o retorno. Tanto os argumentos quanto o retorno usam o formato JSON (*Javascript Object Notation*) por sua ampla integração com bibliotecas já existentes.

3.2. Funcionalidades dos Microserviços

A Tabela 1 enumera algumas funções RPC para implementar as funcionalidades de alguns dos microserviços. As funcionalidades de cada microserviço no ambiente embarcado são as seguintes:

1. **Gerenciador de tarefas embarcado:** atualiza o gerenciador de clientes em nuvem com estatísticas, requisita quais tarefas de aprendizado federado o cliente pode participar, baixa os códigos de treinamento correspondentes e inicia/finaliza a participação em tarefas.
2. **Fonte de dados rotulados:** envia os dados rotulados para o gerenciador de tarefas embarcado utilizá-los no treinamento. A transferência OTA dos códigos de treinamento para os clientes em produção facilita o desenvolvimento de novas aplicações que lidam com dados com formatos novos.
3. **Serviço de inferência para app:** mantém códigos de inferência de tarefas que estão prontas para serem utilizadas pelos aplicativos. A arquitetura é agnóstica ao código e provê suporte à implantação de serviços de IA completamente novos, desde que o dispositivo embarcado tenha recursos computacionais suficientes e as fontes de dados adequadas.
4. **Aplicação do cliente:** inclui outros *softwares* que interagem com a arquitetura para usufruir de modelos em execução no serviço de inferência para *app*.

No ambiente em nuvem, os microserviços, com suas respectivas funcionalidades, são:

Tabela 1. Algumas funções RPC importantes nos microsserviços gerenciador de clientes e gerenciador de tarefas em nuvem. Quando omitido, o retorno é uma confirmação de sucesso ou mensagem de erro.

Função	Microsserviço	Argumentos	Retorno
atualiza_info	Gerenc. de clientes	id, qnt_dados, recursos, contribuicao_acuracia	-
lê_info_cliente	Gerenc. de clientes	id	qnt_dados, recursos, contribuicao_acuracia
cria_tarefa	Gerenc. de tarefas	id, ip, porta, cred, arquivos, criterios_selecao, args, tags	-
inicia_tarefa	Gerenc. de tarefas	id	-
pede_tarefa	Gerenc. de tarefas	id_cliente	id, ip, porta, cred, arquivos, criterios_selecao, args, tags
download_cód	Gerenc. de tarefas	id_tarefa, nomes, cred	conteudo_arquivos

1. **Gerenciador de tarefas em nuvem:** consulta informações no gerenciador de clientes, responde aos pedidos de tarefas dos clientes, cria, inicializa e finaliza tarefas a pedido do operador e disponibiliza códigos de tarefas para os clientes baixarem.
2. **Gerenciador de clientes:** registra atualizações com informações e estatísticas de treinamento dos clientes e oferece essas informações para o gerenciador de tarefas em nuvem mediante consulta.
3. **Aplicação de gerenciamento:** consiste em um programa para que o operador envie comandos para criar, consultar, editar, inicializar e finalizar tarefas de treinamento, bem como disponibilizar os modelos resultantes para inferência em aplicações embarcadas. A versão atual da ferramenta oferece interfaces de linha de comando e por *frontend web*.

