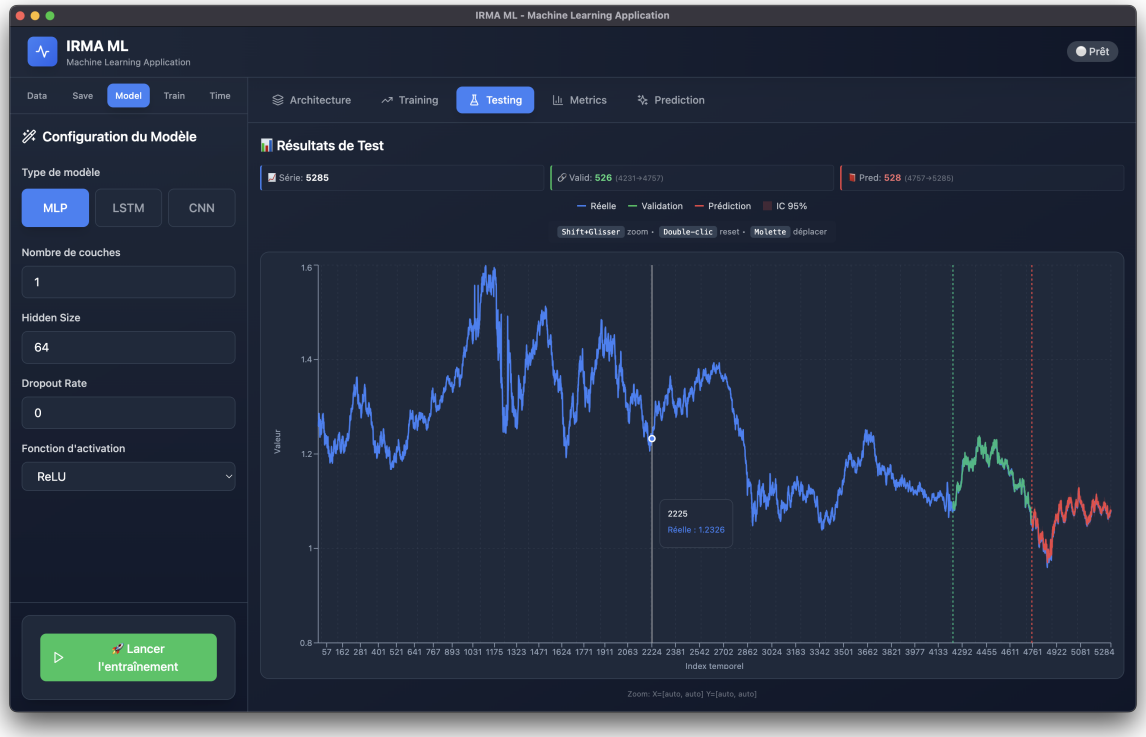
Cahier des Charges

Application de Prédiction de Séries Temporelles par Réseaux de Neurones Profond

MLApp



Projet Génie Logiciel

Membres du groupe:
Gatien Séguy – Serveur IA
Maxime Degraeve – Interface Utilisateur
Sofiane Boucherba – Serveur Data

29 janvier 2026

Table des matières

Chapitre 1

Introduction

1.1 Contexte du Projet

L'application **MLApp** est une solution complète de prédiction de séries temporelles utilisant des réseaux de neurones profonds. Elle permet aux utilisateurs de charger des données temporelles, configurer et entraîner différentes architectures de réseaux (MLP, CNN, LSTM), visualiser les performances en temps réel via streaming SSE, et effectuer des prédictions multi-pas avec intervalles de confiance.

Objectifs Principaux

- Fournir une interface intuitive pour la manipulation de réseaux de neuronnes et de séries temporelles
- Supporter plusieurs architectures de réseaux de neurones
- Permettre le suivi en temps réel de l'entraînement
- Implémenter plusieurs stratégies de prédiction multi-pas
- Calculer et afficher les intervalles de confiance des prédictions
- Assurer la persistance des modèles et contextes d'entraînement

1.2 Architecture Trois-Tiers

L'application suit une architecture **trois-tiers** séparant clairement les responsabilités entre trois composants distincts qui communiquent via des API REST.

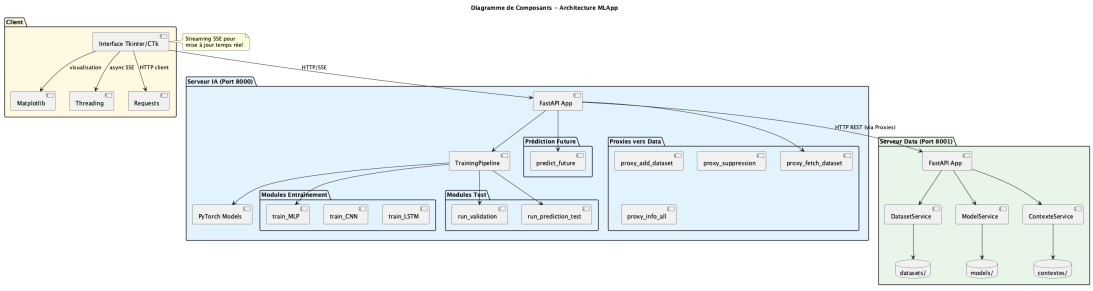


FIGURE 1.1 – Diagramme de composants – Architecture globale MLApp

1.2.1 Composant 1 : Interface Utilisateur (UI)

Technologies UI

- **Framework** : Tkinter / CustomTkinter
- **Graphiques** : Matplotlib avec backend TkAgg
- **Threading** : Pour le streaming non-bloquant des événements SSE
- **Client HTTP** : Bibliothèque `requests`

L'interface utilisateur est responsable de :

- La collecte des paramètres de configuration (modèle, entraînement, données)
- L'envoi des requêtes HTTP au Serveur IA
- La réception et l'affichage du streaming SSE en temps réel
- La mise à jour dynamique des graphiques (loss, validation, prédiction)
- La gestion de l'état global de l'application

1.2.2 Composant 2 : Serveur IA (Port 8000)

Technologies Serveur IA

- **Framework API** : FastAPI avec uvicorn
- **Deep Learning** : PyTorch (CPU/CUDA/MPS)
- **Streaming** : Server-Sent Events (SSE)
- **Validation** : Modèles Pydantic

Le Serveur IA constitue le cœur du système :

- Orchestration du flux de données entre l'UI et le Serveur Data
- Prétraitement des séries temporelles (normalisation, split)
- Entraînement des modèles (MLP, CNN, LSTM)
- Évaluation automatique et génération de prédictions
- Streaming des métriques en temps réel

1.2.3 Composant 3 : Serveur Data (Port 8001)

Technologies Serveur Data

- **Framework API** : FastAPI
- **Stockage Datasets** : Fichiers JSON
- **Stockage Modèles** : Fichiers .pth (PyTorch)
- **Stockage Contextes** : Fichiers JSON
- **Validation** : Modèles Pydantic

Le Serveur Data centralise toute la persistance :

- Gestion des datasets (CRUD)
- Sauvegarde et chargement des modèles entraînés
- Persistance des contextes d'entraînement
- Communication exclusive avec le Serveur IA via REST

1.3 Méthodologie de Conception

Ce projet suit une méthodologie de conception orientée objet utilisant UML pour spécifier l'architecture et la structure du système.

Phases de Conception

1. Conception Architecturale

- Découpe du système en sous-systèmes faiblement couplés
- Spécification des responsabilités de chaque composant
- Définition des interfaces de communication (API REST)

2. Conception Détaillée

- Structure orientée objet (classes, attributs, méthodes)
- Diagrammes de séquence pour les flux principaux
- Diagrammes de classes pour chaque sous-système

Chapitre 2

Spécifications Fonctionnelles

Ce chapitre détaille exhaustivement toutes les fonctionnalités de l'application MLApp, organisées en 7 groupes principaux.

2.1 F-1 : Gestion des Données

2.1.1 F-1.1 : Importation de Données

Description

L'utilisateur peut importer des séries temporelles depuis un fichier local au format JSON. Les données sont validées puis transmises au Serveur Data pour stockage persistant.

Format des données attendu :

```json
{
  "timestamps": [
    "2024-01-01T00:00:00",
    "2024-01-02T00:00:00",
    "2024-01-03T00:00:00"
  ],
  "values": [100.5, 101.2, null, 102.8]
}
```

Listing 2.1 – Format TimeSeriesData

Pré-conditions :

— Le fichier doit être au format JSON valide

— Les données doivent contenir deux listes : `timestamps` et `values`

— Les longueurs des deux listes doivent être strictement identiques

— Les timestamps doivent être au format ISO 8601

Post-conditions :

— Les données sont stockées dans le Serveur Data sous forme de fichier JSON

— Un nom unique est attribué au dataset

— Le dataset devient disponible pour la sélection dans l'interface
— Les métadonnées (date début, date fin, pas temporel) sont extraites

Flux principal :

1. L'utilisateur clique sur « Charger Dataset »
2. Le système ouvre une boîte de dialogue de sélection de fichier
3. L'utilisateur sélectionne un fichier JSON
4. Le système valide le format des données via Pydantic (`TimeSeriesData`)
5. Le système demande un nom pour le dataset
6. Le système vérifie l'unicité du nom auprès du Serveur Data
7. Les données sont envoyées au Serveur Data via le Serveur IA (proxy)
8. Le système confirme l'importation avec un message de succès

Flux alternatifs :

— **4a.** Format invalide → Affichage d'un message d'erreur précis (validation Pydantic)
— **6a.** Nom déjà existant → Demande d'un nouveau nom unique
— **7a.** Erreur réseau → Affichage d'une erreur de connexion

2.1.2 F-1.2 : Sélection de Dataset

Description

L'utilisateur peut parcourir et sélectionner un dataset parmi ceux disponibles sur le Serveur Data. La sélection permet de visualiser les métadonnées et de choisir une plage temporelle.

Informations affichées pour chaque dataset :

— Nom du dataset
— Date de début (premier timestamp)
— Date de fin (dernier timestamp)
— Pas temporel (intervalle entre deux points consécutifs)
— Nombre total de points

Flux principal :

1. L'utilisateur ouvre la fenêtre de gestion des datasets
2. Le système requête le Serveur IA (`POST /datasets/info_all`)
3. Le Serveur IA proxy vers le Serveur Data
4. La liste des datasets avec métadonnées est affichée dans un Treeview
5. L'utilisateur sélectionne un dataset
6. L'utilisateur peut optionnellement définir une plage de dates
7. La sélection est validée et stockée dans les paramètres de configuration

2.1.3 F-1.3 : Filtrage Temporel

Description

L'utilisateur peut restreindre les données à une période spécifique en définissant une date de début et une date de fin. Un pas d'échantillonnage peut également être appliqué.

