CESI
ÉCOLE D'INGÉNIEURS

CESI LINEACT

Laboratoire CESI LINEACT (UR 7527)

Mémoire technique

Déploiement et optimisation d'une infrastructure d'apprentissage fédérée

THOMAS Paul

Janvier - Août 2026

Tuteurs d'alternance : Jean-François DOLLINGER, Mohamed El Amine BRAHMIA et Ahmed-Rafik BAAHMED.

Enseignant référent : Hervé BELTZ

Cursus : Première année en cycle ingénieur (FISA A3 Informatique)

Organisme d'accueil : CESI LINEACT (UR 7527), Campus de Strasbourg, France

Année universitaire : 2025/2026

Fiche de confidentialité

Ce document doit être complété pour tout rapport ou mémoire diffusé à CESI École d'Ingénieurs et contenant des informations sur l'entreprise d'accueil.
Ce document est établi en trois exemplaires : un sera conservé par CESI École d'Ingénieur, un autre par l'entreprise, le troisième devra impérativement être intégré au rapport par l'élève.

This page must be included in any document prepared for evaluation purposes at CESI Graduate School of Engineering and that contains information about a host company.
It is produced in three copies: one for the school's records, one for the company, and one which must be included in the student's report.

TITRE DU DOCUMENT EN FRANCAIS	**Mémoire technique**
Title of the document in English	Technical report
NOM DE L'ETUDIANT *Student's Name*	THOMAS Paul
PROGRAMME *Program*	FISA A3 - Informatique
ENTREPRISE D'ACCUEIL *Host company*	CESI LINEACT (UR 7527)
TUTEUR(S) D'ALTERNANCE *In-company supervisor*	Jean-François DOLLINGER, Mohamed El Amine BRAHMIA et Ahmed-Rafik BAAHMED.

☒ ***DIFFUSION LIBRE***	Les rapports / mémoires sont conservés en archives et ils peuvent être librement consultés. Ils peuvent être utilisés par les destinataires, les études peuvent faire l'objet de publication …
Full disclosure	*The documents are kept in the school's archives and can be consulted freely. They can be used by anyone who receives them and the studies they contain can be published…*
☐ ***DIFFUSION RESTREINTE***	Les rapports / mémoires **sont restitués à l'entreprise** à l'issue de la soutenance. Aucune reproduction n'est autorisée. La responsabilité de cette opération est confiée au stagiaire. Les informations communiquées oralement aux membres du jury lors de la soutenance font également l'objet d'une obligation de confidentialité. A ce titre, les membres du jury s'engagent à ne pas divulguer ni utiliser, toute information qu'ils seraient amenés à connaître pendant une durée indéterminée.
Limited disclosure	*The documents **will be returned to the company** at the end of the oral presentation. No copies are allowed. Retrieval of the document is the sole responsibility of the student. The information communicated orally to the members of the assessment board during the oral presentation is also subject to confidentiality. The members of the board therefore commit to not divulging or using any information that may come to their knowledge for an indefinite period of time.*

Dans le cadre de la politique de lutte contre le plagiat, les rapports / mémoires seront susceptibles d'être analysés pour en vérifier les sources et ceci quel que soit le mode de diffusion prévu ci-dessus.

As part of our anti-plagiarism policy, it is likely that reports will be analysed to verify the sources, regardless of the level of confidentiality specified above.

Date:	*Date: 03/07/2026*	*Date:*
L'entreprise / The company	*L'étudiant/ The student*	*L'école / The school*

Résumé / Abstract

Français

Ce mémoire technique présente l'industrialisation et l'automatisation d'une infrastructure d'apprentissage fédérée[1] déployée sur une vingtaine de cartes embarquées[2] hétérogènes, dans le cadre des travaux de recherche du laboratoire CESI LINEACT à Strasbourg. Ce travail fait suite à un premier prototype, développé pendant mon stage précédent, qui validait le principe d'un entraînement fédéré sans être conçu pour un usage en production. L'objectif a été de transformer cette maquette en une plateforme démonstratrice durable, capable de faire fonctionner en conditions réelles des algorithmes jusqu'alors testés uniquement en simulation.

Le travail a porté sur deux axes principaux, à savoir la consolidation de l'infrastructure, avec une intégration automatique des nouveaux appareils en limitant les interventions manuelles, et le développement logiciel, afin que la plateforme prenne en charge plusieurs types d'expérimentations, mais également un système de réservation de machines. Des expérimentations ont permis de valider des résultats concrets sur plusieurs types d'entraînements, notamment un algorithme de sélection[3] des appareils, une méthode d'agrégation intelligente pour accélérer et optimiser l'apprentissage, ainsi qu'un fonctionnement décentralisé direct entre machines, actuellement en cours de finalisation.

La plateforme est aujourd'hui utilisée quotidiennement par les chercheurs du laboratoire, pour leurs expérimentations comme pour la réservation des équipements, et a été présentée à différents publics. Le mode décentralisé[4] est encore en cours de développement et fait l'objet de prochaines mises à jour, aux côtés de l'ouverture du système de réservation aux étudiants et de la potentielle intégration de capteurs réels à la plateforme.

Anglais

This technical report presents the industrialization and automation of a federated learning infrastructure deployed across approximately twenty heterogeneous embedded boards, as part of the research work conducted at the CESI LINEACT laboratory in Strasbourg. This work follows an initial prototype, developed during my previous internship, which validated the proof of concept of federated training without being designed for production use. The objective was to transform this prototype into a sustainable demonstrator platform capable of running algorithms in real-world conditions that had previously only been tested in simulation.

The work focused on two main areas : Infrastructure consolidation, featuring automatic onboarding of new devices to minimize manual interventions, and software development, enabling the platform to support various types of experiments as well as a machine reservation system. Experiments validated concrete results across several training setups, notably client selection, an intelligent agregation method to accelerate learning, and direct decentralized operation between machines, which is currently being finalized.

Today, the platform is used daily by laboratory researchers for both their experiments and equipment reservations, and it has been presented to various audiences. The decentralized mode remains under development and will be included in upcoming updates, alongside opening the reservation system to students and potentially integrating physical sensors into the platform.

Table des matières

1. Introduction

L'apprentissage fédéré permet d'entraîner un modèle d'intelligence artificielle directement sur les appareils qui produisent la donnée, sans avoir à la centraliser sur un serveur unique. Cette approche répond à des besoins concrets, comme la confidentialité des données ou les performances d'entraînement, qui en font un sujet de recherche actif. Cela correspond particulièrement bien à notre cas, pour des applications dans le bâtiment intelligent, où de nombreux capteurs cohabitent et interagissent entre eux. Les différents algorithmes et approches conçues pour optimiser le processus de Federated Learning ont été validés sur simulateur par mes encadrants, mais cela pose tout de même des difficultés complètement différentes dès qu'il s'agit de le faire fonctionner sur du matériel réel, des appareils peuvent tomber en panne, le réseau peut être instable, la puissance de calcul limitée, … C'est l'écart entre ces deux approches, celui de la simulation et celui du déploiement réel, que ce mémoire technique cherche à combler.

Ce travail poursuit le stage de deuxième année ayant abouti à un premier prototype fonctionnel avec quelques Raspberry Pi[5], un protocole de communication[6] développé sur mesure et une configuration manuelle pour chaque carte. Ce prototype avait suffi à obtenir les premiers résultats, mais pas à en faire un outil réellement exploitable. La moindre panne nécessitait une intervention manuelle, l'ajout d'un nouvel appareil demandait une reconfiguration complète, et aucune interface ne permettait à un chercheur d'utiliser la plateforme pour la contrôler et l'exploiter facilement. La mission d'alternance présentée ici part de ce constat pour transformer le prototype en une plateforme robuste, durable et utilisable au quotidien par les chercheurs.

Ce changement d'échelle change complètement la perspective du problème à résoudre. Il ne s'agit plus seulement de faire fonctionner un algorithme sur quelques appareils, mais de garantir que la plateforme avec plus de 20 appareils hétérogènes reste opérationnelle sans intervention constante, qu'une nouvelle carte puisse y être intégré facilement, et que plusieurs utilisateurs puissent réserver et utiliser des machines en parallèle du système FL principal. Le travail que j'ai mené avec l'accompagnement de mes tuteurs, aborde donc deux grandes parties. La première aborde l'infrastructure et le réseau, qui vise à rendre l'ensemble des appareils robustes et autonomes. La seconde aborde plutôt l'aspect logicielle, qui vise à rendre les approches et algorithmes de l'équipe de recherche réellement utilisables et exploitables.

Après une présentation du laboratoire d'accueil, ce mémoire détaillera le sujet et les objectifs de la mission, l'organisation ainsi que le travail mené depuis le début de l'année, puis les résultats obtenus et les perspectives qui en découlent.

2. Présentation du laboratoire CESI LINEACT

2.1. Historique

Le laboratoire CESI LINEACT (Laboratoire d'Innovation Numérique pour les Entreprises et les Apprentissages au service de la Compétitivité des Territoires) constitue aujourd'hui le pilier recherche de CESI. Créé en 2015, le laboratoire a obtenu la labellisation Équipe d'Accueil en 2018, devenant ainsi l'Unité de Recherche UR 7527 reconnue officiellement par le ministère de l'Enseignement Supérieur, de la Recherche et de l'Innovation.

Le laboratoire est une structure de recherche appliquée avec une volonté de créer des liens efficaces entre la recherche scientifique fondamentale, la formation d'ingénieurs et les besoins opérationnels concrets des entreprises et des territoires.

Le laboratoire ne cherche pas uniquement à produire de la connaissance théorique ou à publier des articles scientifiques. Il vise à rendre cette connaissance exploitable, transférable et directement utile aux acteurs économiques et sociaux en rendant les travaux directement accessible et exploitable aux entreprises. Cela est particulièrement pertinent dans un contexte où les chercheurs et ingénieurs doivent être capables de comprendre des problématiques complexes tout en proposant des solutions concrètes et rapidement déployables.

2.2. Axes de recherche

Les travaux et les axes de recherche du laboratoire CESI LINEACT couvrent un champ large : ingénierie numérique, systèmes intelligents, interaction humain-système, industrie du futur, apprentissage et compétences, ainsi que transitions territoriales et environnementales. Ces derniers sont structurés autour de 3 axes :

- Recherche de qualité
- Recherche avec et pour la société
- Recherche appliquée et intégratrice à la frontière des mondes numériques et physiques au service des spécificités territoriales.

Ces 3 axes de recherche de CESI LINEACT se reposent sur 4 domaines applicatifs :

- L'industrie 5.0
- La construction 4.0 et la ville durable
- Les formations du futur
- Les services numériques

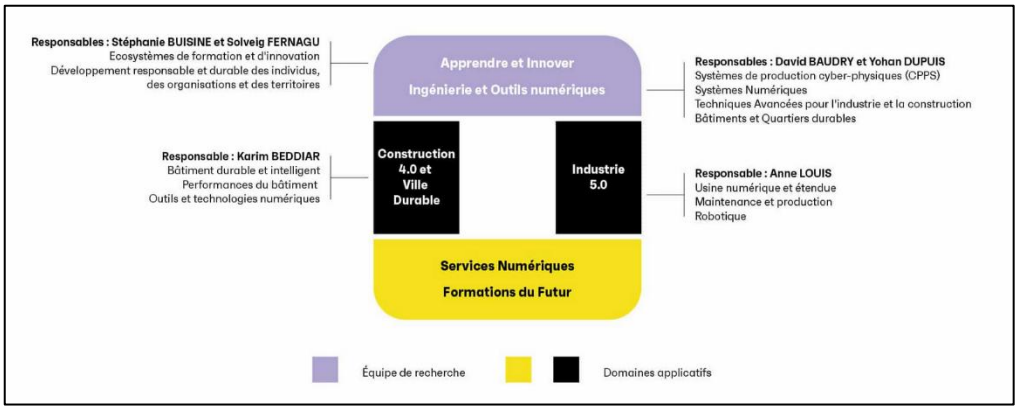

Figure 1 : Axes de recherche et thématiques du laboratoire de recherche CESI LINEACT

Sur le site de Strasbourg où je réalise mon alternance, les thématiques sont basées sur les domaines de l'informatique (services numériques et industrie 5.0) et du BTP (construction 4.0 et ville durable), avec des travaux et recherches souvent liés aux enjeux environnementaux. Ces axes de recherches et domaines applicatifs produisent des projets très concrets, avec des réels résultats.

2.3. Organisation du laboratoire et équipes

Le laboratoire est présent sur plus de 15 campus CESI répartis sur l'ensemble du territoire français. Cette organisation multi-campus est une force structurelle, chaque site développe des travaux et expertises spécialisés avec l'environnement territorial. Il existe également trois plateformes de recherche et de transfert, deux sur le site de Nanterre, ainsi qu'une troisième à Rouen.

CESI LINEACT compte aujourd'hui plus de 200 personnes (février 2026) : 10 directeurs de recherche, 110 enseignants-chercheurs, 2 cadres administratifs et financiers, 16 ingénieurs de recherche, ainsi que 74 doctorants. Le laboratoire fonctionne principalement avec une organisation transversale, composée de deux grandes équipes :

- L'équipe 1 « Apprendre et Innover » relève principalement des Sciences cognitives, Sciences sociales et Sciences de l'éduction avec pour finalité l'étude, la conception et l'évaluation d'écosystèmes d'apprentissage et d'innovation en formation et au travail, dans leurs dimensions sociotechniques, économiques et humaines.
- L'équipe 2, dont je fais partie, « Ingénierie et Outils Numériques » concentre ses principaux domaines scientifiques autour des Sciences du numérique, du Génie des Systèmes Industriels, de la Recherche Opérationnelle et des Sciences de l'Ingénieur.