3.3. Gateways e Interfaces

As mensagens RPC enviadas para o *broker* do cliente que devem ser encaminhadas para microsserviços em nuvem trafegam por *gateways* no cliente e em nuvem, conectados pela interface azul na Figura 1. Os *gateways* também proveem acesso para que as aplicações se comuniquem com o restante da arquitetura. A aplicação de gerenciamento, por exemplo, é um *frontend* que dispara mensagens HTTP para o *gateway* em nuvem executar as chamadas RPC correspondentes no *backend*. O endereço IP do equipamento do usuário pode mudar se a rede móvel implementar DHCP (*Dynamic Host Configuration Protocol*), ou pode nem mesmo ser alcançável, caso a rede utilize NAT (*Network Address Translation*). Assim, adota-se um modelo igual ao utilizado em aplicações *web* com HTTP, nas quais apenas o dispositivo embarcado atua como cliente *web*, enquanto o servidor em nuvem apenas aceita as conexões abertas pelos clientes e encaminha as respostas às requisições. Utiliza-se o arcabouço Flask como servidor HTTP (*Hypertext Transfer Protocol*) em nuvem e a biblioteca Requests, em Python, como cliente HTTP. A interface GRPC (Google RPC), em laranja na Figura 1, não faz parte da proposta da arquitetura, pois é implementada pela biblioteca Flower executada no cliente e no servidor.

3.4. Modelo de Dados do Sistema

O domínio de clientes mantém desde informações básicas, como o identificador (ID) do cliente e os seus recursos computacionais, até estatísticas como a acurácia média,

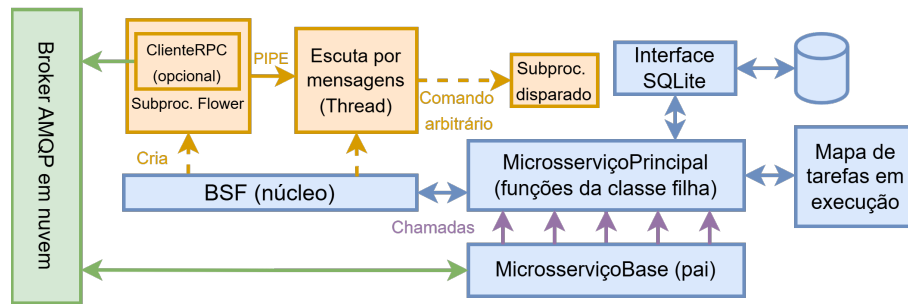


Figura 2. O gerenciador de tarefas em nuvem recebe mensagens RPC pela interface verde e realiza as chamadas de funções correspondentes (em roxo). A classe `ClienteRPC` realiza chamadas em outros microserviços.

a quantidade de dados e o número de desconexões. Já o domínio das tarefas tem o registro de todas as tarefas com seus IDs, se estão em execução ou não, palavras-chave que auxiliam na busca e uma enumeração de todos os nomes de arquivos com os códigos relativos à tarefa. O ambiente em nuvem gerencia as informações a respeito dos clientes e das tarefas disponíveis, permitindo com que os clientes recebam uma lista de tarefas definida pelo servidor com base em suas informações básicas e estatísticas. Para implementação dos bancos de dados (BD) do gerenciador de clientes e do gerenciador de tarefas em nuvem, foi utilizado o SQLite 3 [Hipp 2024].

3.5. Gerenciadores de Tarefas e Biblioteca de Subprocessos do Flower

A Figura 2 ilustra o funcionamento interno do microserviço gerenciador de tarefas em nuvem. O `microserviçoBase` é uma classe que abstrai a comunicação com o *broker*, permitindo que funções RPC sejam implementados em classes filhas. A função `inicia_tarefa`, por exemplo, consulta o BD SQLite usando o ID da tarefa e utiliza a Biblioteca de Subprocessos do Flower (BSF) para iniciar o código do servidor como um processo filho do Linux. Outra *thread* também é criada pela BSF para escutar mensagens recebidas desse servidor Flower por meio de um *pipe* do Linux. A BSF foi desenvolvida para este trabalho uma vez que não foi encontrada uma alternativa pronta para iniciar e gerenciar mensagens de subprocessos Flower. O Código 1 apresenta um trecho de código de um servidor Flower adaptado para se tornar compatível com essas funcionalidades, providas pela BSF. Uma das principais contribuições é que a AGATA não incorre em muita sobrecarga de desenvolvimento a um programador familiarizado com o Flower. O mapa de execução de tarefas é um dicionário que mapeia o ID de uma tarefa em execução em um objeto (*HandlerTarefa*) utilizado para interrompê-la por meio da BSF.

```

1 syspath.append(path.abspath(path.join(path.dirname(__file__), '.')))
2 from bsf.messageiro import Messageiro # Bib. proposta
3 msg = Messageiro()
4 ''' Restante do código Flower entra aqui'''
5 try:
6     start_server(...) # Flower inicia servidor
7     msg.envia_info("Finalizou") # Avisa sucesso pelo pipe
8 except Exception as e:
9     msg.dispara_gatilho("script.py", "arg1") # Opcional
10    msg.envia_erro(e) # Avisa falha pelo pipe
  