Paramètres de filtrage :

— `date_debut` : Date de début de la période (format YYYY-MM-DD)
— `date_fin` : Date de fin de la période (format YYYY-MM-DD)
— `pas_temporel` : Entier ≥ 1 (1 = tous les points, 2 = 1 sur 2, etc.)

2.1.4 F-1.4 : Suppression de Dataset

Description

L'utilisateur peut supprimer définitivement un dataset du Serveur Data. Une confirmation est demandée avant la suppression.

Flux principal :

1. L'utilisateur sélectionne un dataset dans la liste
2. L'utilisateur clique sur « Supprimer »
3. Le système demande confirmation
4. Si confirmé, le système envoie une requête de suppression
5. Le fichier JSON correspondant est supprimé du Serveur Data
6. La liste est rafraîchie

2.2 F-2 : Configuration du Modèle

2.2.1 F-2.1 : Choix de l'Architecture

Description

L'utilisateur choisit le type de réseau de neurones à utiliser parmi trois architectures disponibles : MLP (Perceptron Multi-Couches), CNN (Réseau Convolutif) ou LSTM (Long Short-Term Memory).

TABLE 2.1 – Comparaison des architectures disponibles

Architecture	Avantages	Cas d'usage
MLP	Simple, rapide à entraîner	Séries sans forte dépendance temporelle
CNN	Capture les patterns locaux	Séries avec motifs répétitifs
LSTM	Mémoire long terme	Séries avec dépendances longues

2.2.2 F-2.2 : Paramètres MLP

TABLE 2.2 – Paramètres de configuration MLP

Paramètre	Type	Description	Défaut
`nb_couches`	int ≥ 1	Nombre de couches cachées	2
`hidden_size`	int > 0	Nombre de neurones par couche	64
`dropout_rate`	float $\in [0, 1]$	Taux de dropout (régularisation)	0.0
`fonction_activation`	enum	ReLU, GELU, tanh, sigmoid, leaky_relu	ReLU

2.2.3 F-2.3 : Paramètres CNN

TABLE 2.3 – Paramètres de configuration CNN

Paramètre	Type	Description	Défaut
`nb_couches`	int ≥ 1	Nombre de couches convolutives	2
`hidden_size`	int > 0	Nombre de filtres par couche	64
`kernel_size`	int > 0	Taille du noyau de convolution	3
`stride`	int > 0	Pas d'application du noyau	1
`padding`	int ≥ 0	Padding autour des données	0
`fonction_activation`	enum	Type d'activation	ReLU

2.2.4 F-2.4 : Paramètres LSTM

TABLE 2.4 – Paramètres de configuration LSTM

Paramètre	Type	Description	Défaut
`nb_couches`	int ≥ 1	Nombre de couches LSTM empilées	2
`hidden_size`	int > 0	Dimension de l'état caché	64
`bidirectional`	bool	LSTM bidirectionnel	False
`batch_first`	bool	Dimension batch en premier	True

2.3 F-3 : Configuration de l'Entraînement

2.3.1 F-3.1 : Paramètres Temporels

TABLE 2.5 – Paramètres temporels

Paramètre	Type	Description	Défaut
`horizon`	int ≥ 1	Nombre de pas à prédire	1
`window_size`	int ≥ 1	Taille de la fenêtre d'observation	15
`portion_decoupage`	float $\in (0, 1)$	Ratio train/(validation+test)	0.8

2.3.2 F-3.2 : Paramètres d'Optimisation

TABLE 2.6 – Paramètres d'optimisation

Paramètre	Type	Description	Défaut
`optimisateur`	enum	Adam, SGD, RMSprop, Adagrad, Adadelta	Adam
`learning_rate`	float > 0	Taux d'apprentissage	0.001
`decroissance`	float ≥ 0	Weight decay (L2 regularization)	0.0
`scheduler`	enum	Plateau, Cosine, OneCycle, None	None
`patience`	int ≥ 1	Patience pour scheduler Plateau	10

2.3.3 F-3.3 : Paramètres de Fonction de Perte

TABLE 2.7 – Fonctions de perte disponibles

Fonction	Formule	Usage
`MSE`	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Standard, sensible aux outliers
`MAE`	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Robuste aux outliers
`Huber`	Combinaison MSE/MAE	Compromis

2.3.4 F-3.4 : Paramètres d'Entraînement

TABLE 2.8 – Paramètres d'entraînement

Paramètre	Type	Description	Défaut
`nb_epochs`	int ≥ 1	Nombre d'époques d'entraînement	1000
`batch_size`	int ≥ 1	Taille des mini-lots	4
`clip_gradient`	float > 0	Seuil de clipping du gradient	None

2.4 F-4 : Entraînement et Monitoring

2.4.1 F-4.1 : Pipeline en 3 Phases

Pipeline d'Entraînement et d'Évaluation

Le pipeline se décompose en **trois phases** exécutées séquentiellement, dont les résultats sont affichés dans l'onglet Testing :
Phase 1 – Entraînement (80%) : Apprentissage sur les données d'entraînement avec rétropropagation. Affichage de la courbe de loss dans l'onglet Training.
Phase 2 – Validation (10%) : Test ONE_STEP avec teacher forcing (recalibration immédiate sur y_{true}). Calcule le `residual_std` nécessaire pour les intervalles de confiance. Affichage en **courbe verte**.
Phase 3 – Test Prédictif (10%) : Test ONE_STEP avec recalibration sur y_{true}. Utilise `residual_std` pour afficher l'intervalle de confiance à 95%. Affichage en **courbe rouge** avec halo.

Extrait du code `execute_full_pipeline()`

```
# ========== PHASE 1 : ENTRAINEMENT ==========
yield sse({"type": "phase", "phase": "train", "status": "start"})
for msg in self.run_training():
    yield sse(msg)

# ========== PHASE 2 : VALIDATION ==========
yield sse({"type": "phase", "phase": "validation", "status": "start
    "})
for msg in self.run_validation():
    yield sse(msg)

# ========== PHASE 3 : TEST PREDICTIF ==========
yield sse({"type": "phase", "phase": "prediction", "status": "start
    "})
for msg in self.run_prediction_test():
    yield sse(msg)
```

Découpage des données (fonction `split_data_three_way()`) :

- `X_train`, `y_train` : portion_decoupage% (ex : 80%) des données pour l'entraînement
- `X_val`, `y_val` : $(100 - portion\_decoupage/2)\%$ (ex : 10%) des données pour la validation (calcul `residual_std`)
- `X_test`, `y_test` : $(100 - portion\_decoupage/2)\%$ (ex : 10 %) des données pour le test prédictif (avec IC)
- **Série réelle** (bleu) : Les données complètes (`serie_complete`)
- **Validation** (vert) : Résultats de `run_validation()` – événements `val_pair`
- **Prédiction** (rouge + halo) : Résultats de `run_prediction_test()` – événements `pred_point` avec `low`/`high`

Onglets de l'interface :

- **Représentation** : Visualisation de la série temporelle brute

— **Training** : Courbe de loss pendant la Phase 1
— **Testing** : Résultats des Phases 2 et 3 (vert + rouge superposés sur la série bleue)
— **Metrics** : Métriques globales (MSE, MAE, RMSE, R^2)
— **Prediction** : Prédiction libre sur horizon H (données d'entraînement ou nouvelles données)

2.4.2 F-4.2 : Streaming SSE (Server-Sent Events)

Description

L'entraînement utilise le streaming SSE pour la communication temps réel entre le Serveur IA et l'Interface Utilisateur. Chaque événement est formaté en JSON et envoyé via `StreamingResponse`.

Types d'événements SSE :

```
// Informations de split
{"type": "split_info", "n_train": 800, "n_val": 100, "n_test": 100}

// Serie complete pour affichage
{"type": "serie_complete", "values": [...]}

// Debut de phase
{"type": "phase", "phase": "train", "status": "start"}

// Progression epoch par epoch
{"type": "epoch", "epochs": 1, "avg_loss": 0.4532, "epoch_s": 12.5}
{"type": "epoch", "epochs": 2, "avg_loss": 0.3821, "epoch_s": 11.8}
```

Listing 2.2 – Événements SSE – Phase Entraînement

```
// Debut phase validation
{"type": "phase", "phase": "validation", "status": "start"}

// Info de depart
{"type": "val_start", "n_points": 500, "dims": 1, "idx_start": 4015}

// Paires y_true / y_pred pour chaque point
{"type": "val_pair", "idx": 4015, "y": 1.234, "yhat": 1.228}
{"type": "val_pair", "idx": 4016, "y": 1.241, "yhat": 1.235}
// ... pour chaque point de validation

// Fin validation avec metriques et residual_std
{"type": "val_end",
 "metrics": {"overall_mean": {"MSE": 0.0012, "MAE": 0.025}},
 "all_true": [...], "all_predictions": [...],
 "residual_std": 0.035}

{"type": "phase", "phase": "validation", "status": "end"}
```

Listing 2.3 – Événements SSE – Phase Validation (courbe verte)

```
// Debut phase prediction
{"type": "phase", "phase": "prediction", "status": "start"}

// Info de depart
{"type": "pred_start", "n_steps": 500, "strategy": "one_step",
 "idx_start": 4515, "window_size": 15,
 "config": {"model_type": "mlp"}}

// Points de prediction avec IC 95%
{"type": "pred_point", "step": 1, "idx": 4515,
 "yhat": 1.089, "y": 1.092, "low": 1.020, "high": 1.158}
{"type": "pred_point", "step": 2, "idx": 4516,
 "yhat": 1.085, "y": 1.088, "low": 1.016, "high": 1.154}
// ... pour chaque point de test

// Fin prediction avec resultats complets
{"type": "pred_end",
 "predictions": [...], "pred_low": [...], "pred_high": [...],
 "y_true": [...], "metrics": {"MSE": 0.0015, "MAE": 0.028},
 "idx_start": 4515}

{"type": "phase", "phase": "prediction", "status": "end"}
```

Listing 2.4 – Événements SSE – Phase Test Prédictif (courbe rouge + IC)

```
// Donnees pour affichage final dans l'onglet Testing
{"type": "final_plot_data",
 "series_complete": [...],  // Serie bleue
 "val_predictions": [...],  // Courbe verte
 "val_true": [...],
 "pred_predictions": [...], // Courbe rouge
 "pred_low": [...],         // Halo bas
 "pred_high": [...],        // Halo haut
 "pred_true": [...],
 "idx_val_start": 4015,
 "idx_test_start": 4515,
 "val_metrics": {...},
 "pred_metrics": {...}}

// Fin du pipeline
{"type": "fin_pipeline", "done": 1}
```

Listing 2.5 – Événements SSE – Données finales

2.4.3 F-4.3 : Visualisation Temps Réel

L'interface met à jour dynamiquement plusieurs graphiques :

— **Graphique Loss** : Courbe de la loss moyenne par époque

— **Graphique Validation** : Comparaison y_{true} vs $\hat{y}$ sur l'ensemble de validation

— **Graphique Prédiction** : Prédictions avec halo de confiance

— **Métriques** : Affichage MSE, MAE, RMSE, R^2 en temps réel

2.4.4 F-4.4 : Arrêt de l'Entraînement

Description

L'utilisateur peut interrompre l'entraînement à tout moment. Un flag `stop_flag` est positionné et vérifié à chaque époque par le générateur d'entraînement.