Mon sujet d'alternance fait partie des projets portés par l'unité de recherche CESI LINEACT. Il répond à des enjeux actuels liés à l'optimisation des performances en intelligence artificielle distribuée. J'ai la chance de faire partie de l'équipe ION (Ingénierie et Outils Numériques) et d'être accompagné trois chercheurs spécialisés en Edge-Cloud, systèmes distribués et IoT : Jean-François Dollinger, Amine Brahmia et Ahmed Baahmed.

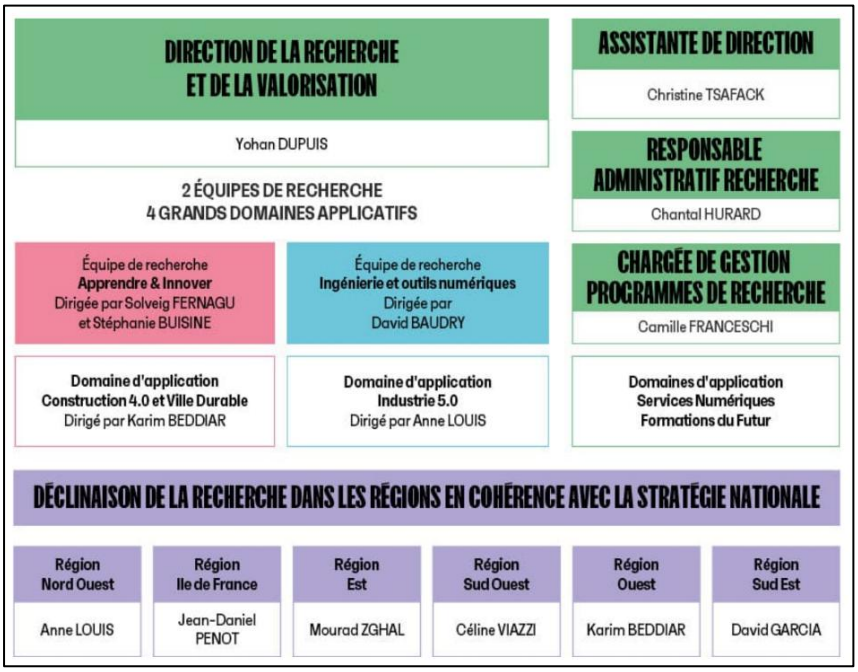

Figure 2 : Organigramme du laboratoire de recherche CESI LINEACT

3. Présentation du sujet et des objectifs

Le sujet de ce mémoire technique s'articule autour d'un déploiement et de l'optimisation d'une infrastructure d'apprentissage fédéré. Cette infrastructure constitue un élément important d'un démonstrateur de bâtiment intelligent plus large, que le laboratoire souhaite créer pour y intégrer progressivement ses différents axes de recherche. L'apprentissage fédéré y tient une place importante dans ce dernier, car il permet d'entraîner des modèles directement sur les appareils qui produisent la donnée (capteurs, cartes de calcul embarquées), sans avoir à centraliser cette donnée sur un serveur, ce qui correspond bien aux contraintes de confidentialité et aux besoins de performances d'entraînement.

Ce sujet a été créé en s'appuyant sur plusieurs objectifs, qui ont guidés les choix techniques tout au long de ce mémoire. L'un des objectifs principaux était de rendre observable, en conditions réelles, des résultats jusqu'ici réalisées au simulateur, sur un seul appareil. Les travaux de recherche du laboratoire ont permis de publier différents algorithmes et approches validés avec des résultats significatifs sur des environnements simulés. La plateforme doit maintenant permettre de refaire ces mêmes expérimentations sur du matériel physique, avec les contraintes réelles de l'apprentissage fédéré, comme la latence réseau, la puissance de calcul limitée et les pannes matérielles, que la simulation ne peut pas reproduire. Au-delà d'une démonstration ponctuelle, la plateforme doit rester utilisable dans la durée, pour, par exemple, accueillir de nouveaux algorithmes sans réécriture complète, intégrer de nouveaux appareils sans configuration manuelle compliquée, et résister aux pannes courantes d'un ensemble de cartes embarquées (coupure réseau, problèmes de mémoire, déconnexions, …). Le dernier objectif était de créer un outil de réservation de machines opérationnel pour les chercheurs du laboratoire. Cette plateforme va, à terme, pouvoir également être présentée et utilisée par des étudiants afin de mettre à disposition une infrastructure matérielle pour les projets liées à la formation de l'école d'ingénieurs CESI.

Sur le plan concret, cela se traduit par deux grands axes de travail, le premier aborde le réseau et l'infrastructure. Il a fallu préparer un ensemble hétérogène de cartes Edge[7] (Raspberry Pi, Jetson[8]) ainsi que de permettre l'intégration de nouveaux clients[9] avec une approche quasiment « plug & play ». Il a fallu repenser la préparation des cartes pour automatiser l'installation du système et le déploiement des services, dont notamment son code source ainsi que ses dépendances. La seconde partie aborde plutôt l'aspect fonctionnel et algorithmique. Il a fallu déployer les différents travaux de recherche et algorithmes du laboratoire, modifier les protocoles de communication, gérer les performances, et créer une interface web dédiée aux chercheurs pour contrôler la plateforme, lancer des expérimentations et en analyser leurs performances. À plus long terme, cette infrastructure doit permettre d'expérimenter de nouvelles stratégies d'expérimentations.

Ces objectifs se résument dans cette problématique, qui structure l'ensemble de ce mémoire :

> **Comment industrialiser, automatiser et améliorer une infrastructure d'apprentissage fédéré constituée d'une vingtaine de cartes embarquées hétérogènes, afin qu'elle prenne en charge plusieurs types d'expérimentations, un système de réservation de machines et qu'elle puisse évoluer, en y intégrant facilement des nouveaux appareils ?**

4. Organisation du travail

Ma mission a démarré début 2026, avec le rythme de l'alternance entre semaines en entreprise et semaines à l'école, ces dernières permettant de prendre du recul sur les choix d'architecture avant de repasser au développement. Dès le départ, deux travaux ont avancé en parallèle, le système d'apprentissage fédéré, et la plateforme de réservation des machines.

Sur le plan algorithmique, la plateforme à évoluée au fur et à mesure, pour prendre en charge aujourd'hui trois types d'expérimentations, développés dans cet ordre :

- Un entraînement classique, mode d'origine reconsolidé, basé sur l'agrégation FedAvg[10] et plusieurs méthodes de sélection de clients (FedCD-CS, FedRP-CS, FedCDRP).
- Un entraînement sur séries temporelles, avec une approche de Split Learning[11], pensé pour des données produites en temps réel (capteurs de bâtiment intelligent) et couplé à un algorithme complexe d'agrégation et d'optimisation AMGF[12].
- Une stratégie décentralisée, SoFeel[13], encore en cours de développement, fonctionnant de façon asynchrone sur une topologie pair-à-pair

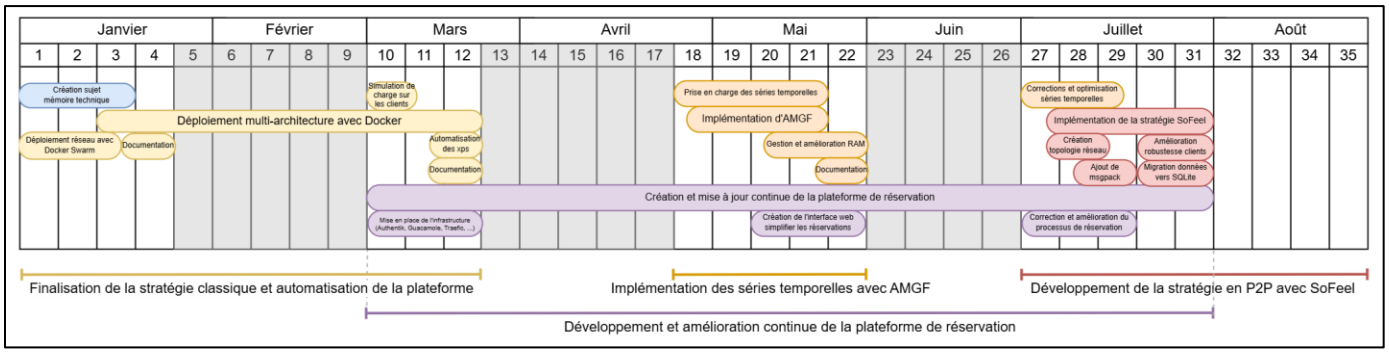

Figure 3 : Planning et organisation du travail sur toute la durée du mémoire technique.

Le déroulement du travail a commencé par la consolidation et l'amélioration du socle d'infrastructure (intégration automatique des appareils, mécanismes de reconnexion, meilleure gestion de la mémoire RAM ...), a continué par la fiabilisation des algorithmes existants pour le mode classique, puis le développement du mode séries temporelles avec le Split Learning et AMGF, et finalement le développement sur SoFeel, qui est encore en cours actuellement. Le développement du système de réservation de machines et des interfaces de suivi s'est poursuivi en parallèle et en continu tout au long de ces étapes.

5. Travail effectué

Dans cette partie, je vais détailler l'ensemble des travaux réalisés au cours du projet. Dans les différentes sous-sections, des explications en détails vont être apportés sur la démarche ainsi que les détails techniques appliqués à la plateforme. Cette section mettra en avant l'architecture conçue, les différents choix techniques réalisés et l'ensemble des travaux qui ont permis de répondre à la problématique de ce mémoire technique.

5.1. Industrialisation et déploiement de l'infrastructure

Cette première partie aborde la consolidation de l'infrastructure et du réseau, dont notamment l'architecture logicielle retenue, l'intégration automatique de nouveaux appareils, et l'automatisation du déploiement du code sur l'ensemble du cluster.

5.1.1. Architecture repensée pour la production

L'architecture du projet est constituée de quatre grandes briques logicielles principales, déployées en conteneurs[14] sur un réseau Docker Swarm[15]. Il y a un serveur d'orchestration qui gère et coordonne les expérimentations, des clients Python[16] installés sur chaque Raspberry Pi ou Jetson, un broker MQTT[17] pour la communication, et un frontend pour pouvoir suivre et contrôler l'ensemble de la plateforme.

Le serveur contrôle le démarrage et l'arrêt de tous les composants. Au lancement, il instancie séquentiellement les différents gestionnaires, comme la communication (qui encapsule les connexions MQTT et WebSocket[18]), le gestionnaire des clients, ou encore les différentes stratégies des expérimentations. Chacune de ces briques sont liées à FastAPI[19], un framework permettant de créer une API REST[20]. Ce dernier expose plusieurs groupes de routes qui permettent de récupérer ou d'envoyer des données au frontend, que ce soit pour le démarrage et l'arrêt d'une expérimentation, les routes pour contrôler le système de réservation de machines, ou encore les routes d'historique qui s'appuient sur la base de données. Chacun de ces groupes sont découpés dans des fichiers uniques permettant de séparer les responsabilités de chacun des fichiers et faciliter l'ajout de nouvelles fonctionnalités sans avoir à modifier les routes existantes à chaque fois.

Le conteneur MQTT, permet la communication via un broker, et est conçu pour des équipements IoT, parfait pour notre ensemble de cartes embarquées. Les communications s'appuient sur des topics[21] structurés sur lesquels les clients vont déposer leurs messages et le serveur va pouvoir les récupérer très naturellement sans créer des connexions avec chacun des clients. Ces derniers utilisent ce canal de communication MQTT pour envoyer par exemple, leur annonce du matériel au démarrage, la taille du dataset local, les métriques de diversité et de performance destinées aux algorithmes de sélection, les poids du modèle mis à jour, ou encore les canaux dédiés aux passes forward[22] et backward[23] du Split Learning, que nous allons voir plus tard dans ce rapport.

Le niveau de qualité de service (QoS[24]) a été configuré au niveau 1, ce qui garantit la remise de chaque message au moins une fois, et permet ainsi d'éviter les problèmes liés aux potentielles coupures du réseau.

En parallèle du canal MQTT, le serveur à une connexion WebSocket dédiée pour les communications en sens inverse, du serveur vers les clients. Un ping est envoyé à intervalle régulier à chaque nœud[25] connecté, pour vérifier leur statut. Si un client ne répond pas dans le délai imparti, la connexion est immédiatement considérée comme perdue, ce qui permet de détecter une déconnexion de manière autonome et indépendante de l'état du broker MQTT.

Le cluster Swarm est organisé autour d'un nœud manager, le serveur, et de nœuds workers correspondant aux Edge. La configuration du cluster déclare chaque service avec des contraintes de placement explicites, le serveur, le frontend et le broker sont contraints au nœud manager, tandis que le service client est déployé en mode « global » sur les nœuds workers, ce qui signifie concrètement qu'une instance est automatiquement lancée sur chaque nouvelle carte qui rejoint le cluster, sans configuration supplémentaire à écrire. Des limites de mémoire RAM[26] ont également été fixées pour chaque service, afin d'éviter des problèmes de saturation mémoire qui pourraient faire tomber l'ensemble du nœud manager si un composant venait à consommer plus que prévu.