```

Código 1. Trecho de código do servidor Flower.

Quanto ao gerenciador de tarefas embarcado, este microserviço não possui funções invocadas por RPC. Ele utiliza uma classe chamada `ClienteRPC` para ativamente chamar funções em outros microserviços por RPC, como `atualiza_info`, `pede_tarefa` e `download_cód`. O cliente também utiliza a BSF e um mapa de tarefas em execução para gerenciar a inicialização e finalização de tarefas recebidas do servidor.

3.6. Diagrama de Sequência

Para ilustrar o funcionamento da AGATA, a Figura 3 apresenta um diagrama de sequência do gerenciamento de uma tarefa federada usando os microserviços da arquitetura. O símbolo de banco de dados é utilizado toda vez que uma função invocada manipula o banco de dados. As seguintes cores foram utilizadas para as mensagens: verde para mensagens entre microserviços no mesmo ambiente trafegando através do *broker*, azul para mensagens entre ambientes diferentes que passam pelos *gateways* além dos *brokers*, roxo para mensagens entre o usuário e os microserviços e rosa para mensagens na interface HTTP de *download*. Há também uma ilustração de uma das telas da aplicação de gerenciamento. O *gateway* em nuvem recebe mensagens HTTP da aplicação e chama as funções por RPC nos microserviços adequados.

Os números na Figura 3 correspondem aos passos descritos a seguir. O operador decide criar uma nova tarefa, de modo a preencher os campos do banco de dados de tarefas (1). Em seguida, o operador inicia uma tarefa previamente registrada informando o ID. Isso dispara um servidor Flower com IP, porta e argumentos especificados previamente no banco de dados, utilizando a BSF. A tarefa é atualizada para o estado ativo no banco de dados (2). Concorrentemente aos passos (1) e (2), o cliente periodicamente envia atualizações, caso existam, de suas estatísticas locais, armazenadas no banco de dados de clientes (3). Em seguida, ele pergunta pelas tarefas disponíveis compatíveis com suas características (4). O gerenciador de tarefas em nuvem consulta as informações dos clientes e os critérios de seleção das tarefas ativas. Para cada tarefa, ele verifica se o cliente é compatível e, em caso afirmativo, coloca a tarefa em uma lista a ser retornada (5). O cliente decide ou não executar as tarefas, sendo a política de decisão considerada nesse trabalho apelidada de *no máximo uma*, na qual o cliente executa a primeira tarefa da lista desde que não haja outra sendo executada (6). Caso deseje ingressar em uma tarefa, ele utiliza as informações recebidas do servidor no passo anterior para baixar os códigos da tarefa e, em seguida, o BSF inicia a tarefa Flower (7).

3.7. Gatilhos para Automação

A BSF dispara a execução de subprocessos arbitrários mediante o recebimento de uma mensagem do código Flower em execução no servidor ou no cliente, conforme Figura 2. Esses gatilhos embutidos no código da tarefa permitem tomadas de decisão automatizadas para mudar adaptativamente os hiperparâmetros ou clientes selecionados, ou mesmo finalizar a tarefa atual e iniciar uma nova no lugar. Um exemplo de disparo de gatilho é apresentado na linha 9 do Código 1. Com esses mecanismos, a ferramenta também serve como uma plataforma para implementar laços de controle nos quais as tarefas são dinamicamente modificadas a partir das estatísticas de treinamento dos clientes. Como esta adaptação em tempo real não é encontrada em outras ferramentas, a AGATA simplifica o desenvolvimento de estratégias mais otimizadas de aprendizado federado.