Mécanisme :

1. L'utilisateur clique sur « Arrêter »
2. Une requête `POST /stop_training` est envoyée
3. Le flag `stop_flag` du `TrainingPipeline` passe à `True`
4. La boucle d'entraînement vérifie le flag et interrompt le générateur
5. Un événement SSE de fin est envoyé

2.5 F-5 : Stratégies de Prédiction Multi-Pas

Fondement Scientifique

Les stratégies de prédiction implémentées sont basées sur la littérature scientifique, notamment :

— Taieb & Hyndman (2014) – *A review of multi-step ahead forecasting*

— Chevillon (2007) – *Direct multi-step estimation and forecasting*

— Benidis et al. (2022) – *Deep Learning for Time Series Forecasting : Tutorial*

2.5.1 F-5.0 : Stratégies dans l'Onglet Testing

L'onglet Testing affiche les résultats de deux méthodes d'évaluation qui utilisent toutes deux la stratégie **ONE_STEP avec recalibration sur** y_{true} :

TABLE 2.9 – Méthodes d'évaluation dans l'onglet Testing

Courbe	Méthode	Description	Fonction
Vert	Validation	Teacher forcing avec recalibration immédiate. Calcule `residual_std`	`run_validation()`
Rouge	Test Prédictif	c IC 95% utilisant `residual_std`	`run_prediction_test()`

Courbe Verte – Validation (`run_validation()`)

Phase 2 du pipeline. Pour chaque point de validation :

1. Prédiction $\hat{y}$ à partir de la fenêtre courante
2. Comparaison avec y_{true} pour calculer les résidus

3. Envoi événement `val_pair` avec `y` et `yhat`

À la fin, calcul de `residual_std = std(y_true - y_pred)` pour les IC.

Courbe Rouge – Test Prédictif (`run_prediction_test()`)

Phase 3 du pipeline. Pour chaque point de test :

1. Prédiction $\hat{y}$ à partir du contexte
2. Calcul IC 95% : `low = yhat - z × residual_std`, `high = yhat + z × residual_std`
3. Recalibration : `context[-1] = normalize(y_true)`
4. Envoi événement `pred_point` avec `yhat`, `y`, `low`, `high`

Le halo rouge représente l'intervalle de confiance à 95%.

Extrait du code de `run_prediction_test()`

```python
# Intervalle de confiance (constant pour one-step avec recalib)
uncertainty = residual_std * z_score  # z_score = 1.96 pour 95%
low = float(y_pred - uncertainty)
high = float(y_pred + uncertainty)

# ONE-STEP avec recalibration : utiliser la VRAIE valeur
if y_true is not None:
    next_val = normalize(y_true)  # Recalibration sur y_true
else:
    next_val = y_pred_norm  # Mode recursif si pas de y_true

context = np.roll(context, -1)
context[-1] = next_val
```

2.5.2 F-5.1 : Stratégies Disponibles

TABLE 2.10 – Stratégies de prédiction implémentées

Stratégie	Description
`ONE_STEP`	Prédiction 1 pas + recalibration immédiate sur y_{true}.
`RECALIBRATION`	Prédiction multi-pas + recalibration périodique tous les N pas
`RECURSIVE`	Prédiction autorégressive pure, sans recalibration (erreurs cumulées)
`DIRECT`	Prédiction multi-horizon de H pas simultanément

Note sur l'Onglet Prediction

L'onglet **Prediction** permet d'effectuer une prédiction libre sur un horizon H. Cette prédiction peut être faite :

— Sur les données d'entraînement (pour visualiser l'extrapolation)

— Sur de nouvelles données (utilisation en production)

Dans ce cas, si aucune donnée réelle n'est disponible pour recalibration, le mode bascule automatiquement en **RECURSIVE** (ligne 770-771 du code : `next_val = y_pred_norm`).

2.5.3 F-5.1 : Stratégie ONE_STEP (Gold Standard)

Recommandation

Cette stratégie est considérée comme le **gold standard** car elle évalue la vraie capacité prédictive du modèle sans accumulation d'erreurs.

Algorithme :

1. Extraire la fenêtre courante de taille `window_size`
2. Normaliser la fenêtre
3. Prédire le pas suivant avec le modèle
4. Dénormaliser la prédiction
5. Calculer l'intervalle de confiance : $IC = \hat{y} \pm z \times \sigma_{residual}$
6. **Recalibration immédiate** : remplacer la dernière valeur de la fenêtre par y_{true}
7. Répéter pour chaque pas

Avantages :

— Incertitude constante ($\sigma_{residual}$)

— Pas d'accumulation d'erreurs

— Évaluation réaliste de la performance

2.5.4 F-5.2 : Stratégie RECALIBRATION

Algorithme :

1. Prédire plusieurs pas en utilisant les prédictions précédentes
2. Tous les `recalib_every` pas, recalibrer sur y_{true} si disponible
3. L'incertitude croît avec le nombre de pas depuis la dernière recalibration :

$$\sigma_{step} = \sigma_{residual} \times \sqrt{steps\_since\_recalib + 1}$$

2.5.5 F-5.3 : Stratégie RECURSIVE

Algorithme :

1. Prédire le pas suivant
2. Utiliser la prédiction comme entrée pour le pas suivant (autorégression)
3. **Jamais de recalibration**
4. L'incertitude croît exponentiellement :

$$\sigma_{step} = \sigma_{residual} \times \sqrt{step + 1}$$

Attention

Cette stratégie diverge rapidement car les erreurs s'accumulent. L'intervalle de confiance devient très large après quelques pas. Utile uniquement pour la prédiction « pure » dans le futur sans données réelles.

2.5.6 F-5.4 : Stratégie DIRECT

Algorithme :

1. Le modèle prédit directement H pas d'un coup
2. Évite l'accumulation d'erreurs autorégressive
3. Nécessite un modèle entraîné pour la sortie multi-horizon

2.5.7 F-5.5 : Calcul des Intervalles de Confiance

Les intervalles de confiance à 95% sont calculés avec :

$$IC_{95\%} = \hat{y} \pm z_{0.975} \times \sigma_{residual} \times \sqrt{h} \tag{2.1}$$

Où :

— $\hat{y}$ est la prédiction dénormalisée

— $z_{0.975} = 1.96$ pour un niveau de confiance de 95%

— $\sigma_{residual}$ est l'écart-type des résidus calculé en phase de validation

— h est l'horizon de prédiction (nombre de pas depuis la dernière recalibration)

2.6 F-6 : Métriques d'Évaluation

TABLE 2.11 – Métriques d'évaluation

Métrique	Formule	Interprétation
MSE	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Erreur quadratique moyenne
MAE	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Erreur absolue moyenne
RMSE	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$	Racine de MSE
R^2	$1 - \frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$	Coefficient de détermination
MAPE	$\frac{100}{n}\sum_{i=1}^{n}\left\lvert\frac{y_i-\hat{y}_i}{y_i}\right\rvert$	Erreur pourcentage absolue

2.7 F-7 : Persistance des Modèles et Contextes

2.7.1 F-7.1 : Sauvegarde de Modèle

Description

À la fin de l'entraînement, le modèle peut être sauvegardé sur le Serveur Data. Le fichier .pth contient le `state_dict` du modèle PyTorch, encodé en Base64 pour le transfert HTTP.

Données sauvegardées :

— `state_dict` : Poids du modèle PyTorch

— `norm_params` : Paramètres de normalisation (mean, std)

— `window_size` : Taille de la fenêtre d'entrée

— `model_type` : Type d'architecture (MLP/CNN/LSTM)

2.7.2 F-7.2 : Sauvegarde de Contexte

Le contexte complet d'entraînement est sauvegardé en JSON :

```
{
  "payload": {
    "Parametres_temporels": {...},
    "Parametres_choix_reseau_neurones": {...},
    "Parametres_optimisateur": {...},
    "Parametres_entrainement": {...}
  },
  "payload_model": {...},
  "payload_dataset": {...},
  "payload_name_model": {"name": "mon_modele"}
}
```

Listing 2.6 – Structure du contexte

2.7.3 F-7.3 : Chargement de Modèle

L'utilisateur peut charger un modèle précédemment sauvegardé pour :

— Continuer l'entraînement (fine-tuning)

— Effectuer des prédictions sur de nouvelles données

— Comparer les performances avec un nouveau modèle

Chapitre 3

Diagrammes de Cas d'Utilisation

3.1 Vue d'Ensemble des Cas d'Utilisation

Le diagramme de cas d'utilisation présente l'ensemble des fonctionnalités accessibles à l'utilisateur de MLApp. L'acteur principal est l'**Utilisateur** qui interagit avec le système via l'interface graphique.