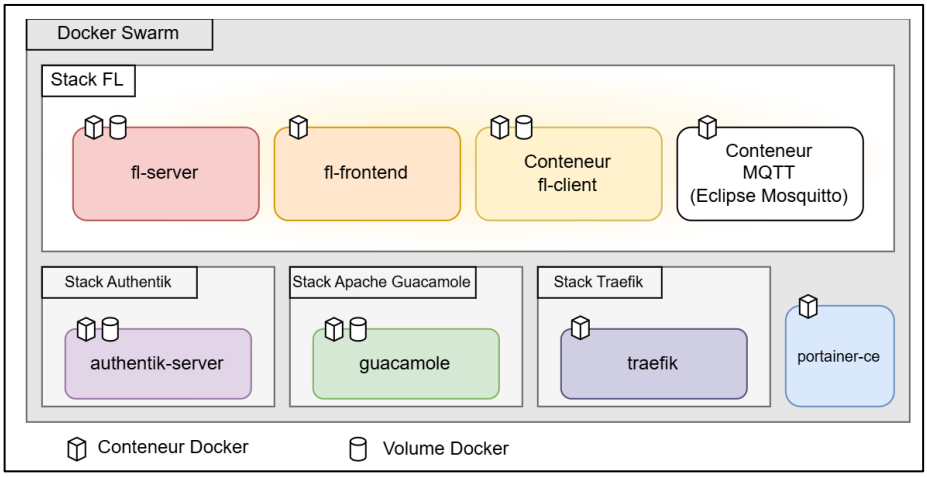

Figure 4 : Différentes briques logicielles du système, séparées et organisées en conteneurs et Stacks Docker distincts

Ces quatre briques ne suffisent cependant pas, à elles seules, à faire de la plateforme un outil réellement utilisable au quotidien par plusieurs chercheurs. J'ai donc ajouté une couche de sécurisation des accès et de supervision déployée sur le nœud manager, autour de quatre services complémentaires.

Le premier, Traefik[27] joue le rôle de reverse proxy [28]pour toutes les requêtes. Il force les requêtes HTTPS, avec le chiffrement TLS[29] et route chaque requête vers le bon service interne. Il s'appuie sur Docker, qui lit directement les labels de chaque service Swarm pour découvrir automatiquement les routes à exposer. Cela permet de simplifier le déploiement d'un nouveau service Swarm, en le définissant simplement directement sur Docker, sans jamais toucher à la configuration de Traefik. Avant de router une requête vers un service protégé, Traefik délègue l'authentification de l'utilisateur à Authentik[30], un fournisseur d'identité OIDC[31] qui est un point

unique d'authentification (SSO) pour l'ensemble des services. Que ce soit le frontend, Guacamole[32] ou Portainer[33], tout passe par le même compte, et cela permet de gérer plus facilement les permissions. Une piste d'amélioration serait d'intégrer directement le SSO du CESI afin d'utiliser le même compte pour l'ensemble des services (ent, moodle ...) ainsi que pour la plateforme FL. Il y a également Apache Guacamole, le cœur du système de réservation de machines. Il permet de rediriger automatiquement l'utilisateur vers une passerelle d'accès distant sans client, qui permet d'avoir un protocole SSH[34] ou VNC[35] directement affichable dans le navigateur web, sans installer rien d'autre. Et enfin, le dernier service supplémentaire, Portainer. Il complète cet ensemble avec une interface de gestion et de supervision du cluster Docker Swarm (état des services, logs, redémarrages manuels), utile pour diagnostiquer un problème d'infrastructure là où le frontend créé reste centré sur les expérimentations ou la réservation de machines.

Ces quatre briques supplémentaires n'interviennent jamais directement dans le cycle d'apprentissage fédéré, mais elles gèrent la façon dont la plateforme est concrètement utilisée et administrée en toute sécurité, ce qui en fait une part à part entière du travail d'industrialisation mené sur cette alternance.

Ce schéma ci-dessous synthétise l'ensemble de cette architecture et la façon dont ses différentes briques interagissent. On y voit la requête filtrée par Traefik puis authentifiée par Authentik, avant d'être routée soit vers le frontend, soit vers Guacamole ou Portainer selon l'usage recherché. La logique principale de la plateforme, à droite, montre le serveur fl-server gérer les communications (avec MQTT et Websocket) avec chaque carte Edge, chacune hébergeant le code Python fl-client ainsi que, potentiellement, plusieurs conteneurs réservés par des utilisateurs.

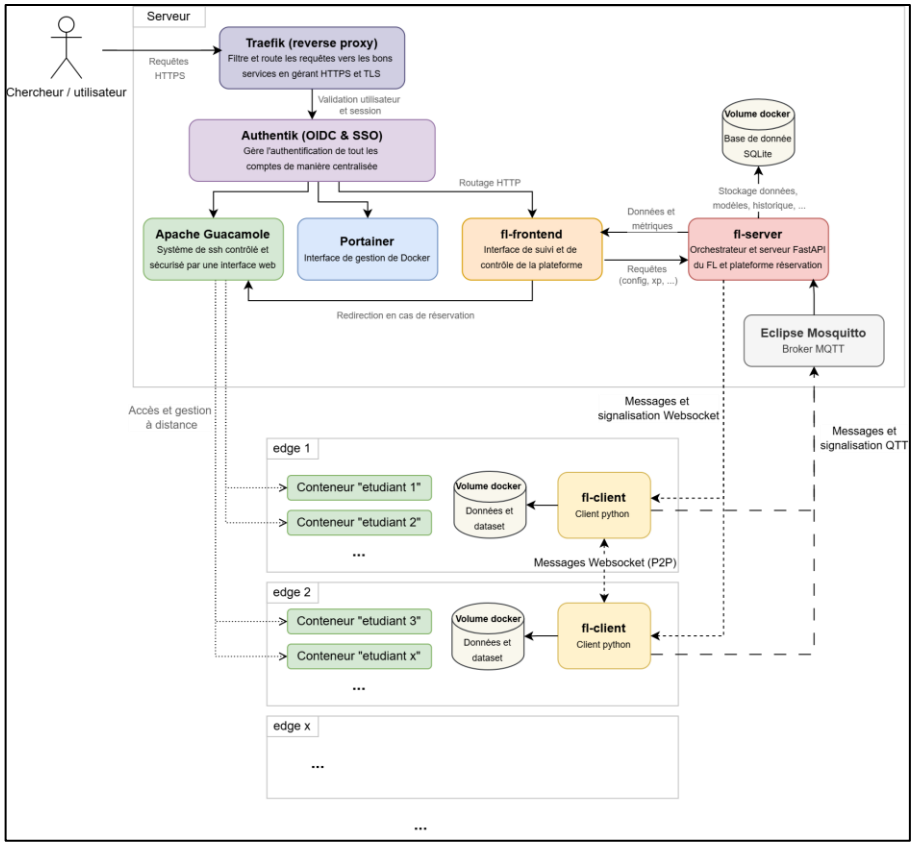

Figure 5 : Architecture logicielle de la plateforme avec son réseau Docker Swarm

5.1.2. Intégration Plug & Play de nouveaux clients

L'un des objectifs du mémoire technique portait sur l'intégration facile de nouveaux appareils sur la plateforme. J'ai réussi à simplifier l'ajout d'un client à la plateforme, avec la simple exécution d'une seule commande sur la carte à ajouter. Docker Swarm, met à disposition un jeton d'authentification, qui une fois transmis à l'un des nœuds, se connectera automatiquement au cluster Swarm et passera en mode « worker ». Il va donc automatiquement récupérer et déployer le conteneur « fl-client », sans aucun script d'installation ni dépendance supplémentaire à préparer manuellement.

Lors du démarrage du conteneur client, les caractéristiques matérielles de la carte sont récupérées à l'aide de différentes librairies Python.

```
architecture = platform.machine()          # "armv7l", "aarch64", "x86_64"
ram_mb = psutil.virtual_memory().total // (1024 * 1024)
cpu_cores = os.cpu_count()
hostname = socket.gethostname()
disk_total_gb, disk_free_gb = shutil.disk_usage("/")
```

Figure 6 : Extrait de code Python permettant de récupérer les informations matérielles des appareils (CPU, RAM, usage du disque, ...)

Ces informations sont empaquetées et publiées sur un topic MQTT dédié, avec le QoS défini au niveau 1. Cela demande au broker de conserver le dernier message publié sur ce topic. Cela est très pratique si le serveur démarre après le client, par exemple à la suite d'une mise à jour, il reçoit immédiatement l'annonce conservée par le broker, sans devoir attendre que le client publie de nouveau spontanément.

En parallèle, le client établit une connexion WebSocket vers le serveur et s'enregistre avec un identifiant unique, actuellement basé sur son hostname[36]. Ce choix évite de s'appuyer sur l'adresse IP. Si plusieurs nœuds sont déployés sur un autre campus CESI, par exemple, tous les appareils situés derrière le même réseau partageront la même IP publique, ce qui rendrait impossible l'identification individuelle de chaque carte (problème similaire lors du routage dynamique au sein d'un réseau Docker Swarm en mode overlay). Cette combinaison, MQTT pour les caractéristiques matérielles et WebSocket pour l'enregistrement logique, donne au serveur et aux clients la possibilité de communiquer dans les deux sens via les deux canaux de communication.

Le serveur maintient une liste des clients connus, mis à jour à chaque réception d'un message d'annonce d'enregistrement. Aucune liste statique de clients attendus n'est créée, tout est dynamique. Le serveur accepte tout client qui s'annonce correctement, ce qui rend possible l'ajout ou le retrait d'une carte « à chaud », sans redémarrage. Les clients apparaissent automatiquement sur l'interface web dès leur enregistrement.

Si un client perd sa connexion réseau, le processus principal encapsule le cycle de vie complet du client dans une boucle qui retente indéfiniment, avec un court délai entre les tentatives. À chaque reconnexion réussie, le client republie son annonce matérielle et renvoie son message d'enregistrement, ce qui le réintègre automatiquement dans la liste du serveur et dans l'ensemble des cartes disponibles pour les prochains rounds[37], sans aucune intervention manuelle.

Après qu'un client se soit déconnecté puis soit revenu en plein milieu d'un round ou d'une update[38], il peut rejoindre l'expérimentation en cours grâce au mécanisme de reconnexion automatique. Le serveur lui retransmet la configuration du round ainsi que le point de reprise exact où il doit recommencer à s'entraîner, le réintégrant dans les clients actifs sans devoir attendre le round ou l'update suivante.

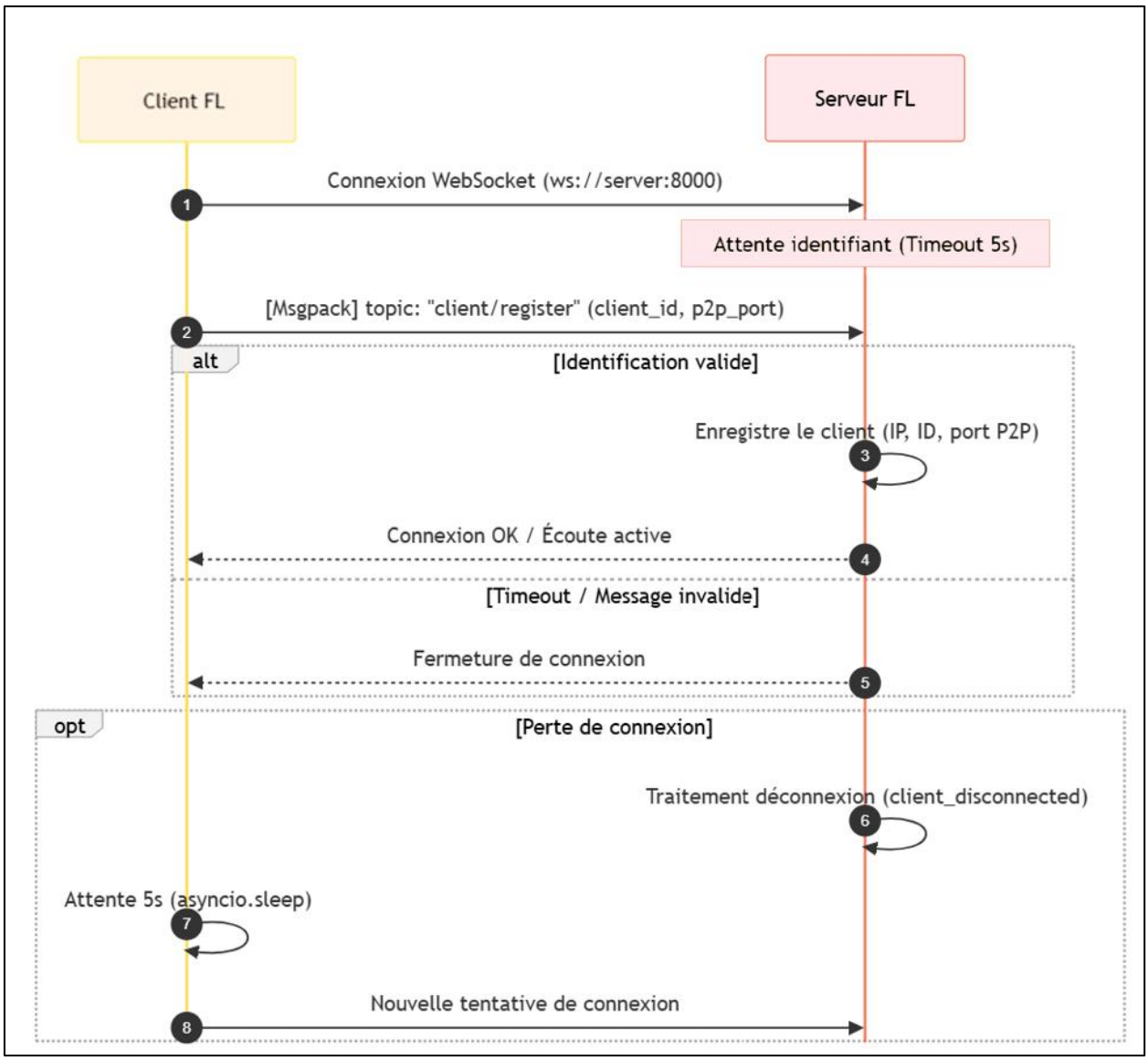

Figure 7 : Diagramme séquence du protocole de connexion et d'enregistrement des clients à chaud

Ce diagramme détaille le déroulement de ce protocole de connexion. On y voit également bien la logique de reconnexion automatique décrite plus tôt, le client ne reste jamais bloqué dans un état d'échec constant.