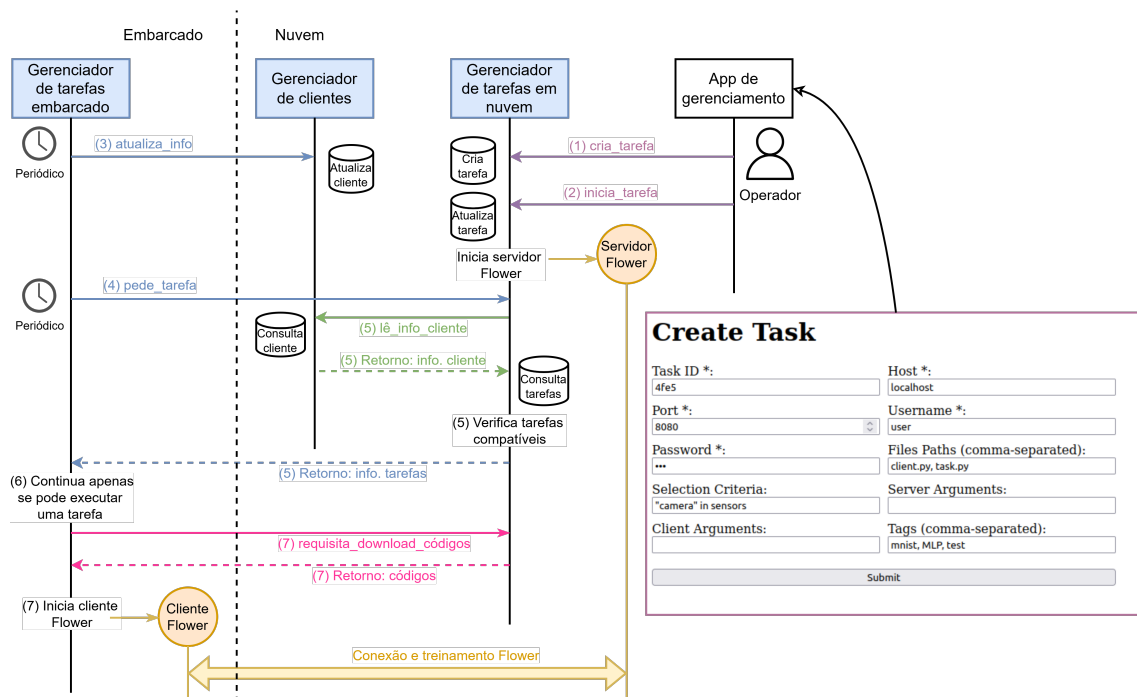


Figura 3. As linhas verticais representam a passagem do tempo de cima pra baixo. Enquanto setas contínuas ilustram mensagens RPC, setas pontilhadas indicam as mensagens de retorno.

4. Disponibilização e Uso

A ferramenta foi disponibilizada em um repositório no GitHub, acessível através do link <https://github.com/GTA-UFRJ/FLMNGR>. O repositório contém um arquivo README.md com os artefatos para reproduzir os experimentos desta seção e as informações sobre máquinas utilizadas, bem como a documentação. Para ilustrar as funcionalidades da ferramenta, foi considerado um caso de uso de treinamento de uma rede neural com o Flower e o PyTorch utilizando dois clientes com a base de dados CIFAR10. Durante o salão de ferramentas, a demonstração será feita em dispositivos portáteis na rede local sem fio das instalações onde ocorrerá o evento, sem haver a necessidade de acesso à Internet. A interação com o gateway em nuvem nos experimentos utiliza códigos executados na linha de comando. Adicionalmente, foi disponibilizada também uma interface em HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*) e JavaScript. Uma das páginas web é apresentada na Figura 3.