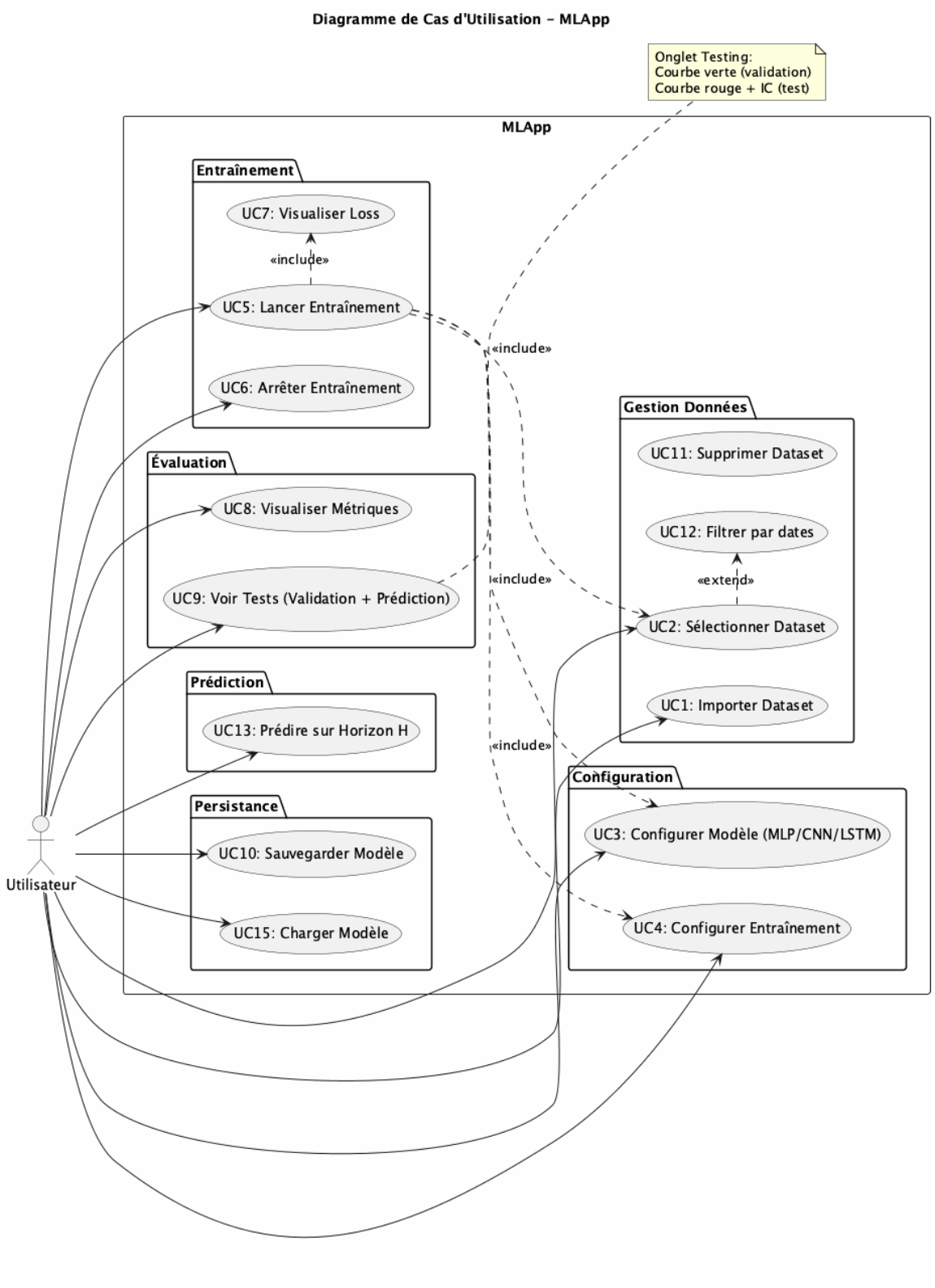


FIGURE 3.1 – Diagramme de cas d'utilisation – MLApp

3.2 Description des Cas d'Utilisation

3.2.1 UC1 : Importer Dataset

— **Acteur** : Utilisateur
— **Description** : Charger un fichier JSON contenant une série temporelle
— **Préconditions** : Fichier JSON valide avec timestamps et values
— **Postconditions** : Dataset stocké sur le Serveur Data

3.2.2 UC2 : Sélectionner Dataset

— **Acteur** : Utilisateur
— **Description** : Choisir un dataset parmi ceux disponibles
— **Extensions** : UC12 (Filtrer par dates)

3.2.3 UC3 : Configurer Modèle

— **Acteur** : Utilisateur
— **Description** : Choisir l'architecture (MLP/CNN/LSTM) et ses hyperparamètres
— **Inclus dans** : UC5 (Lancer Entraînement)

3.2.4 UC4 : Configurer Entraînement

— **Acteur** : Utilisateur
— **Description** : Définir les paramètres d'entraînement (epochs, batch_size, etc.)
— **Inclus dans** : UC5 (Lancer Entraînement)

3.2.5 UC5 : Lancer Entraînement

— **Acteur** : Utilisateur
— **Description** : Démarrer le pipeline d'entraînement complet
— **Inclut** : UC2, UC3, UC4, UC7
— **Préconditions** : Dataset sélectionné, modèle configuré

3.2.6 UC6 : Arrêter Entraînement

— **Acteur** : Utilisateur
— **Description** : Interrompre l'entraînement en cours

3.2.7 UC7 : Visualiser Loss en Temps Réel

— **Description** : Affichage dynamique de la courbe de loss pendant l'entraînement
— **Inclus dans** : UC5

3.2.8 UC8 : Visualiser Métriques

— **Acteur** : Utilisateur

— **Description** : Consulter les métriques (MSE, MAE, RMSE, R^2)

3.2.9 UC9 : Effectuer Prédiction

— **Acteur** : Utilisateur

— **Description** : Visualiser les résultats du pipeline d'évaluation

— **Inclut** : UC5

— **Onglet Testing** : Affiche automatiquement après entraînement :

  — Courbe verte (validation) : `run_validation()` – teacher forcing

  — Courbe rouge + halo (test) : `run_prediction_test()` – one-step + IC 95%

— **Onglet Prediction** : Prédiction libre sur horizon H (données train ou nouvelles)

3.2.10 UC10 : Sauvegarder Modèle

— **Acteur** : Utilisateur

— **Description** : Persister le modèle entraîné sur le Serveur Data

— **Inclut** : UC5 (modèle doit être entraîné)

3.2.11 UC11 : Supprimer Dataset

— **Acteur** : Utilisateur

— **Description** : Supprimer définitivement un dataset

3.2.12 UC15 : Charger Modèle

— **Acteur** : Utilisateur

— **Description** : Charger un modèle précédemment sauvegardé

Chapitre 4

Diagrammes de Classes

Introduction aux Diagrammes de Classes

Les diagrammes de classes présentés dans cette section détaillent la structure interne de chacun des trois sous-systèmes composant l'application **MLApp**. Ils traduisent la conception orientée objet adoptée pour la mise en œuvre du projet.

Chaque diagramme correspond à l'un des tiers de l'architecture globale :

— **L'Interface Utilisateur (UI)** regroupe les classes responsables de la collecte des paramètres, de l'interaction avec l'utilisateur et de la visualisation en temps réel.

— **Le Serveur IA** constitue le cœur du système avec le `TrainingPipeline`, les modules d'entraînement et les stratégies de prédiction.

— **Le Serveur Data** centralise la gestion et la persistance des données avec les services `DatasetService`, `ModelService` et `ContexteService`.

4.1 Diagramme de Classes – Interface Utilisateur

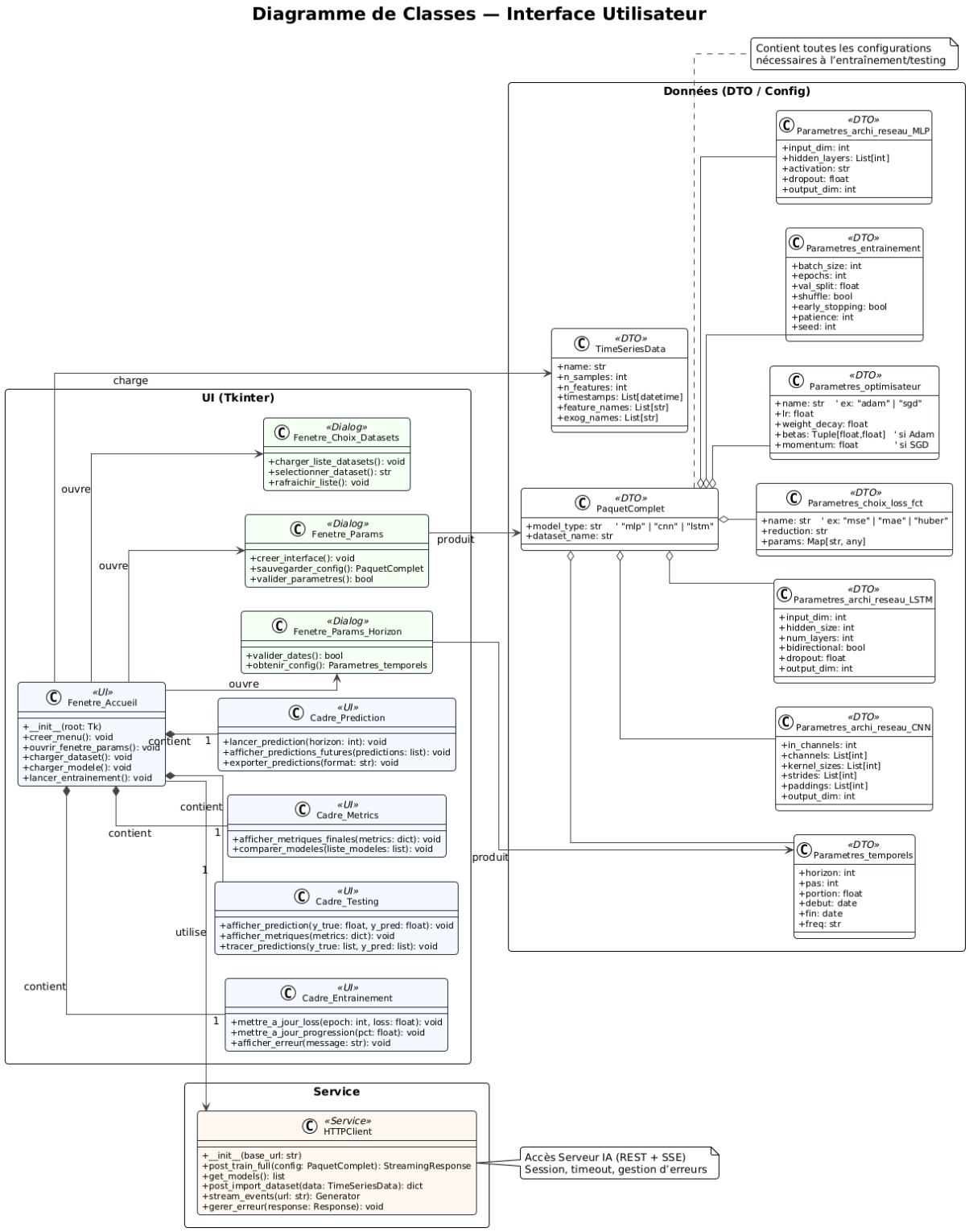


FIGURE 4.1 – Diagramme de classes – Interface Utilisateur (Tkinter/CustomTkinter)