5.1.3. Automatisation du déploiement des travaux du laboratoire

La mise à jour du code a été automatisée via une pipeline CI/CD[39] déclenchée à chaque commit sur la branche principale (main) du projet. Les images finales sont publiées sur GHCR (Github Container Registry)[40], puis récupérées automatiquement par les nœuds du cluster Swarm lors du déploiement.

Pour optimiser ce processus, la pipeline s'appuie sur deux mécanismes :

- La pipeline vérifie d'abord si les modifications impactent les dépendances lourdes (version de Python, TensorFlow[41]). Si c'est le cas, une image de base est reconstruite avec le cœur du système (Linux Alpine, Python, librairies). Sinon, seule l'image applicative l'est, qui contient uniquement le code de l'application. Tout cela utilise un système de cache Docker, qui réduit le temps de build de plusieurs minutes pour la majorité des commits, tout en évitant le transfert inutile de gros volumes sur le réseau Swarm.
- La plateforme tourne de façon stable sur trois architectures matérielles : ARMv7 (Raspberry Pi 2), ARM64 (Jetson et Raspberry Pi récentes) et x86_64 (serveur). Grâce à l'émulation processeur et au build dynamique, Github Actions génère des images multi-architecture qui intègrent les différentes variantes. Au démarrage, le moteur Docker de chaque carte sélectionne automatiquement la version adaptée à son processeur.

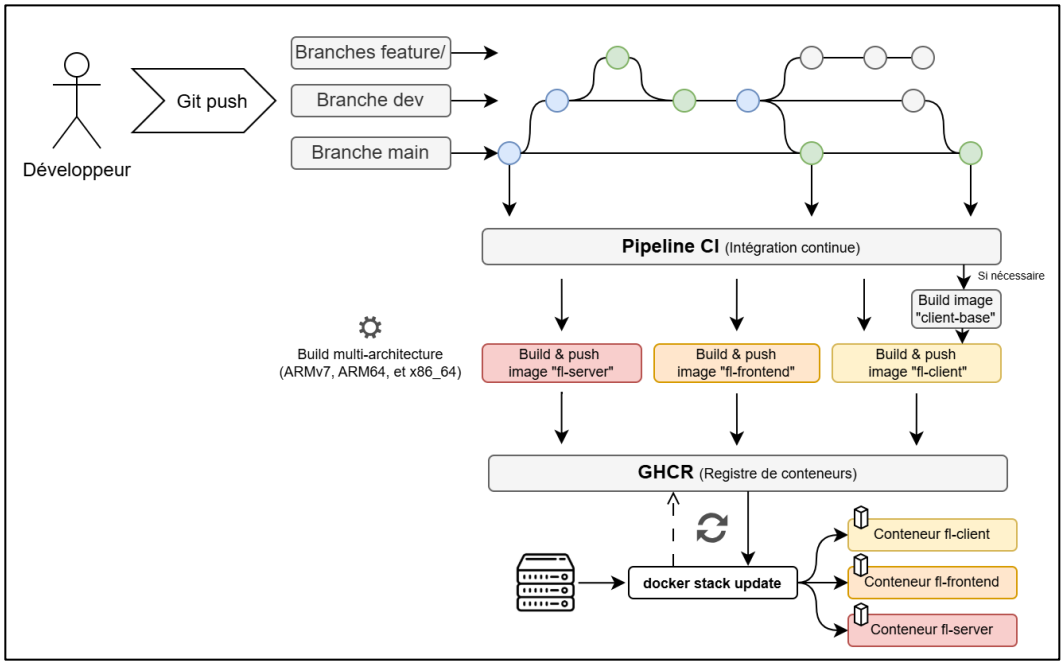

Figure 8 : Schéma représentant l'ensemble de la pipeline CI/CD construire pour la plateforme FL

Pour améliorer l'intégration continue dans le projet et améliorer l'architecture du programme, une approche a été conçue pour accueillir les différents algorithmes de recherche du laboratoire sans modifier le code existant. Le serveur a été réorganisé autour de plusieurs design patterns [42](Strategy, Registry, Factory ...) pour optimiser l'ensemble du projet.

Pour donner un exemple concret, les expérimentations sont définies par une interface commune qui explique ce que doit implémenter chaque mode d'entraînement. Les trois types d'expérimentation héritent de cette dernière, une pour l'entraînement classique avec les sélections de clients (FedCDRP, FedCD-CS, FedRP-CS, …), une pour le Split Learning sur séries temporelles avec AMGF, et une pour le mode pair-à-pair SoFeel. Le serveur choisit la bonne stratégie à partir d'une simple chaîne de caractères transmise dans la configuration de l'expérimentation, et délègue chaque message reçu à la stratégie active sans jamais avoir besoin de connaître le détail du mode d'expérimentation en cours.

Pour la sélection des clients au sein d'un round, dans l'expérimentation classique, le design pattern « Registry » a été appliqué. Celui-ci associe chaque nom de stratégie à sa fonction de sélection, chacune reçoit les informations des clients et le contexte du round, et retourne simplement la liste de clients sélectionnés.

```
SELECTION_METHODS = {
    "FedRandom": fedrandom_client_selection,
    "FedCD-CS": fedcd_client_selection,
    "FedRP-CS": fedrp_client_selection,
    "FedCDRP":  fedcdrp_client_selection,
}
```

Figure 9 : Extrait de code Python, un dictionnaire définissant les méthodes de sélection possibles utilisées par le design pattern Registry.

Ce registre est un simple dictionnaire Python. Ajouter une nouvelle stratégie de sélection revient à écrire une fonction et à l'enregistrer ici, sans toucher au code d'orchestration qui l'appelle. Le même principe s'applique du côté des modèles, avec une « Factory ». Celle-ci instancie le bon modèle à partir de son nom, ce qui permet d'ajouter un nouveau modèle en écrivant une seule classe et en l'enregistrant, sans modifier le reste du système.

Côté serveur et client, toutes les opérations coûteuses en CPU[43] ou en mémoire RAM, sont effectués dans des threads dédiés, ce qui évite de bloquer les boucles d'événements asynchrone. Cela a été mis en place lorsque des opérations critiquent surviennent, que ce soit côté client lors d'un entraînement d'un modèle Keras[44], ou côté serveur lors du calcul de la précision[45] ou des opérations pendant le Split-Learning décrites plus tard dans ce rapport. Ces dernières sont lancées comme des tâches asynchrones indépendantes, ce qui permet aux appareils d'être toujours disponible et d'envoyer ou recevoir des messages pendant des calculs lourds. Cela permet, par exemple, lorsqu'un signal d'annulation d'une expérimentation est reçu du serveur, de pouvoir interrompre proprement l'entraînement en cours plutôt que d'attendre la fin pour réagir à ce message d'arrêt.

J'ai également pu mettre en place une détection au démarrage, permettant de détecter des nouveaux datasets ajoutés à la volée, comme ceux pour les séries temporelles (CU-BEMS, LBNL, LBNL-fixed, N-CMAPSS et USA). Les datasets volumineux, comme MNIST et CIFAR-100 gérés directement via la bibliothèque Tensorflow, et sont téléchargés dans les volumes Docker du serveur et des clients. Côté client, le dataset est uniquement chargé en mémoire lorsque les indices sont transmis par le serveur. Cela permet d'extraire et d'utiliser uniquement la partition locale propre à chaque client. Cette approche permet au serveur de gérer la répartition, des données entre les clients à chaque round.

5.2. Séries temporelles

Après avoir créé l'entraînement classique pendant le stage précédent, qui se base sur un apprentissage commun sur une base de données comme MNIST ou CIFAR, il y a eu une implémentation d'une nouvelle stratégie d'entraînement, qui se base sur des séries temporelles, en lien avec le démonstrateur de bâtiment intelligent. C'est un gros changement d'architecture qui a restructuré les modèles, les données, mais aussi le protocole de signalisation.

5.2.1. Traitement des données

Le démonstrateur a pour objectif dans le long terme de se baser sur des capteurs de bâtiment (température, humidité, …). Les Raspberry Pi n'en sont cependant pas encore équipés, l'ajout de capteurs physiques est seulement une potentielle évolution future pour la suite du projet. En attendant, j'ai intégré une distribution de données créée par Ahmed sur l'infrastructure fédérée. Celle-ci permet de distribuer à chaque client, des données de séries temporelles de réels capteurs, qui permettent de valider dès aujourd'hui l'ensemble de la plateforme réelle. Cela nous permettra de faire basculer la plateforme sur des données réelles dès que l'intégration des capteurs sera prévue, sans changement d'architecture. Nous avons intégré 5 datasets réels de séries temporelles (CU-BEMS, LBNL, LBNL-fixed, N-CMAPSS et USA), axés sur le secteur de l'industrie et la gestion énergétique de bâtiments. L'ajout potentiel de nouveaux jeux de données dans le futur se fait automatiquement par simple ajout de dossiers dans le code source.

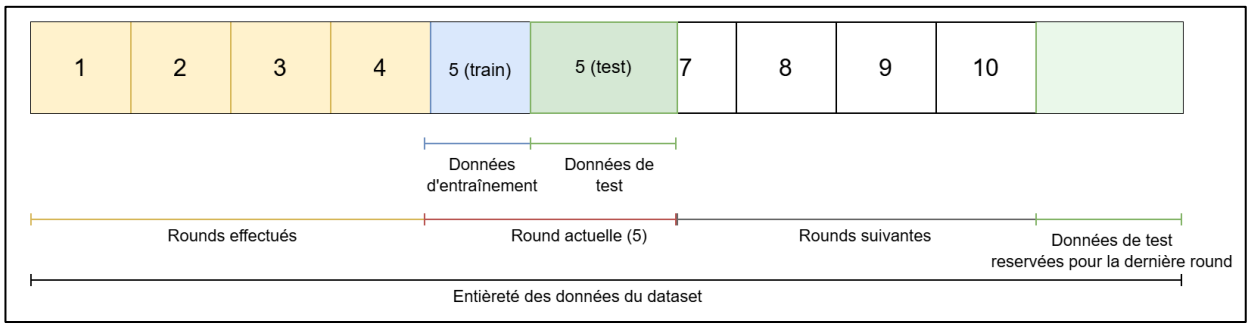

Figure 10 : Représentation d'un dataset de séries temporelles distribué pendant une expérimentation en Split Learning sur l'un des appareils.

Comme l'illustre la figure ci-dessus, le découpage temporel des données pour l'apprentissage fédéré fonctionne avec un principe de fenêtre glissante le long de la série temporelle. L'ensemble du dataset est tout d'abord segmenté en tranches chronologiques correspondant aux différents rounds d'entraînement (ici 10 rounds pour l'exemple). À chaque round, une portion de données est envoyée pour l'entraînement local du client (5 train en bleu), avec des données pour l'évaluation (5 test en vert), qui sont en réalité les données d'entraînement pour les rounds suivantes. Les segments précédents (1 à 4) correspondent aux rounds déjà effectués, tandis que les segments de données suivants (6 à 10) seront balayés progressivement au fur et à mesure que les rounds progressent. Une portion située à la fin du dataset (après le 10ème round) est strictement préservée pour analyser les performances globales du modèle final sur des données qu'il n'a jamais vues. Ce découpage par fenêtre glissante garantit un entraînement réaliste qui respecte strictement la chronologie des données. L'objectif final est d'entraîner le modèle à prédire les données futures à partir des données déjà observées.

5.2.2. Split Learning

Pour les séries temporelles, nous avons décidé de choisir une architecture distribuée et découpée, une méthode d'apprentissage appelée « Split Learning ». Cette approche consiste à scinder le réseau de neurones[46] en deux parties distinctes, réparties entre les clients et le serveur centralisé. Au lieu d'échanger des modèles complets ou d'envoyer des données brutes, les clients ne traitent que la première partie du réseau de neurones et transmet ses résultats intermédiaires au serveur, qui termine les calculs puis renvoie les corrections. Ce découpage permet de préserver la confidentialité des données brutes tout en réduisant drastiquement les coûts de calcul sur les Edge.

L'architecture se décompose en deux parties, une côté serveur et une côté client. En entrée côté client, les données brutes de la série temporelle X_i alimentent localement un réservoir[47] R_i. Ce réservoir n'a pas besoin de rétropropagation, et reste fixe dès l'initialisation. Les représentations que le client i a faites depuis son réservoir alimentent un module Student S_i basé sur un réseau GRU[48], puis dans un Output module O_i qui calcule la prédiction finale et rétro-propage le gradient[49] de la perte $\nabla \mathcal{L}_i$ vers son module Student. Du côté serveur, il va transmettre la sortie du réservoir R_i au module Teacher T, une structure Transformer plus lourde avec des capacités améliorées par rapport à celui du Student. Ce Teacher alimente à son tour le module de sortie local O_i, et le gradient de perte $\nabla \mathcal{L}_i$ lui est renvoyé pour ajuster ses paramètres. C'est grâce à cette double structure que le client apprend, via la distillation[50] de T vers S_i.