O primeiro experimento executa um cenário de uso típico, com: i) registro de uma tarefa pelo operador, ii) inicialização da tarefa, iii) atualização de informações de um cliente, iv) requisição por uma tarefa pelo cliente, v) *download* OTA da tarefa, vi) inicialização da tarefa do lado do cliente, e vii) finalização da tarefa. O experimento termina oferecendo o tempo para execução de cada etapa, em um arquivo de texto. Durante o experimento, um subprocesso que efetua uma escrita em arquivo é disparado quando o servidor alcança determinada acurácia, ilustrando a funcionalidade de gatilhos.

O segundo experimento considera duas versões da tarefa anterior. A versão E apresenta um erro no código do cliente, que ocorre logo após sua inicialização, utilizada para reproduzir uma condição de falha, enquanto a segunda versão (C) executa correta-

mente. O cliente executa a política *no máximo uma* e recebe a lista com ambas as tarefas, com E na frente de C. Quando um erro ocorre no subprocesso que executa a tarefa Flower no cliente, a BSF recebe uma mensagem de erro pelo *pipe*. O tratamento de erro segue com o gerenciador de tarefas do cliente retirando a tarefa do mapa de tarefas e disparando automaticamente um novo contato com o servidor. Sem o uso da AGATA, a troca da tarefa envolveria a intervenção de um operador e o uso de outros *softwares*.

5. Conclusão

Este artigo propõe uma ferramenta para automatizar o gerenciamento de múltiplas tarefas de aprendizado federado nos clientes e no servidor, incluindo inicialização, transferência, modificação e finalização. Com a AGATA, as novas tarefas e as modificações nas tarefas em execução são automaticamente implantadas nos dispositivos dos usuários. Isso viabiliza *pipelines* similares aos de aprendizado centralizado, nos quais cientistas de dados focam apenas na melhoria dos modelos e abstraem a distribuição do aprendizado entre dispositivos. Ademais, a AGATA se difere de ferramentas para a execução de tarefas genéricas em nuvem, pois fornece códigos para os clientes executarem um treinamento dependendo de um critério de seleção. Para avaliar esse critérios, métricas de qualidade de conexão e acurácia serão exploradas em trabalhos futuros. Outra direção de pesquisa promissora é o controle em malha fechada para ajustar parâmetros de treinamento em tempo real, como o número de dados por cliente e o número de clientes.

Referências

- Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K. H., Parcollet, T., de Gusmão, P. P. B. et al. (2020). Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.
- Contributors, P. (2025). Pytorch: An open source machine learning framework. Acessado em: 12 de janeiro de 2025.
- Galende, B. A., Mayoral, S. U., García, F. M. e Lottmann, S. B. (2024). FLIP: A New Approach for Easing the Use of Federated Learning.
- Hipp, D. R. (2024). SQLite: A C library that implements an SQL database engine. Acessado em: 12 de janeiro de 2025.
- McMahan, B., Moore, E., Ramage, D., Hampson, S. e y Arcas, B. A. (2017). Communication-efficient Learning of Deep Networks from Decentralized Data. *Artificial Intelligence and Statistics*, páginas 1273–1282.
- Moon, J., Yang, S. e Lee, K. (2024). FedOps: A Platform of Federated Learning Operations With Heterogeneity Management. *IEEE Access*, 12:4301–4314.
- Pivotal Software, Inc. (2025). *RabbitMQ Documentation*. Acessado em: 12 de janeiro de 2025.
- Wang, E., Chen, B., Chowdhury, M., Kannan, A. e Liang, F. (2023). FLINT: A Platform for Federated Learning Integration. *Proceedings of Machine Learning and Systems*, 5:21–34.
- Yang, S., Moon, J., Kim, J., Lee, K. e Lee, K. (2023). FLScalize: Federated Learning Lifecycle Management Platform. *IEEE Access*, 11:47212–47222.