4.1.1 Description des Classes UI

Classe Fenetre_Acceuil

Classe principale de l'application, hérite de `CTk` (CustomTkinter).
Attributs :

— `JSON_Datasets: dict` – Cache local des datasets disponibles
— `stop_training: bool` – Flag pour arrêter l'entraînement
— `Payload: dict` – Configuration complète à envoyer au serveur
— `Results_notebook: CTkTabview` – Onglets de résultats (Training, Testing, Metrics, Prediction)

Méthodes principales :

— `obtenir_datasets()` – Récupère la liste des datasets depuis le serveur
— `EnvoyerConfig()` – Envoie la configuration et démarre l'entraînement
— `annuler_entrainement()` – Interrompt l'entraînement en cours
— `Formatter_JSON_global()` – Formate le payload complet

Classe Cadre_Entrainement

Onglet dédié à la visualisation de l'entraînement.
Responsabilités :

— Affichage de la courbe de loss en temps réel
— Mise à jour dynamique via `add_data_point()`
— Gestion du canvas Matplotlib

Classe Cadre_Testing

Onglet dédié aux résultats de validation.
Responsabilités :

— Affichage comparatif y_{true} vs $\hat{y}$
— Affichage des métriques de validation

Classe Cadre_Prediction

Onglet dédié aux prédictions avec intervalles de confiance.
Responsabilités :

— Affichage des prédictions avec halo de confiance
— Visualisation des bornes haute et basse

Classes de Configuration

— `Gestion_Datasets` – Fenêtre de gestion des datasets (import, suppression, sélection)
— `Fenetre_Params` – Configuration des hyperparamètres du modèle
— `Fenetre_Params_horizon` – Configuration des paramètres temporels

4.2 Diagramme de Classes – Serveur IA

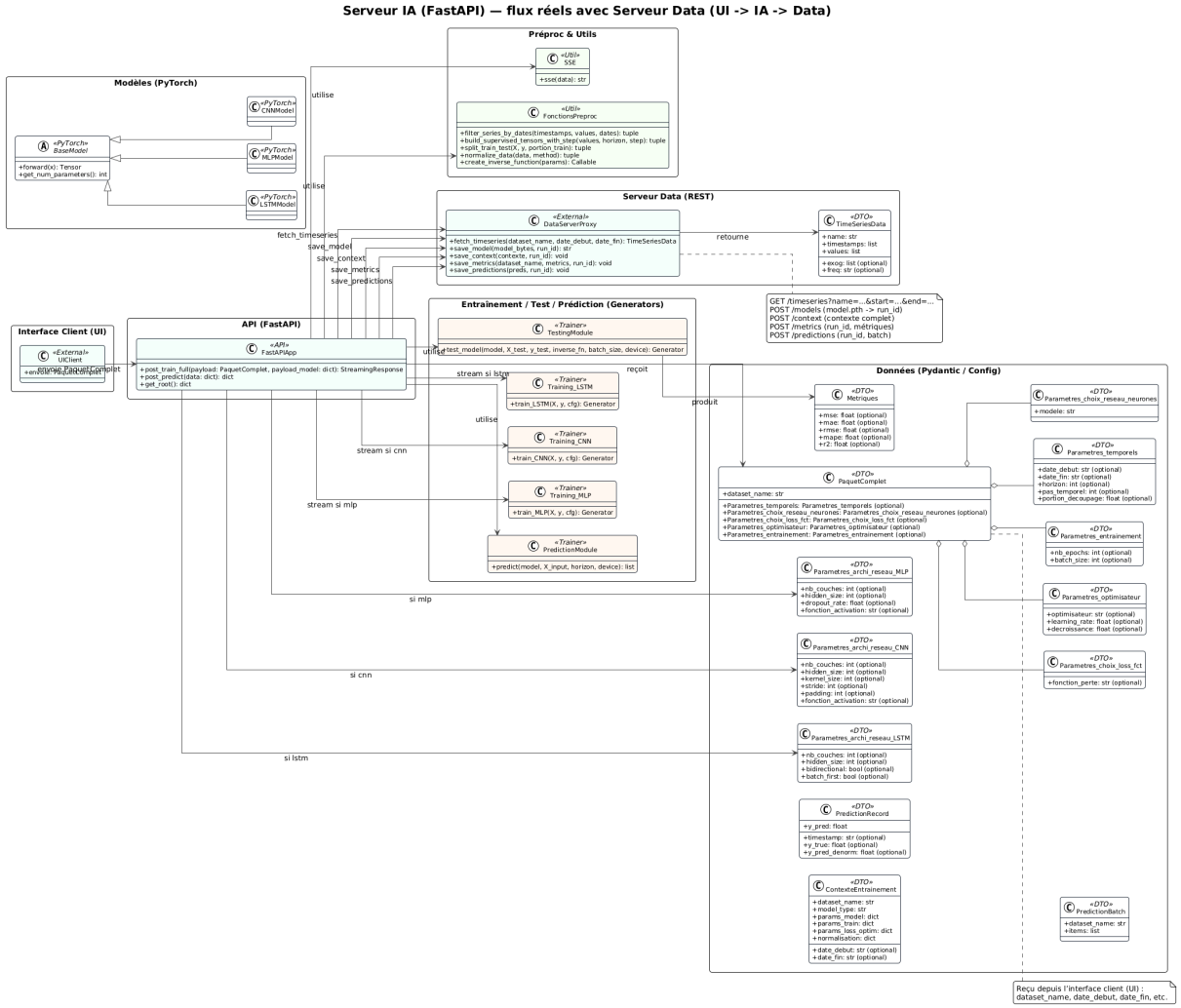


FIGURE 4.2 – Diagramme de classes – Serveur IA (FastAPI + PyTorch)

4.2.1 Description des Classes Serveur IA

Classe FastAPIApp (Controller)

Point d'entrée de l'API REST, expose les endpoints HTTP.
Endpoints :

— `POST /train_full` – Entraînement complet avec streaming SSE

— `POST /stop_training` – Interrompre l'entraînement

— `POST /datasets/info_all` – Proxy vers Serveur Data (liste datasets)

— `POST /datasets/fetch_dataset` – Proxy vers Serveur Data (récupérer dataset)

— `POST /datasets/add_dataset` – Proxy vers Serveur Data (ajouter dataset)

— `GET /` – Health check

Classe TrainingPipeline

Classe centrale orchestrant le pipeline d'entraînement en 3 phases.
Attributs :

— `cfg: PaquetComplet` – Configuration complète
— `payload_model: dict` – Paramètres spécifiques au modèle
— `device: torch.device` – CPU/CUDA/MPS
— `X_train, y_train, X_val, y_val, X_test, y_test: Tensor` – Données splitées
— `norm_params: dict` – Paramètres de normalisation (mean, std)
— `inverse_fn: Callable` – Fonction de dénormalisation
— `model_trained: nn.Module` – Modèle entraîné
— `residual_std: float` – Écart-type des résidus (pour le halo)
— `stop_flag: bool` – Flag d'arrêt

Méthodes :

— `load_data()` – Charge et valide les données
— `setup_model_config()` – Configure le modèle selon le type
— `preprocess_data()` – Crée les tenseurs supervisés
— `normalize_data()` – Normalise les données (z-score)
— `split_data_three_way()` – Split 80/10/10
— `run_training()` – Générateur pour la phase d'entraînement
— `run_validation()` – Générateur pour la phase de validation
— `run_prediction_test()` – Générateur pour la phase de test prédictif
— `execute_full_pipeline()` – Exécute le pipeline complet

Modules d'Entraînement

Générateurs Python pour chaque type de modèle :

— `train_MLP` – Entraînement Perceptron Multi-Couches
— `train_CNN` – Entraînement Réseau Convolutif
— `train_LSTM` – Entraînement Long Short-Term Memory

Chaque générateur `yield` des événements SSE à chaque époque.

Modules de Test et Prédiction

— `test_model_validation` – Évaluation sur l'ensemble de validation
— `predict_multistep` – Prédiction multi-pas avec stratégies
— `PredictionStrategy` – Enum des stratégies (ONE_STEP, RECALIBRATION, RECURSIVE, DIRECT)
— `PredictionConfig` – Configuration de la prédiction (stratégie, recalib_every, etc.)

4.3 Diagramme de Classes – Serveur IA (Simplifié)

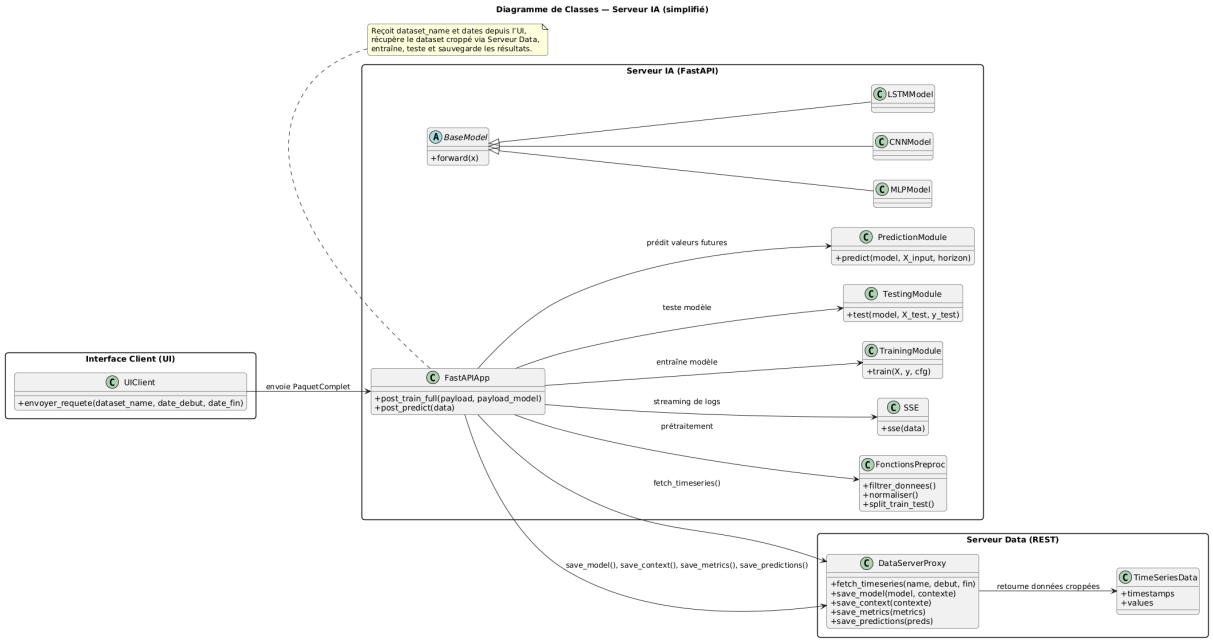


FIGURE 4.3 – Diagramme de classes simplifié – Flux UI → IA → Data

Ce diagramme simplifié illustre le flux global de données entre les trois composants principaux, en mettant en évidence les classes clés et leurs interactions.