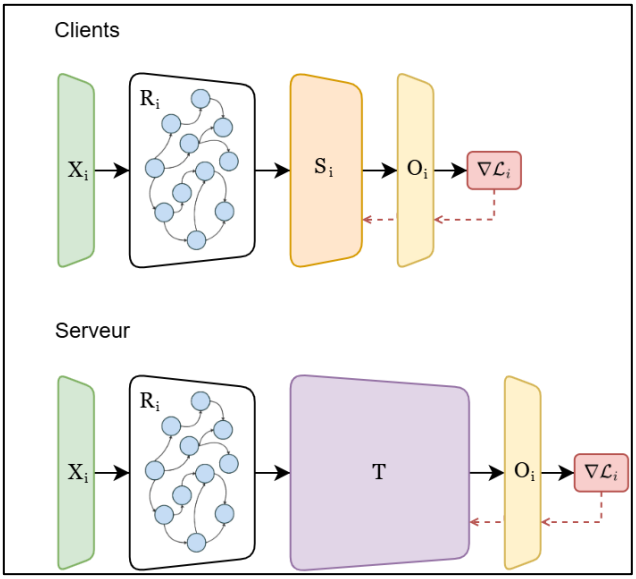

Figure 11 : Décomposition entre le modèle léger des clients (Student) et le modèle plus complet du serveur (Teacher)

L'entraînement s'effectue par update avec une signalisation dédiée au Split Learning. Pour chaque update, le serveur transmet les données prêtes à l'emploi au client, qui les traite via son réservoir et la renvoie au serveur. Le serveur la traite au travers de son module Teacher puis retourne le résultat. Le client évalue alors la fonction de perte, calcule le gradient de l'erreur par rapport à la représentation reçue et le transmet au serveur. Ce dernier ajuste les poids du Teacher et renvoie une représentation corrigée, qui sert de cible pour la mise à jour du module Student local. Lors de chaque update, seules les mises à jour des modules Student et de sortie sont échangées, le réservoir reste inchangé et le Teacher demeure centralisé sur le serveur.

Dans ce mode d'expérimentation, la méthode de reconnexion automatique que j'avais créé permet aux clients déconnectés, de réintégrer immédiatement l'entraînement à l'update actuelle, sans attendre la fin de la round complète.

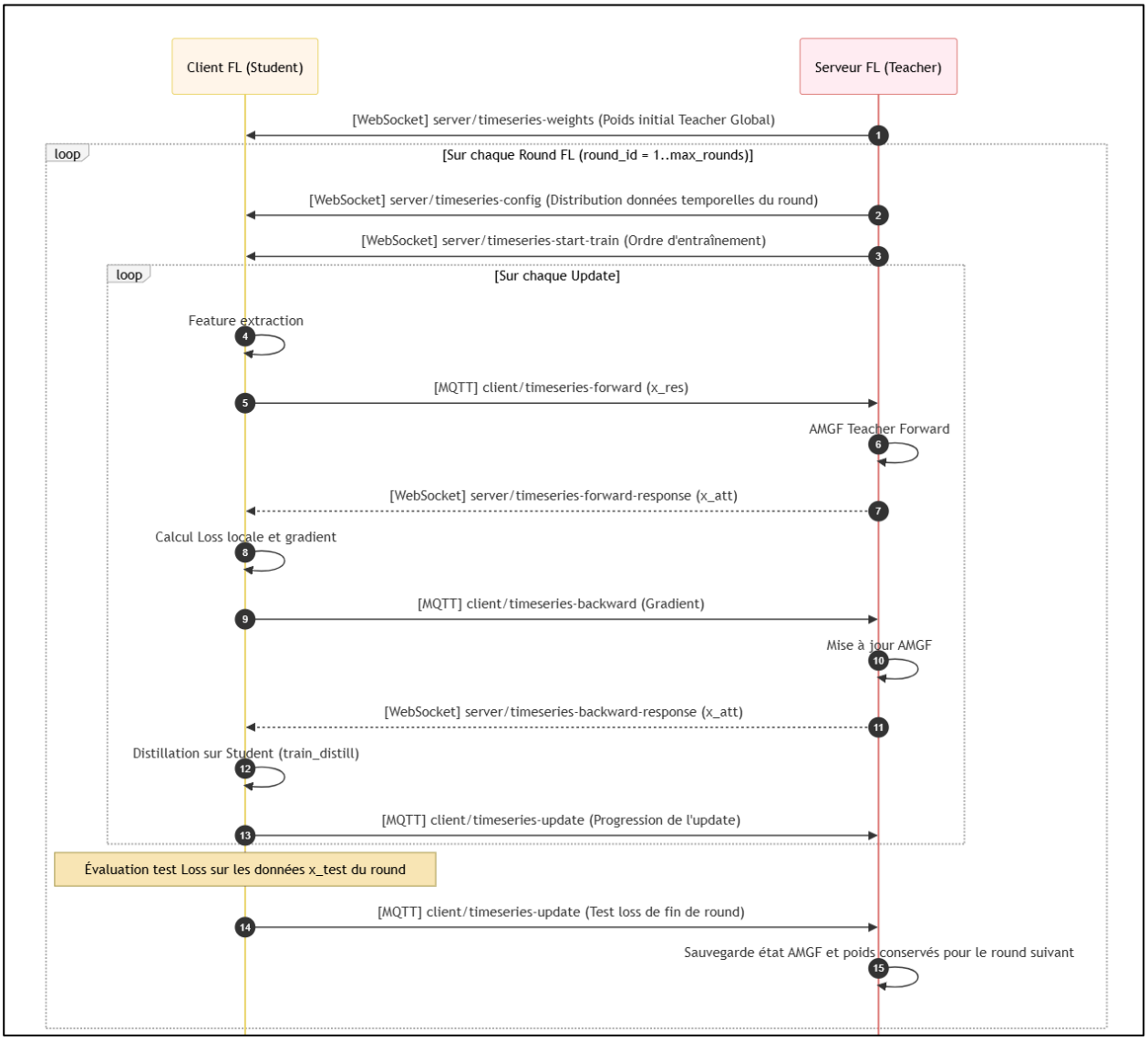

Figure 12 : Diagramme séquence du processus et de la signalisation du Split Learning, avec le forward et backward pass

Le diagramme ci-dessus permet de visualiser et résumer le détail des échanges entre le client et le serveur au sein d'un processus de Split Learning. Chaque round démarre par l'envoi de la configuration et des données du round. Pour chaque update, le client extrait les caractéristiques via son réservoir et transmet cela au serveur en MQTT, qui la traite à travers le module Teacher avant de renvoyer les nouveaux vecteurs de représentation par WebSocket. Le client calcule alors sa loss[51] locale et son gradient, les retransmet au serveur, qui met à jour son AMGF avant de renvoyer ses vecteurs actualisés grâce à la backpropagation, servant à la distillation du module Student. Cette boucle se répète pour chaque update. Une évaluation finale de la loss de test se réalise juste avant la fin de la round, après la dernière update. L'état AMGF ainsi que les poids sont sauvegardés pour le round suivant, et ainsi de suite !

5.2.3. AMGF

L'AMGF est un algorithme exécuté sur le serveur, conçue par mes encadrants dans l'objectif de stabiliser et d'optimiser l'entraînement du module d'attention Teacher en apprentissage fédéré avec les séries temporelles. Les clients ayant une distribution des données complètement hétérogènes et des datasets avec des données et paramètres différents, les mises à jour directes du modèle peuvent différer complètement. Par exemple, des relevés de températures extrêmement basses présenteront une distribution totalement différente de données collectées dans un environnement plus chaud. Cela provoque des différences majeures dans l'apprentissage du modèle, et peut dégrader fortement les gradients.

Pour résoudre cette problématique, AMGF implémente plusieurs mécanismes pour mettre à jour le module Teacher de manière plus intelligente. Premièrement, pour chaque client, le serveur maintient un historique récurrent de ses mises à jour (un momentum). Ce lissage permet de filtrer le bruit des gradients individuels afin de restituer la véritable trajectoire d'apprentissage de chaque nœud. L'algorithme évalue ensuite la ressemblance des trajectoires des gradients entre les clients. Il s'en sert ensuite pour regrouper automatiquement les clients qui évoluent dans la même direction. L'algorithme évalue la stabilité de chaque groupe. Si un groupe est stable et homogène, le learning rate [52]est augmenté pour accélérer la convergence. Et finalement, le serveur agrège les avancées de chaque groupe individuellement pour avoir des mises à jour ciblés. Cela permet à chaque client, lors de l'itération suivante, de recevoir une version du modèle Teacher dont les poids sont spécifiquement adaptés à la dynamique de son groupe (cluster). Ce modèle Teacher permettra finalement à guider chacun des modules Student de ses clients via la distillation.

L'intégration d'AMGF permet d'accélérer la convergence globale grâce à l'anticipation et de protéger le modèle central contre la divergence liée à l'hétérogénéité des données. Elle garantit également une meilleure qualité des connaissances transmises aux modèles Student sur l'Edge. Avec cet algorithme, l'ensemble de ces opérations d'orchestration sont exécutés sur le serveur central, n'ajoutant ainsi aucune charge de calcul ou de communication supplémentaire pour les clients embarqués.

5.3. SoFeel, le mode fédéré décentralisé

Le troisième type d'expérimentation créé sur la plateforme, SoFeel, a démarré plus récemment et reste à ce jour en cours de développement. Il n'est pas encore entièrement implémenté, mais ses bases sont déjà créées et une bonne partie de son fonctionnement est déjà opérationnelle. Contrairement aux deux modes précédents, qui restent organisés autour du serveur central, SoFeel repose sur une architecture pair-à-pair où les clients échangent directement entre eux.

5.3.1. Génération de la topologie réseau

La première brique de ce mode a consisté à générer une topologie réseau entre les clients participants, pour simuler au mieux une situation réelle en pair-à-pair. Plusieurs modèles de génération de topologies ont donc été comparés avant de retenir une approche définitive. Des tests ont été effectués sur un modèle de graphe aléatoire (Mesh), où chaque paire de clients est reliée avec une probabilité p fixée, une méthode de graphe géométrique aléatoire (IoT RGG), positionnant virtuellement les clients dans un espace en 2D et ne reliant que les nœuds suffisamment proches. Egalement sur le modèle de Waxman, qui a une probabilité aléatoire décroissance de connexion avec un nœud à distance. Et finalement des tests sur un graphe hiérarchique (HGG), qui organise les nœuds en groupes reliés entre eux par des nœuds pivots.

Le choix s'est donc porté pour l'instant sur le modèle Mesh. Il est facile à créer, car il utilise un seul paramètre de probabilité p, facile à expliquer, et très reconnu dans la recherche grâce à des bases théoriques solides. À l'inverse, si les trois autres modèles sont plus proches d'un vrai réseau IoT, ils demandent des réglages complexes (distance, positionnement, clustering) qui rendent les expériences plus difficiles à justifier.

Ce choix reste cependant arbitraire pour l’instant, la génération de la topologie reste indépendante au reste du système. Elle est uniquement créée dans configuration de l'expérimentation, et diffusée aux clients connectés. Cela signifie, que dans le futur, nous pouvons tester d’autres générations de topologies sur la plateforme pour tester les expérimentations avec différentes situations.

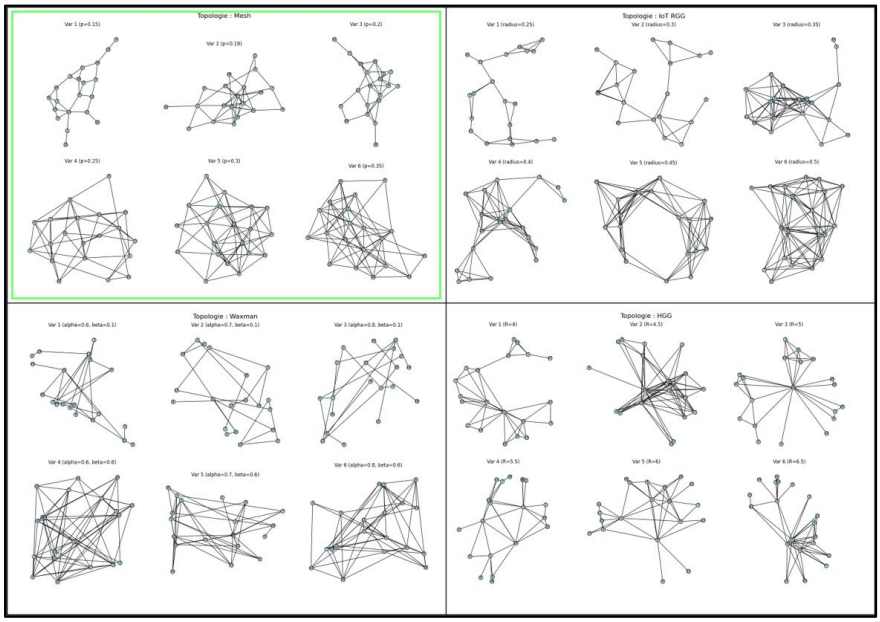

Figure 13 : Essais de génération de topologies réseau différentes avec des paramètres variables afin de trouver la meilleure configuration pour 20 nœuds

5.3.2. Entraînement et signalisation asynchrone

Le passage d'une implémentation synchrone à un mode d'expérimentation entièrement asynchrone pour SoFeel a nécessité de gros changements. Les appels bloquants, l'attente de réponse d'un client, l'attente de fin de round, et toutes ces fonctions ont été remplacés par des mécanismes asynchrones. Cela permet à chaque appareil de travailler à son rythme, sans être bloqué par le nœud le plus lent sur le réseau.

Sa signalisation, bien différente des autres modes d'expérimentations, fonctionne en asynchrone et surtout en mode pair-à-pair. À la réception du signal de démarrage d'un round, chaque client extrait la liste de ses voisins directs depuis la topologie reçue, et créé une connexion Websocket direct avec ses voisins, Grâce à celle-ci, ses voisins viendront déposer leurs mises à jour de modèle.

Chaque client entraîne son propre modèle sur ses données locales, puis, en fin de round, publie à ses voisins direct ses poids encodés avec la connexion Websocket directe en pair-à-pair. Lorsqu'un client reçoit les poids d'un voisin, il effectue une agrégation locale avec la méthode FedAvg. Pour l'instant, l'agrégation reste relativement simple, ce qui est un choix de simplicité pour cette première version du mode. Les évolutions futures prévoient d'introduire des fonctionnements d'agrégation plus complexes, créés par mes encadrants. Cela pourra facilement être changé sans modifier le reste des mécanismes de communication en P2P[53].

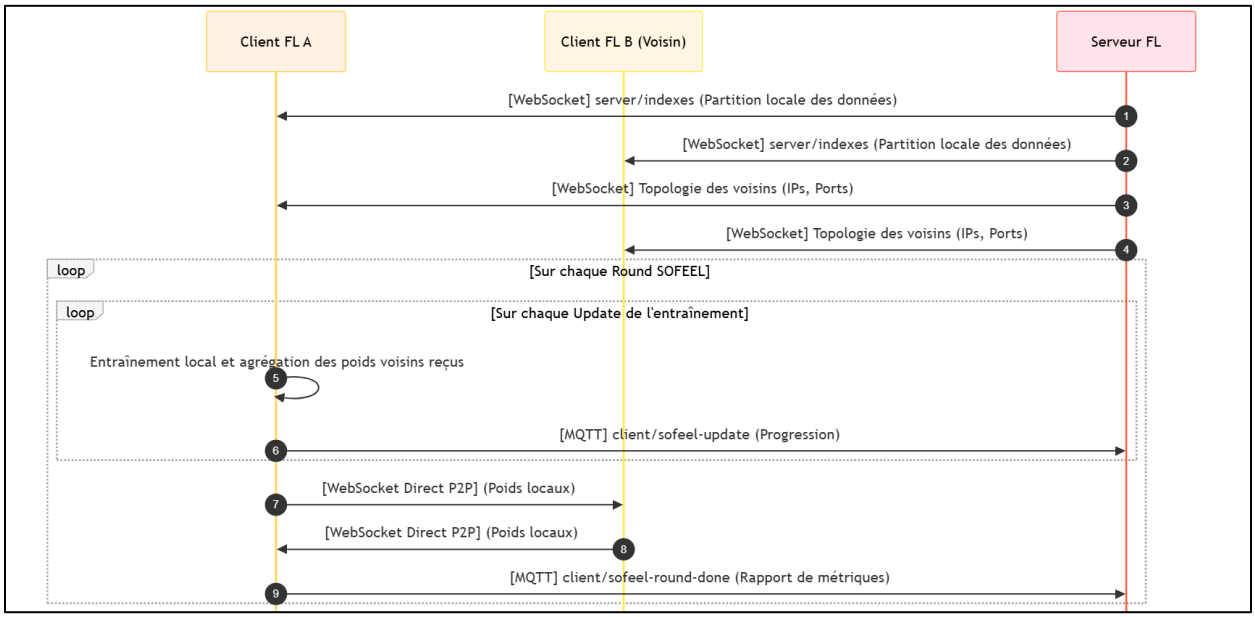

Figure 14 : Diagramme séquence du processus et de la signalisation du mode d'expérimentation SoFeel

Comme l'illustre ce schéma, ce fonctionnement s'appuie sur deux flux distincts. Lors de l'initialisation, le serveur transmet à chaque client sa partition de données et la topologie de ses voisins via WebSocket. Les nœuds exécutent ensuite leurs rounds d'apprentissage tout seul, et chaque client transmet directement ses poids mis à jour à ses voisins à travers une connexion WebSocket P2P dédiée, sans aucun transit par le serveur central. Il y a cependant toujours un envoi de données par MQTT au serveur pour envoyer les métriques d'entraînement, ce qui permet d'assurer le suivi global de l'expérimentation.

Optimisations générales, robustesse et stockage

Au-delà des différents modes d'expérimentations présentés jusqu'ici, une bonne partie de mon travail a porté sur des optimisations sur l'ensemble de la plateforme, quel que soit le type d'entraînement en cours. Ces optimisations abordent plusieurs domaines assez différents, mais qui ont tous pour objectif de rendre la plateforme plus rapide, plus fiable et plus simple à maintenir au quotidien.

5.3.3. Communications réseau

Le protocole utilisait initialement le format JSON[54] pour les échanges entre les clients et le serveur. Ce format, lisible par un humain, a un coût import, car un tableau numérique converti en texte, avec ses séparateurs et ses guillemets, produit un message nettement plus volumineux qu'une représentation binaire équivalente. Le passage à MessagePack[55], un format de données plus optimisé, a permis de grandement réduire la taille de chaque message tout en étant plus rapide à envoyer et à recevoir, puisqu'il s'appuie directement sur une représentation binaire brute sans conversion intermédiaire en chaîne de caractères.

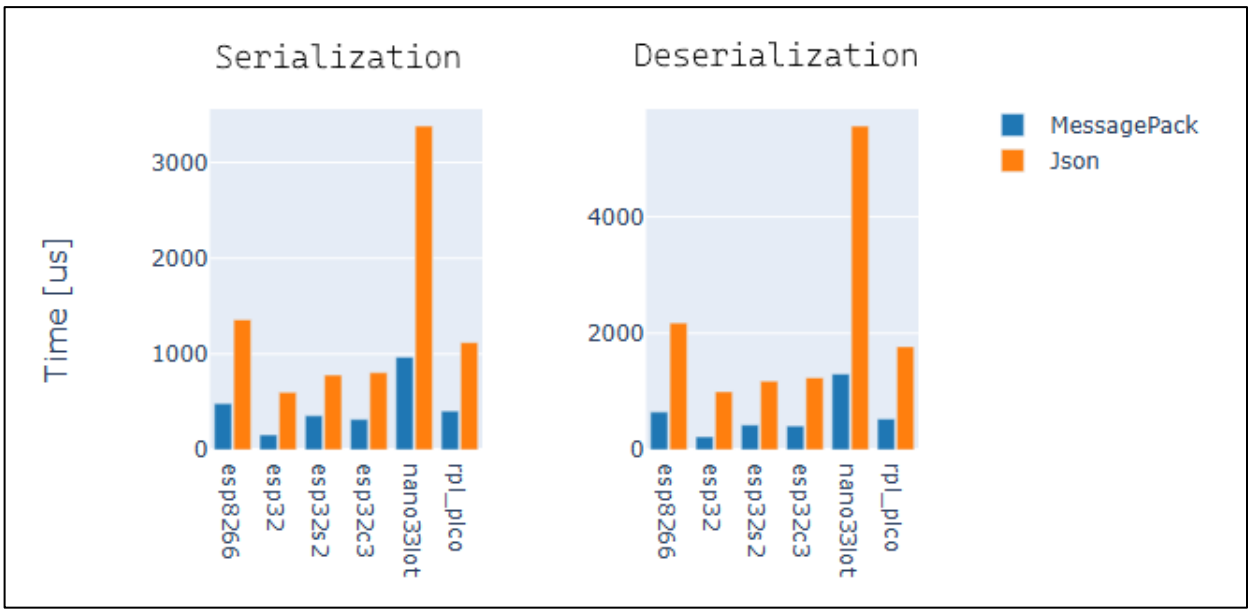

Figure 15 : Benchmark du temps de sérialisation / désérialisation de messages MessagePack / JSON sur différents appareils (source : https://github.com/fabianoriccardi/benchmark-json-messagepack)

Comme le montre ce benchmark, ce gain se remarque par un écart important lors de la désérialisation, où MessagePack reste sous la barre des 1000 µs quand JSON la dépasse fréquemment selon la carte. Cet écart, qui reste important sur plusieurs types de systèmes embarqués testés, a confirmé que le gain de MessagePack ne dépendait pas d'une architecture particulière et justifiait son utilisation généralisée sur l'ensemble de la plateforme FL.

Tous les poids de modèle, les activations ou les gradients qui sont communiquées pendant les expérimentations sont d'abord compressés dans un format binaire compact avant d'être

envoyées, ce qui permet d'éviter une surcharge de la bande passante, surtout avec les expérimentations en Split-Learning ou des échanges se font très fréquemment, deux aller-retour par update sur chacun des vingt clients, potentiellement plusieurs centaines de fois par round.

J'ai également remplacé la taille du batch[56] fixe, utilisée au départ pour tous les clients, par un calcul dynamique à chaque appareil et à leurs données. Elle se construit par le nombre de samples (échantillons) disponibles localement, du nombre d'époques souhaité, et du nombre d'updates visé pour le round :

```
raw_batch_size = (nb_samples * epochs) / nb_updates
client_batch_size = max(8, round(raw_batch_size / 8) * 8)
```

Figure 16 : Extrait du code Python permettant de déterminer la taille de batch pour chacun des clients

Cette formule ajuste automatiquement la taille de batch de sorte que chaque client effectue exactement le même nombre de mises à jour (update) du modèle par round, indépendamment de la taille de son dataset local, un client disposant de deux fois plus de données locales utilisera un batch deux fois plus grand. Une borne basse à été forcée à 8 samples par batchs, ce qui permet d'éviter des modifications de gradient trop violentes, et un arrondi au multiple de 8, qui est la norme en termes de taille de batch en Machine Learning.

5.3.4. Stockage des données

L'historique des expérimentations reposait au départ sur de multiples fichiers au format CSV. Cela devenait, notamment à cause des derniers types d'expérimentation en séries temporelles, compliqué et coûteux en mémoire, puisqu'ils devaient être rechargés intégralement à chaque consultation. J'ai migré cet historique vers une base de données SQLite[57] centralisée, avec une couche de validation de type qui s'assure que les données insérées comme celles lues respectent bien le schéma attendu.

Le schéma de la base sépare clairement plusieurs types d'information, à savoir les expérimentations elles-mêmes (identifiant, type, configuration complète, horodatage), les métriques globales par round (précision/loss, durée), les métriques détaillées par client et par round, qui diffèrent selon le mode d'entraînement, avec par exemple un score de performance et un indice de diversité Shannon pour le mode classique, ou des métriques de loss pour les séries temporelles. Il y a également les spécificités matérielles des appareils du cluster, et les tables propres au service de réservation de machines. Ce passage vers une base structurée a libéré une beaucoup de mémoire du serveur pendant les expérimentations. Lorsque plusieurs milliers d'updates se sont succédées sur les vingt clients, la différence de RAM économisée est flagrante. Cela est dû premièrement parce que seules les lignes réellement demandées sont chargées, plutôt que l'intégralité d'un fichier, et de même lorsqu'on souhaite ajouter une nouvelle information dans la base au lieu d'ouvrir le fichier CSV.

Une couche de validation a été mise en place à plusieurs endroits. Premièrement, au niveau des routes de l'API, chaque point d'entrée déclare la forme attendue de son message, ce qui permet de rejeter automatiquement une requête mal formée avec un message d'erreur explicite, avant même que le code du serveur ne soit exécuté. Au niveau de la configuration globale du serveur, l'ensemble des variables d'environnement (ports, topic, délais, …) sont validés et typés au démarrage. Une variable mal formatée provoque un arrêt immédiat avec une erreur claire, plutôt qu'un comportement indéfini découvert seulement en pleine expérimentation. Au niveau des échanges avec la base de données, le même mécanisme de validation garantit que les données insérées respectent le schéma attendu et que les données lues sont automatiquement transformées en objets Python correctement typés. Ces différents mécanismes de validation détectent les erreurs de format au plus tôt dans la chaîne, plutôt que de les découvrir en pleine expérimentation, ce qui aurait pu gâcher plusieurs heures de calcul sur la totalité des cartes.

Enfin, j'ai séparé la gestion des données légères et celle des gros fichiers. La base SQLite ne conserve que les informations très courtes, comme les configurations, les métriques et les réservations, tandis que les éléments volumineux, comme par exemple les modèles obtenus d'une expérimentation sont directement enregistrés sur le disque du serveur. Cela garantit que les sauvegardes soient presque instantanées, et permet de télécharger les gros fichiers en continu, sans jamais surcharger la mémoire du serveur.

5.3.5. Détection de pannes et robustesse des clients

Pour améliorer la robustesse des appareils et avoir une meilleure détection de pannes, chaque client publie régulièrement, un message de Heartbeat[58] contenant ses métriques systèmes collectés en temps réel :

```
{
  "client_id": "rpi-12",
  "status": "TRAINING",
  "cpu_percent": 78.5,
  "ram_percent": 62.3,
  "ram_used_mb": 612,
  "timestamp": "2026-07-15T14:32:10Z"
}
```

Figure 17 : Exemple de métriques systèmes collectés et envoyées depuis les appareils de la plateforme

Côté serveur, chaque réception de ce signal met à jour la dernière date connue d'activité du client concerné. Lorsqu'un client ne répond plus dans le délai imparti (timeout), le programme est configuré pour l'exclure du round ou de l'update. Toutes les informations sont émises en direct à l'interface web pour avoir un suivi en direct.