4.4 Diagramme de Classes – Serveur Data

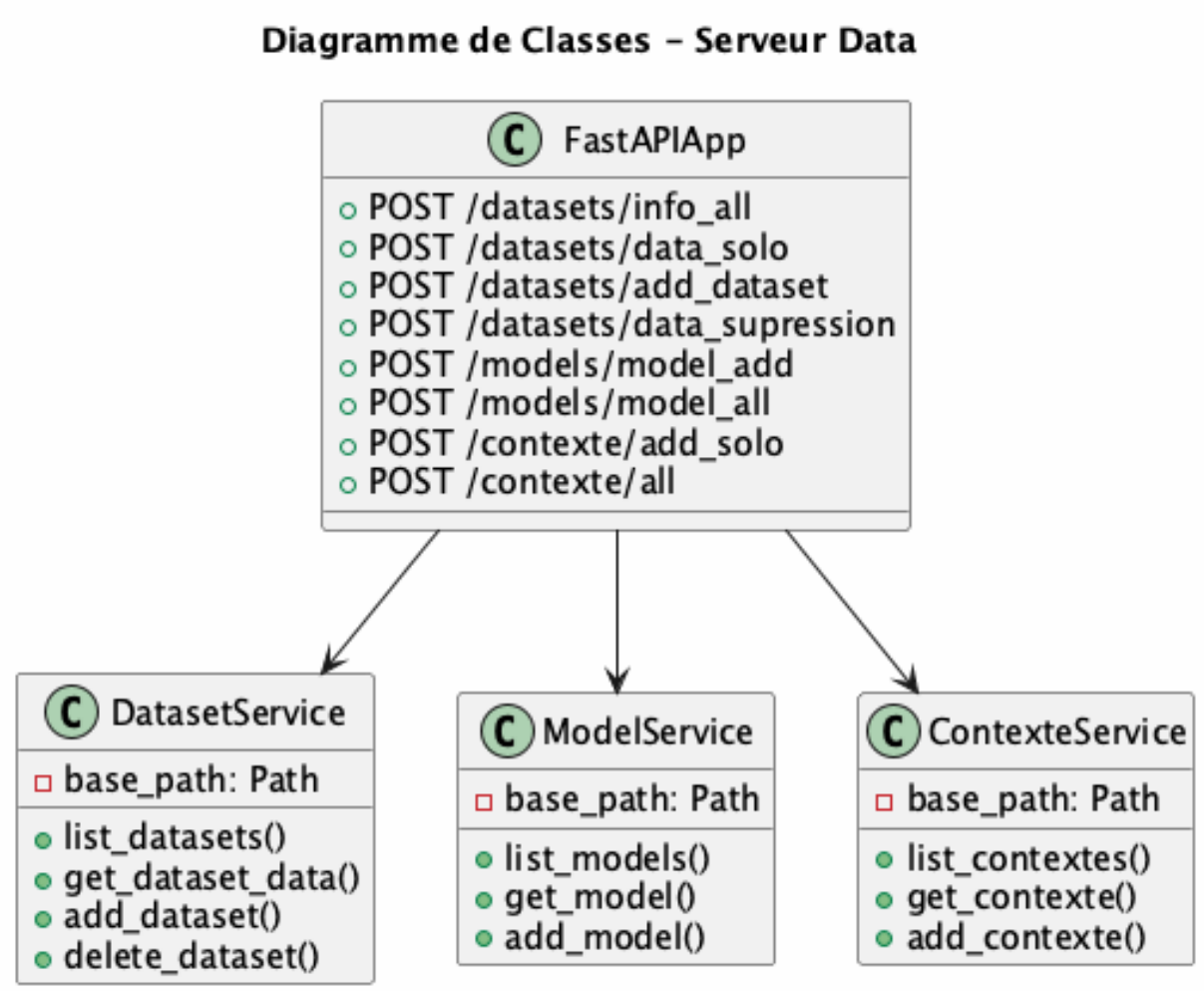


FIGURE 4.4 – Diagramme de classes – Serveur Data (FastAPI + JSON)

4.4.1 Description des Classes Serveur Data

Classe FastAPIApp (Controller)

Point d'entrée de l'API REST du Serveur Data.

Endpoints Datasets :

— `POST /datasets/info_all` – Liste tous les datasets avec métadonnées

— `POST /datasets/data_solo` – Récupère un dataset filtré

— `POST /datasets/add_dataset` – Ajoute un nouveau dataset

— `POST /datasets/data_supression` – Supprime un dataset

Endpoints Modèles :

— `POST /models/model_all` – Liste tous les modèles .pth

— `POST /models/model_add` – Ajoute un modèle (Base64)

— `POST /models/model_delete` – Supprime un modèle

Endpoints Contextes :

— `POST /contexte/add_solo` – Sauvegarde un contexte d'entraînement

— `POST /contexte/obtenir_solo` – Récupère un contexte

Services de Gestion

— `DatasetService` – CRUD sur les fichiers JSON des datasets

— `ModelService` – CRUD sur les fichiers .pth des modèles

— `ContexteService` – CRUD sur les fichiers JSON des contextes

Stockage Fichiers

Structure des dossiers sur le Serveur Data :

```
SERVEUR_DATA/
|-- datasets/          # Fichiers JSON des series temporelles
|   |-- EURO.json
|   |-- CACAO.json
|   +-- marees.json
|-- models/            # Fichiers .pth (PyTorch state_dict)
|   |-- mon_modele.pth
|   +-- mon_modele2.pth
+-- contextes/         # Fichiers JSON des contextes d'entrainement
    |-- contexte_model_mlp.json
    +-- contexte_model_lstm.json
```

4.5 Classes DTOs Pydantic

Les Data Transfer Objects (DTOs) assurent la validation des données échangées entre les composants.

4.5.1 DTOs Serveur IA

TABLE 4.1 – Classes Pydantic – Serveur IA

Classe	Description
`TimeSeriesData`	timestamps : List[datetime], values : List[Optional[float]]
`PaquetComplet`	Agrège tous les paramètres de configuration
`Parametres_temporels`	horizon, window_size, portion_decoupage
`Parametres_choix_reseau_neurones`	modele (MLP/LSTM/GRU/CNN)
`Parametres_choix_loss_fct`	fonction_perte (MSE/MAE/Huber)
`Parametres_optimisateur`	optimisateur, learning_rate, scheduler
`Parametres_entrainement`	nb_epochs, batch_size, clip_gradient
`Parametres_archi_reseau_MLP`	nb_couches, hidden_size, dropout_rate, activation
`Parametres_archi_reseau_CNN`	+ kernel_size, stride, padding
`Parametres_archi_reseau_LSTM`	+ bidirectional, batch_first

4.5.2 DTOs Serveur Data

TABLE 4.2 – Classes Pydantic – Serveur Data

Classe	Description
`ChoixDatasetRequest`	message : str
`ChoixDatasetRequest2`	name, dates : List[str], pas_temporel : int
`AddDatasetPacket`	payload_name, payload_dataset_add : TimeSeriesData
`newModelRequest`	name, data (Base64 encoded)
`DeleteModelRequest`	name
`PaquetComplet2`	payload, payload_model, payload_dataset, payload_name_model
`ChoixContexteRequest`	name

Chapitre 5

Diagrammes de Séquence

Les diagrammes de séquence illustrent le comportement dynamique du système lors des différentes interactions. Ils montrent la séquence des messages échangés entre les composants au fil du temps.

Génération des Diagrammes pour Overleaf

Tous les diagrammes sont générés par `uml.py`. Pour compiler sur Overleaf :

1. Générer les PNG localement : `python uml.py -output .`
2. Uploader `main.tex` + tous les `*.png` sur Overleaf
3. Compiler avec pdfLaTeX

5.1 DS1 : Configuration du Modèle

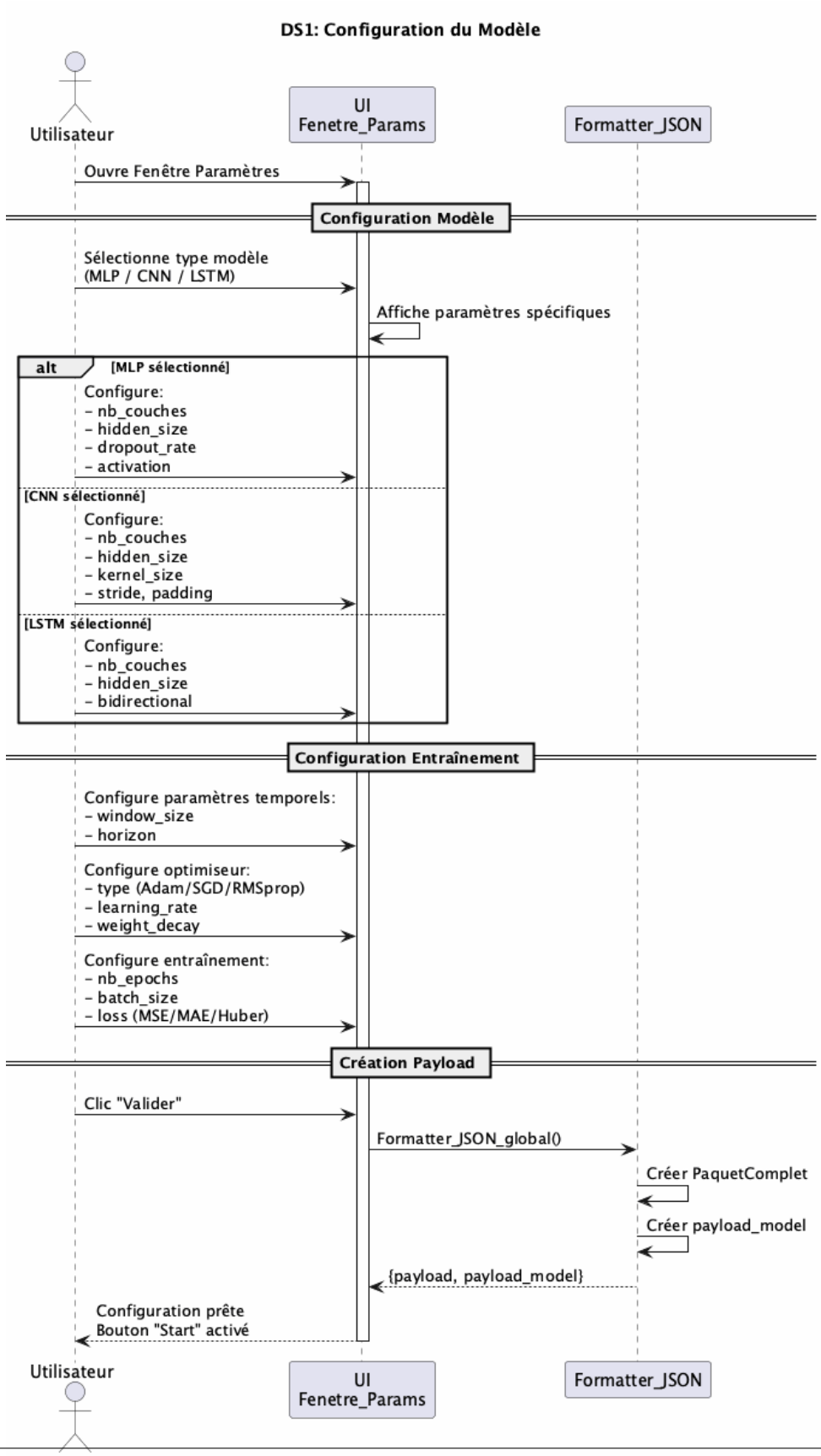


FIGURE 5.1 – Configuration du modèle et des paramètres d'entraînement

Description : L'utilisateur configure les paramètres via l'interface `Fenetre_Params` :