Avec la connexion WebSocket, un mécanisme supplémentaire de ping a été intégré. Un message est envoyé à intervalle régulier, et si aucune réponse n'est reçue dans le délai prévu, la connexion est considérée comme perdue. Cette détection est complémentaire du Heartbeat MQTT, car le ping WebSocket détecte les coupures réseau brutales, câble débranché, perte de Wi-Fi, tandis que le signal de vie MQTT détecte plutôt les blocages purement logiciels, comme un processus d'entraînement resté figé sur un calcul sans jamais planter ni se déconnecter.

Lorsqu'un blocage est détecté côté client (plantage, freeze, ou perte totale de connexion réseau) enclenche le watchdog[59] prévu à cet effet. Cela force la recréation complète du conteneur client, et va ainsi rétablir les connexions MQTT et WebSocket. L'annonce d'enregistrement est republiée, et le client redevient disponible pour le round ou l'update suivante sans intervention humaine. J'ai également mis en place un nettoyage des anciens fichiers datant de plus de 24 heures sur les appareils, pour supprimer, par exemple, les anciennes images dockers accumulés sur la carte SD après de nouveaux déploiements. Que ce soit côté serveur ou client, des mécanismes de gestion de RAM, via Garbage collector [60]ou via Tensorflow directement permettent de gérer et libérer la mémoire allouée à des moments clés (fin de rounds, saturation, …).

Nous avons également décidé d'ajouter un processus de simulation de charge pendant les expérimentations. Celui-ci permet, grâce une option configurable depuis l'interface web, de simuler une charge d'utilisation CPU / mémoire RAM avec différents niveaux (bas, moyen, élevé, critique). Ces niveaux sont déterminés par des consommations aléatoires, avec des transitions probabilistes d'un état à l'autre. Cela permet d'avoir une charge variable dans le temps, ce qui se rapproche énormément des conditions réelles d'un appareil Edge. Concrètement, le programme crée des processus qui exécutent des boucles de calcul actif pour occuper un pourcentage donné du processeur, tandis qu'un second alloue ou libère dynamiquement des blocs mémoire pour atteindre la cible de consommation RAM du moment. Ce simulateur créé permet de mesurer l'impact de chaque niveau de charge sur les temps d'entraînement et la précision (accuracy) des modèles locaux.

5.4. Plateforme démonstrateur et interface web

Cette dernière partie présente la plateforme web telle qu'elle est utilisée au quotidien par les chercheurs. C'est le tableau de bord du suivi et configuration des expérimentations, l'interface pédagogique de démonstration avec l'analyse de modèles, et le système de réservation de machines.

5.4.1. Configuration, suivi et comparaison des expérimentations

L'interface web a été développé en Nuxt 3, un framework moderne qui apporte beaucoup de fonctionnalités utiles à Vue.js, notamment le rendu côté serveur (SSR) pour optimiser le chargement, la gestion automatique des routes et une structure de projet prête pour la production.

Les mises à jour en direct des données affichées sur l'interface web fonctionne avec l'approche « Server-Sent Events », établie entre le frontend et le serveur. Il permet d'extraire les statistiques courantes de l'expérimentation en cours, et les envoyer au fur et à mesure vers le frontend. Plusieurs types d'événements sont envoyés, comme des messages de log affichés dans la console intégrée du site, les configurations d'une expérimentation, la progression round par round avec les métriques de précision ou de loss, les métriques d'entraînement local de chaque client, ou d'autres données importantes.

Les appels API du frontend sont envoyés et passés par un proxy, qui intercepte les requêtes, les rediriges vers le serveur, et injecte automatiquement l'identité de l'utilisateur authentifié qui est basée sur Authentik, le gestionnaire d'authentification de la plateforme.

Les graphiques de précision et de temps d'exécution s'adaptent automatiquement au type d'expérimentation en cours. En mode classique, l'axe des abscisses affiche les rounds, tandis qu'avec les séries temporelles ou SoFeel, il affiche les updates individuelles des clients sous forme de loss et par update plutôt que par round, puisque le Split Learning génère un volume de métriques beaucoup plus important qu'un simple envoie de poids chaque round.

Une page dédiée permet de sélectionner plusieurs expérimentations passées et de superposer leurs courbes de précision/loss et de durée. On peut donc y voir facilement les paramètres qui diffèrent entre les expérimentations sélectionnées, et comparer les résultats qu'ils ont produits.

Figure 18 : Aperçu de la page historique sur l'interface web de la plateforme

5.4.2. Interface pédagogique

Une autre page permet de voir l'évolution d'un modèle en cours d'entraînement, ou de tester un modèle déjà entraîné. Pour l'instant, deux modes d'analyse selon l'expérimentation en cours sont disponibles. En mode MNIST, un espace de dessin permet à l'utilisateur de tracer un chiffre à main levée. En mode CIFAR-100, une liste de différentes images du dataset est affichée, et il suffit d'un clic pour déclencher la prédiction de l'image.

Dans les deux cas, l'interface affiche le label de prédiction avec la plus grande probabilité ainsi qu'un classement des probabilités les plus élevées. Cette interface a notamment servi de support à une démonstration en direct auprès d'élèves de CM2, pour vulgariser et rendre visuel le principe de l'apprentissage fédéré.

Un autre mode d'analyse adapté pour les séries temporelles est prévu pour le futur, avec potentiellement l'idée de déposer un jeu de données, pour pouvoir analyser les performances de prédiction sur chacun des clients de la plateforme.

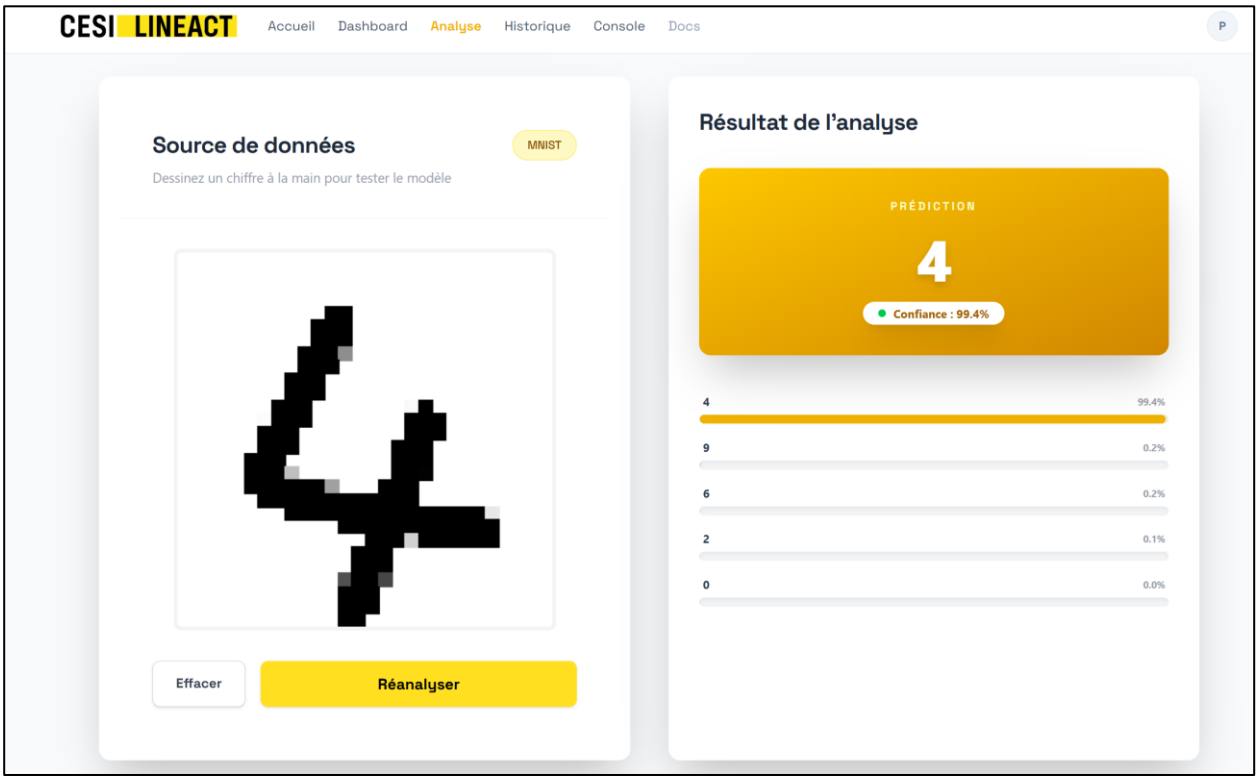

Figure 19 : Aperçu de la page analyse sur l'interface web de la plateforme.

5.4.3. Système de réservation des machines

Un système de réservation de machines, de type Container-as-a-Service[61], a pu être ajouté à la plateforme, dans l'objectif de proposer un accès facile à des appareils physiques. Cette page regroupe l'ensemble des appareils physiques présents dans la salle MLF (serveur, racks de Raspberry Pi, Jetson).

La réservation se déroule en plusieurs étapes. D'abord, le frontend interroge le serveur pour récupérer la liste des nœuds disponibles avec leur mémoire totale. La mémoire réellement proposée à la réservation est soustraite en se basant sur un pourcentage de mémoire totale disponible (25%). Cela évite de saturer un Edge qui potentiellement, tourne en parallèle avec une expérimentation d'apprentissage fédéré. L'utilisateur choisit ensuite un nœud, une limite de ressources et une durée. Le serveur étant un conteneur sur le réseau docker Swarm, va alors pouvoir simplement créer un conteneur, grâce à la librairie Docker de Python. Ce conteneur isolé sera alors créé en tant que service global sur la plateforme, mais avec uniquement l'appareil choisi en tant que worker. Il est donc créé avec les limites demandées sur le nœud cible, et retourne les informations de connexion, un accès direct en ligne de commande, copiable en un clic, ou un lien vers une l'interface d'accès instantanée Guacamole, utilisable directement depuis le navigateur. Ce conteneur créé se base sur une image simpliste Linux Alpine, permettant de l'utiliser comme un vrai système d'exploitation, avec un accès SSH natif.

Une tâche de nettoyage tourne en arrière-plan sur le serveur pour vérifier périodiquement la durée des réservations actives et supprimer automatiquement les conteneurs qui ont atteint leur durée de réservation. L'accès à ce service de réservation est sécurisé par la même authentification centralisée que le reste de la plateforme via Authentik. La supervision de l'ensemble des conteneurs et des nœuds du cluster peut toujours également se faire sur Portainer, une interface open-source déployée séparément.

Figure 20 : Aperçu de la page réservation sur l'interface web de la plateforme

6. Bilan

6.1. Les objectifs atteints

En reprenant la liste d'objectifs de la fiche de validation, nous pouvons résumer l'état d'avancement et les travaux réalisés de la façon suivante. L'installation, la mise en réseau et l'intégration plug-and-play des cartes sont en place, avec un mécanisme de reconnexion à plusieurs niveaux qui limite le besoin d'intervention manuelle. L'automatisation du déploiement des services fonctionne via une pipeline d'intégration continue, les algorithmes de recherche du laboratoire tournent désormais sur l'infrastructure réelle plutôt qu'en simulation, le protocole de communication a été adapté à chacun des modes d'expérimentations, et les interfaces web utilisateur et chercheur sont développées et utilisées au quotidien. Le mode SoFeel a pu être bien avancé, mais reste cependant encore en cours de développement et de finalisation.

6.2. Résultats des expérimentations

Les expérimentations menées sur la plateforme ont permis de valider concrètement l'apport d'AMGF sur l'entraînement du modèle en Split Learning. L'étude d'ablation menée sur l'algorithme avec plusieurs variantes (comme par exemple, no_mompd et no_momclus) confirme sa pertinence. Le fait de désactiver n'importe lequel de ses mécanismes dégrade les performances globales et altère la capacité du système à prédire les évolutions de données. À l'inverse, la version complète (AMGF full) garantit une loss minimale et constante, en particulier lors de pics de charge ou d'événements imprévus. Ces essais valident en conditions réelles la stabilisation observée en simulation par mon encadrant, tout en mesurant l'impact direct de la latence réseau et de l'hétérogénéité matérielle des cartes Edge.

Le simulateur de charge a également permis d'évaluer la résilience de la plateforme face à des conditions dégradées. Si la montée en charge induit une hausse importante du temps par round, la précision globale du modèle reste globalement constante. Avec les différentes optimisations de l'infrastructure, que ce soit au niveau de la gestion de la mémoire RAM, de la parallélisation des processus ou le passage à MessagePack, j'ai réussi à diviser par près de quatre le temps total moyen des expérimentations. Les mécanismes de robustesse (timeouts, reconnexion automatique à chaud, watchdog …) gérent également les ralentissements et pannes matérielles sans interrompre le déroulement des expérimentations bien mieux.

Du côté du réseau, le remplacement du format JSON par un format binaire optimisé (MessagePack) a permis de réduire de plus de 60 % le volume des données échangées. Cette baisse est particulièrement bénéfique pour le Split Learning, qui nécessite de nombreux échanges d'informations à chaque étape d'entraînement. En rendant le traitement des messages quasi instantané sur les cartes Raspberry Pi et Jetson, cette optimisation évite toute saturation du réseau et accélère le déroulement global des expérimentations.

Enfin, le processus d'intégration de nouveaux équipements avec l'approche « plug & play » à été validé. J'ai testé la réinstallation et la reconfiguration de trois Raspberry Pi, et cela a pris moins de 10 minutes. Ce délai est en grande partie dû aux temps d'attente incompressibles (flashage de l'OS, premier démarrage, …), ce qui signifie qu'un déploiement simultané d'un groupe de cartes beaucoup plus conséquent, s'effectuerait dans une fenêtre temporelle équivalente, ou à peine plus longue.

6.3. Une plateforme utilisée au quotidien

La plateforme est aujourd'hui utilisée activement par les chercheurs du laboratoire, aussi bien pour leurs propres expérimentations des différents algorithmes de Federated Learning que pour la réservation de machines. Le projet a également été présenté à d'autres publics, lors d'une présentation en anglais pendant de la Summer School, présentée avec Ahmed, une vidéo de présentation destinée à valoriser le travail du laboratoire auprès d'un public plus large, ou encore une demi-journée de présentation à des écoliers en CM2, pour vulgariser l'intelligence artificielle et présenter les sujets de recherches du laboratoire CESI LINEACT, dont cette plateforme FL.

Figure 21 : Présentation de la plateforme réalisée à des écoliers en CM2

7. Conclusion

Ce mémoire technique décrit l'évolution de la plateforme d'apprentissage fédéré du laboratoire CESI LINEACT, depuis un premier prototype fonctionnel mais limité à quelques cartes configurées manuellement, vers une infrastructure industrialisée utilisée quotidiennement par les chercheurs. L'enjeu central de cette mission était de combler l'écart entre les résultats obtenus en simulation par l'équipe de recherche et les contraintes réelles d'un déploiement sur du matériel embarqué hétérogène (latence réseau, puissance de calcul limitée, pannes …).

Sur le plan de l'infrastructure, ce travail a permis de transformer un ensemble de cartes configurées à la main en un cluster Docker Swarm capable d'intégrer un nouvel appareil par une simple commande, sans script d'installation ni configuration individuelle. Les mécanismes de reconnexion automatique, de détection de pannes à double niveau (MQTT et WebSocket), et de récupération après incident réduisent fortement le besoin d'intervention manuelle. L'automatisation du déploiement via une pipeline d'intégration continue, combinée à une stratégie de build en deux étages, a par ailleurs rendu la mise à jour du code du laboratoire rapide et fiable, y compris sur trois architectures matérielles différentes.

Sur le plan logiciel et algorithmique, la plateforme prend aujourd'hui en charge trois modes d'expérimentations complémentaires. Le mode classique, basé sur FedAvg et plusieurs stratégies de sélection de clients, a permis de valider concrètement le compromis entre diversité des données et performance matérielle déjà validé en simulation, avec des résultats cohérents sur l'infrastructure réelle. Le mode séries temporelles, combinant Split Learning et l'algorithme d'agrégation AMGF, a nécessité une refonte plus profonde du protocole de communication pour supporter des échanges à chaque update plutôt qu'à chaque round, tout en anticipant la bascule future vers des capteurs physiques de bâtiment intelligent. Enfin, le mode décentralisé SoFeel, encore en cours de développement, pose les bases d'un apprentissage fédéré capable de continuer à progresser même en l'absence de serveur central joignable, une propriété qui n'existait dans aucune des briques précédentes.

Au-delà de l'aspect technique, cette mission a également permis de doter le laboratoire d'un outil réellement approprié par ses utilisateurs. La plateforme est aujourd'hui sollicitée quotidiennement, aussi bien pour des expérimentations de recherche que pour la réservation de machines. Elle a aussi servi de support de valorisation auprès de publics variés, avec plusieurs présentations et démonstrations, ce qui témoigne de sa capacité à rendre concrets l'intelligence artificielle fédérée.

Plusieurs axes de travail restent néanmoins ouverts pour les prochaines alternances. Le développement du mode SoFeel et de sa topologie décentralisée doit être poursuivi, notamment avec l'introduction d'une agrégation plus complexe modèles. Le passage à de véritables capteurs physiques pour l'apprentissage sur séries temporelles permettra de valider la plateforme dans des conditions encore plus proches de l'usage final visé par le démonstrateur de bâtiment intelligent. Enfin, l'ouverture du système de réservation aux étudiants de l'école d'ingénieurs CESI élargira l'usage de l'infrastructure au-delà du seul cadre de la recherche.

8. Annexes

8.1. Bibliographie

Baahmed, A.-R.-E.-M., Dollinger, J.-F., Brahmia, M. E., & Zghal, M. (2024). Hyperparameter Impact on Computational Efficiency in Federated Edge Learning. *849-854*, (p. International Wireless Communications and Mobile Computing (IWCMC)).

Baahmed, A.-R.-E.-M., Dollinger, J.-F., Brahmia, M. E., & Zghal, M. (2026). HiFEL-OCKT: Hierarchical federated edge learning with objective congruence and multi-level knowledge. *Internet of Things*, Internet of Things.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., & Zhao, S. (2021). Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*.

Riccardi, F. (2021). *Benchmark JSON vs MessagePack on Embedded Systems*. Retrieved from GitHub: https://github.com/fabianoriccardi/benchmark-json-messagepack

Thomas, P. (2025). *Mise en place d'un système de Federated Learning à base de Raspberry Pi.* CESI LINEACT.

8.2. Glossaire

[1] **FL (Federated Learning) :** apprentissage automatique distribué et collaboratif préservant la confidentialité des données, qui restent sur les clients.
[2] **Cartes / Systèmes embarqués** : Nano-ordinateurs autonomes disposant de CPU, RAM, …
[3] **Sélection de clients** : Algorithme créé par mes encadrants choisissant à chaque cycle d'entraînement un sous-ensemble d'appareils selon des critères de performance matérielle ou de diversité de données.
[4] **Mode décentralisé (P2P / pair-à-pair)** : Architecture réseau sans serveur central où les nœuds communiquent entre voisins.
[5] **Raspberry Pi :** Nano-ordinateur à faible coût et basse consommation, utilisé comme carte d'entraînement Edge dans l'infrastructure.
[6] **Protocole de communication** : Ensemble de formats de messages régissant les échanges de données entre les clients et le serveur.
[7] **Cartes Edge** : Appareils de calcul situés en périphérie du réseau, au plus près de la source de collecte des données (capteurs).
[8] **NVIDIA Jetson** : Cartes embarquées équipées d'unités de calcul GPU intégrées, optimisées pour le traitement à haute performance et l'exécution d'IA sur l'Edge.
[9] **Clients** : Appareils de calcul connectés à l'infrastructure qui exécutent les phases d'entraînement local sur leurs propres données.
[10] **FedAvg (Federated Averaging)** : Algorithme d'apprentissage fédéré créé par mes encadrants qui calcule la moyenne pondérée des poids des modèles locaux transmis par les clients.
[11] **Split Learning** : Méthode d'apprentissage distribué où le modèle est découpé en plusieurs parties entre le client et le serveur.
[12] **AMGF** : Algorithme d'agrégation créé par mes encadrants sur le serveur stabilisant l'entraînement par mécanisme d'attention et regroupement intelligent des trajectoires de gradients.
[13] **SoFeel** : Stratégie d'apprentissage fédérée créé par mes encadrants, qui fonctionne de manière asynchrone sur une topologie pair-à-pair.
[14] **Conteneurs** : Systèmes virtuels légers isolant une application pour garantir son exécution identique quel que soit l'environnement.
[15] **Docker Swarm** : Outil d'orchestration natif de Docker permettant de gérer un cluster de machines (serveurs et cartes embarquées).
[16] **Clients Python** : Programmes s'exécutant sur chaque carte embarquée pour gérer les calculs locaux et la communication avec le serveur.
[17] **Broker MQTT** : Protocole de messagerie léger permettant le routage et la distribution de messages de type Publication/Abonnement.
[18] **WebSocket** : Protocole de communication bidirectionnel et temps réel avec une connexion permanente TCP entre le serveur et les clients.
[19] **FastAPI** : Framework web Python moderne et rapide permettant de créer des API REST performantes.
[20] **API REST** : Interface basée sur les méthodes standard du protocole HTTP (GET, POST, …) pour échanger des données structurées.
[21] **Topics MQTT** : Canaux sous forme de chemins hiérarchiques permettant d'orienter et de filtrer les messages sur le broker MQTT.
[22] **Passe Forward** : Étape d'un réseau de neurones où les données d'entrée traversent les couches successives pour générer une prédiction.
[23] **Passe Backward (Rétropropagation)** : Étape de calcul de la propagation de l'erreur du modèle de la sortie vers l'entrée pour ajuster ses poids.
[24] **QoS 1** : Niveau de garantie du protocole MQTT assurant qu'un message est délivré au moins une fois au destinataire.
[25] **Nœud (Node)** : Entité informatique matérielle (serveur ou carte embarquée) membre du réseau de communication.
[26] **Mémoire RAM** : Mémoire vive d'un équipement informatique utilisée pour stocker temporairement les données et programmes en cours d'exécution.
[27] **Traefik** : Reverse proxy et équilibreur de charge automatisé découvrant et orientant le trafic web vers les différents conteneurs Docker.
[28] **Reverse Proxy :** Serveur intermédiaire recevant les requêtes externes pour les rediriger vers les bons services internes tout en apportant de la sécurité.
[29] **TLS / HTTPS** : Protocole de chiffrement garantissant la confidentialité et l'intégrité des échanges sur le réseau Web.
[30] **Authentik** : Solution open-source de gestion centralisée des identités et des accès pour sécuriser l'authentification sur la plateforme.
[31] **OIDC (OpenID Connect)** : Protocole d'authentification basé sur OAuth 2.0 permettant de réaliser de l'authentification unique (SSO).
[32] **Apache Guacamole** : Passerelle d'accès distant sans client permettant de contrôler une machine directement depuis un navigateur web.
[33] **Portainer** : Solution open-source simplifiant l'administration, le suivi des logs et la supervision des conteneurs Docker et Swarm.

[34] **SSH** : Protocole de communication sécurisé permettant d'ouvrir un terminal en ligne de commande à distance sur une machine.
[35] **VNC** : Protocole de partage de bureau permettant de visualiser et contrôler à distance l'interface graphique d'une machine.
[36] **Hostname** : Nom unique attribuable à un équipement réseau pour l'identifier indépendamment de son adresse IP.
[37] **Rounds :** Une étape globale complète dans l'apprentissage fédéré comprenant la distribution du modèle, l'entraînement local et l'agrégation finale.
[38] **Update** : Étape intermédiaire dans l'apprentissage fédéré en mode séries temporelles, exécutée par un client sur une portion de son jeu de données (batch/époque).
[39] **Pipeline CI/CD** : Chaîne d'automatisation assurant la compilation, les tests et le déploiement automatique du code à chaque modification.
[40] **GHCR (GitHub Container Registry)** : Registre en ligne de GitHub utilisé pour héberger, versionner et distribuer les images de conteneurs Docker.
[41] **TensorFlow** : Bibliothèque logicielle open-source très connue de Machine Learning pour créer et entraîner des réseaux de neurones.
[42] **Design Patterns** : Patrons de conception logicielle offrant des architectures standards pour organiser, créer et étendre le code de manière modulaire.
[43] **CPU** : Processeur principal d'un ordinateur chargé d'exécuter les instructions du système et les calculs généraux.
[44] **Keras** : Interface de haut niveau basée sur TensorFlow simplifiant la définition et l'entraînement de modèles de Deep Learning.
[45] **Précision/Accuracy** : Métrique de performance mesurant le pourcentage de prédictions correctes réalisées par le modèle sur un jeu de données.
[46] **Réseau de neurones** : Modèle d'IA composé de couches d'unités de calcul interconnectées s'ajustant par apprentissage pour résoudre une tâche.
[47] **Réservoir :** Technique où la première partie d'un réseau récurrent conserve des poids fixes pour extraire dynamiquement les caractéristiques temporelles.
[48] **GRU** : Type de couche de réseau récurrent optimisée pour capturer les dépendances temporelles dans les séries temporelles.
[49] **Gradient** : Vecteur indiquant la direction et l'amplitude de la correction à apporter aux poids pour améliorer modèle.
[50] **Distillation** : Technique où le modèle léger « Student » apprend en cherchant à imiter le comportement du modèle plus complexe « Teacher ».
[51] **Loss (Fonction de perte)** : Valeur numérique quantifiant l'écart entre les prédictions d'un modèle et les valeurs réelles visées
[52] **Learning Rate (Taux d'apprentissage)** : Hyperparamètre définissant l'ampleur du pas de mise à jour des poids du modèle à chaque étape d'optimisation.
[53] **P2P (Peer-to-Peer)** : Architecture informatique où chaque nœud communique et échange directement avec ses pairs sans passer par un serveur central.
[54] **JSON** : Format textuel léger et lisible utilisé pour structurer et échanger des données.
[55] **MessagePack** : Format de sérialisation binaire ultra-compact et rapide, plus performant que le JSON pour la transmission réseau.
[56] **Taille du batch (Batch Size)** : Nombre d'échantillons de données traités simultanément par le modèle avant chaque mise à jour des poids.
[57] **SQLite** : Moteur de base de données relationnelle léger stocké dans un simple fichier local, ne nécessitant aucun serveur dédié.
[58] **Heartbeat** : Message périodique transmis par un client pour indiquer au serveur qu'il est fonctionnel et en ligne.
[59] **Watchdog** : Mécanisme de surveillance réinitialisant ou redémarrant automatiquement un service ou un conteneur bloqué.
[60] **Garbage Collector** : Composant logiciel gérant l'allocation mémoire en libérant automatiquement les objets qui ne sont plus utilisés.
[61] **Container-as-a-Service (CaaS)** : Service permettant aux utilisateurs de réserver, instancier et gérer des conteneurs isolés à la demande sur l'infrastructure.