1. **Sélection du type de modèle :** MLP, CNN ou LSTM
2. **Configuration des paramètres spécifiques :**
   — MLP : `nb_couches`, `hidden_size`, `dropout_rate`, `activation`
   — CNN : `nb_couches`, `hidden_size`, `kernel_size`, `stride`, `padding`
   — LSTM : `nb_couches`, `hidden_size`, `bidirectional`
3. **Configuration de l'entraînement :**
   — Paramètres temporels : `window_size`, `horizon`
   — Optimiseur : type (Adam/SGD/RMSprop), `learning_rate`, `weight_decay`
   — Entraînement : `nb_epochs`, `batch_size`, `loss` (MSE/MAE/Huber)
4. **Création du payload :** `Formatter_JSON_global()` crée `PaquetComplet`

5.2 DS2 : Lancement de l'Entraînement (Phase 1)

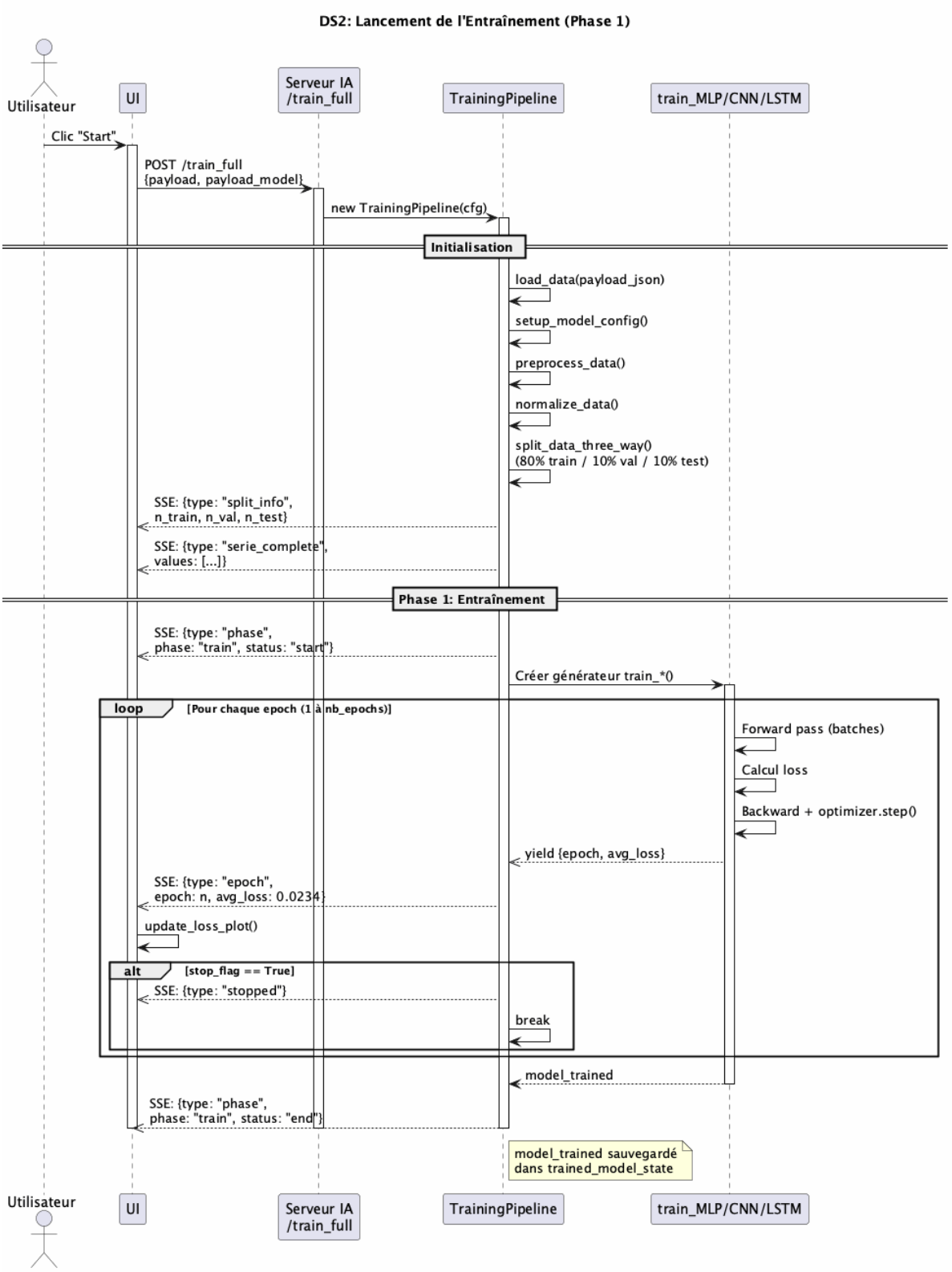


FIGURE 5.2 – Lancement de l'entraînement – Phase 1 du pipeline (80% des données)

Description de la Phase 1 (Entraînement) :

1. **Initialisation du pipeline :**
   — `load_data()` – Charger les données depuis `payload_json`
   — `setup_model_config()` – Configurer hyperparamètres
   — `preprocess_data()` – Créer tenseurs (X, y)
   — `normalize_data()` – Normalisation z-score
   — `split_data_three_way()` – Split 80%/10%/10%
2. **Événements SSE d'initialisation :**
   — `{"type": "split_info", "n_train": 4000, "n_val": 500, "n_test": 500}`
   — `{"type": "serie_complete", "values": [...]}`
3. **Boucle d'entraînement :**
   — Pour chaque epoch : forward pass → calcul loss → backward → optimizer.step()
   — SSE : `{"type": "epoch", "epoch": n, "avg_loss": 0.0234}`
   — Vérification du `stop_flag` à chaque epoch

5.3 DS3 : Phase de Test (Validation + Test Prédictif)

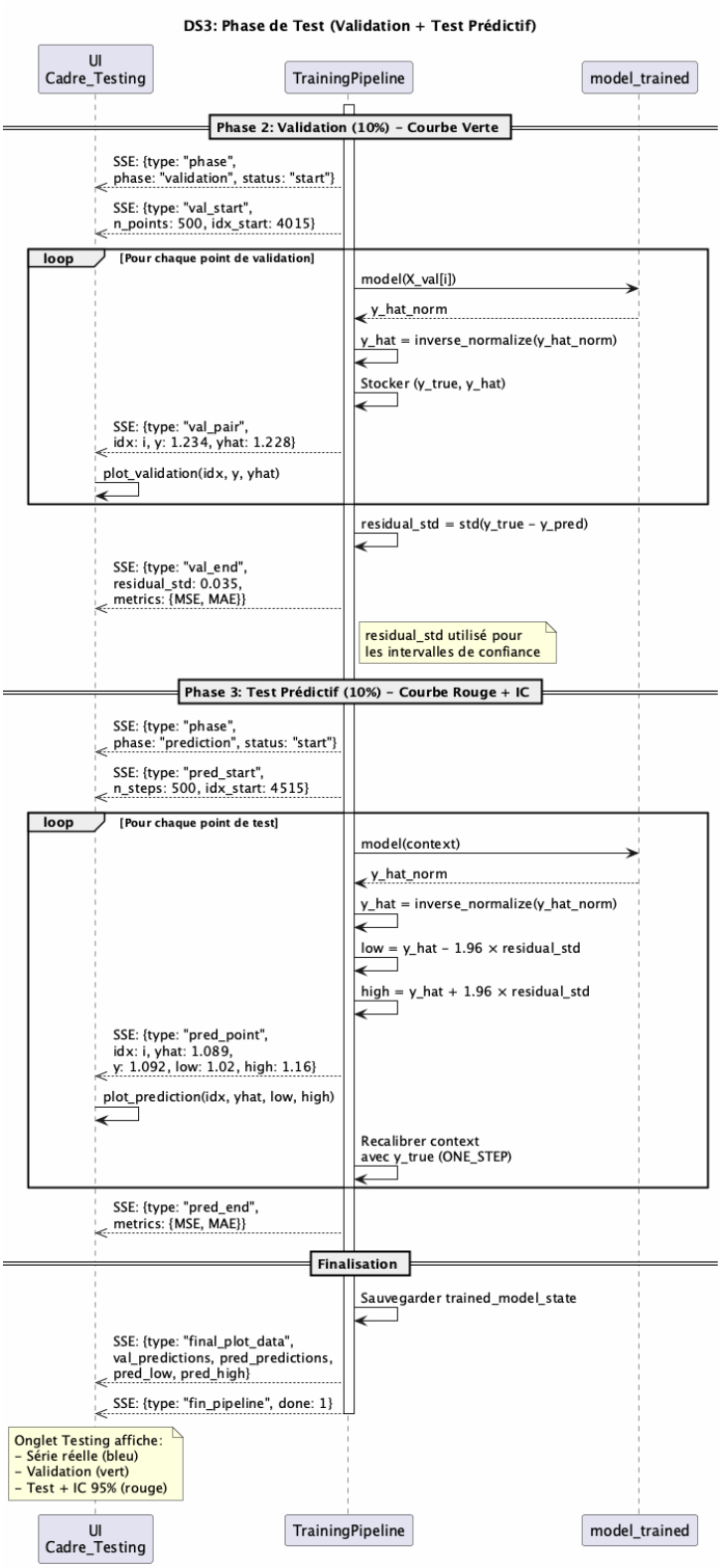


FIGURE 5.3 – Phases 2 et 3 – Validation (courbe verte) et Test Prédictif (courbe rouge + IC)

Description des Phases 2 et 3 :

1. **Phase 2 – Validation (10%) – Courbe Verte :**
   - Mode : Teacher forcing (recalibration immédiate sur y_{true})
   - SSE : `{"type": "val_pair", "idx": i, "y": 1.234, "yhat": 1.228}`
   - Calcul de `residual_std = std(y_true - y_pred)`
   - SSE : `{"type": "val_end", "residual_std": 0.035, "metrics": {...}}`
2. **Phase 3 – Test Prédictif (10%) – Courbe Rouge + IC 95% :**
   - Mode : ONE_STEP avec recalibration sur y_{true}
   - Intervalle de confiance : $IC = \hat{y} \pm 1.96 \times \sigma_{residual}$
   - SSE : `{"type": "pred_point", "idx": i, "yhat": 1.089, "y": 1.092, "low": 1.02, "high": 1.16}`

Interprétation des Courbes (Onglet Testing)

- **Courbe verte proche du bleu** ⇒ Le modèle prédit bien avec teacher forcing
- **Courbe rouge proche du bleu** ⇒ Le modèle prédit bien en mode one-step
- **Halo rouge contient le bleu** ⇒ L'IC 95% est bien calibré

5.4 DS4 : Calcul et Affichage des Métriques

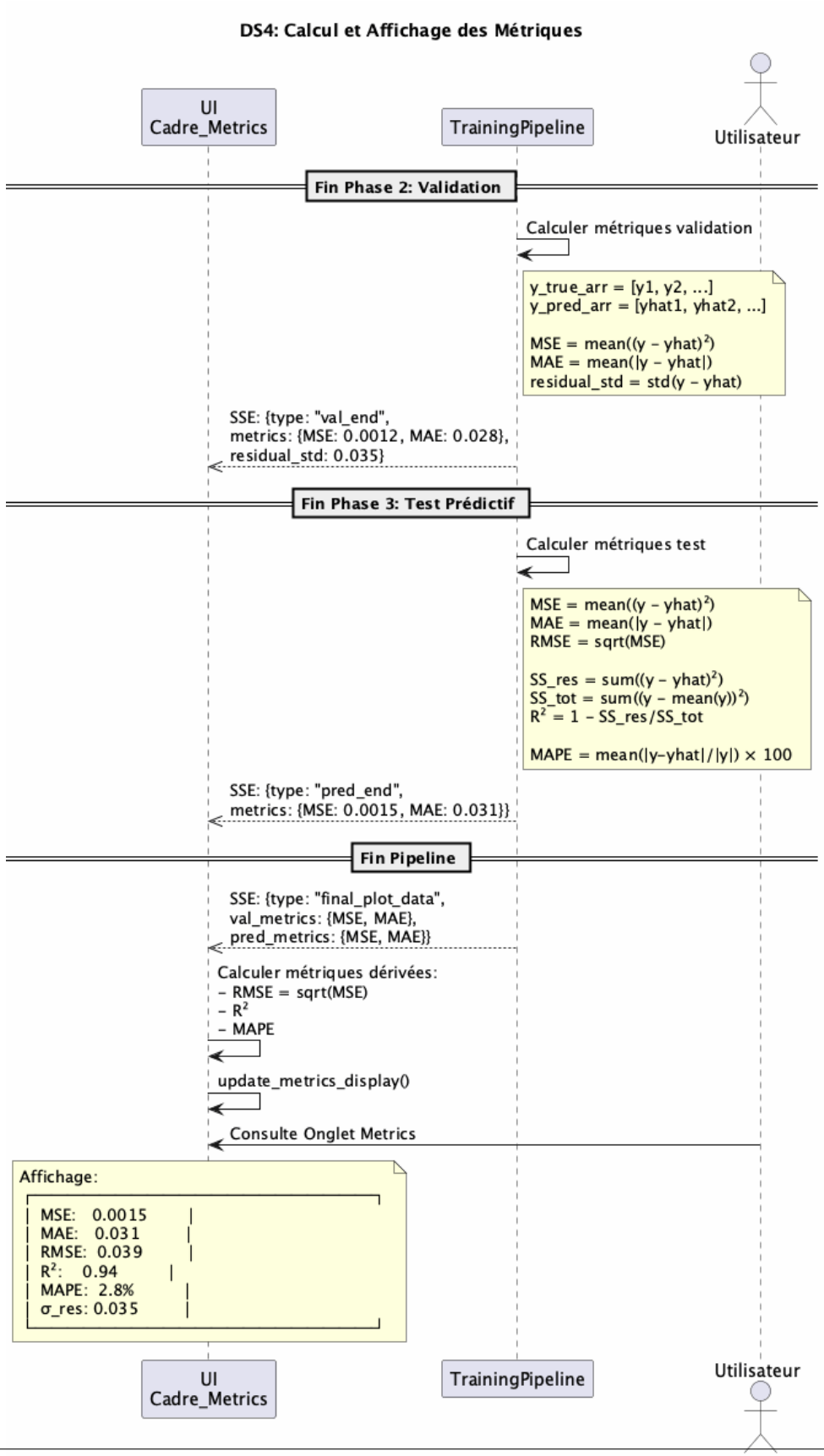


FIGURE 5.4 – Calcul et affichage des métriques dans l'onglet Metrics

Métriques calculées :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.1}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.2}$$

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{5.3}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{5.4}$$

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{5.5}$$

5.5 DS5 : Prédiction Future (Onglet Prediction)

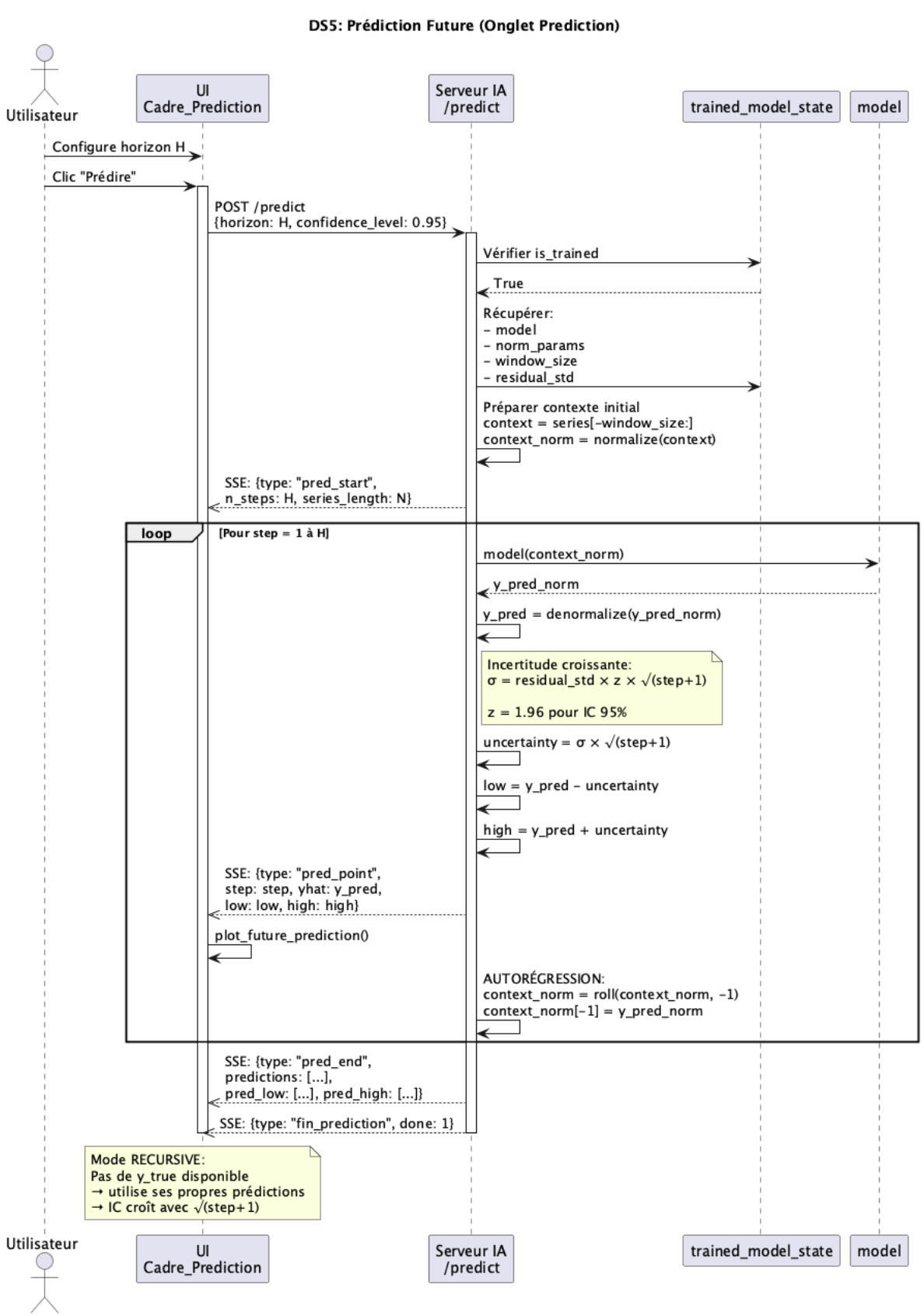


FIGURE 5.5 – Prédiction future sur horizon H (mode RECURSIVE)

Description de l'endpoint `POST /predict` :

1. Configuration de l'horizon H et niveau de confiance
2. Vérification que `trained_model_state.is_trained = True`
3. Préparation du contexte initial (derniers `window_size` points normalisés)
4. **Boucle de prédiction autorégressive :**
   - `y_pred = model(context)`
   - Incertitude croissante : $\sigma = \sigma_{residual} \times z \times \sqrt{step + 1}$
   - SSE : `{"type": "pred_point", "step": step, "yhat": ..., "low": ..., "high": ...}`
   - **Autorégression :** `context[-1] = y_pred_norm`

Mode RECURSIVE

Dans l'onglet Prediction, il n'y a pas de y_{true} disponible (prédiction dans le futur). Le modèle utilise ses propres prédictions pour le contexte suivant, ce qui accumule les erreurs. L'IC croît avec $\sqrt{step + 1}$.

5.6 DS6 : Import Dataset

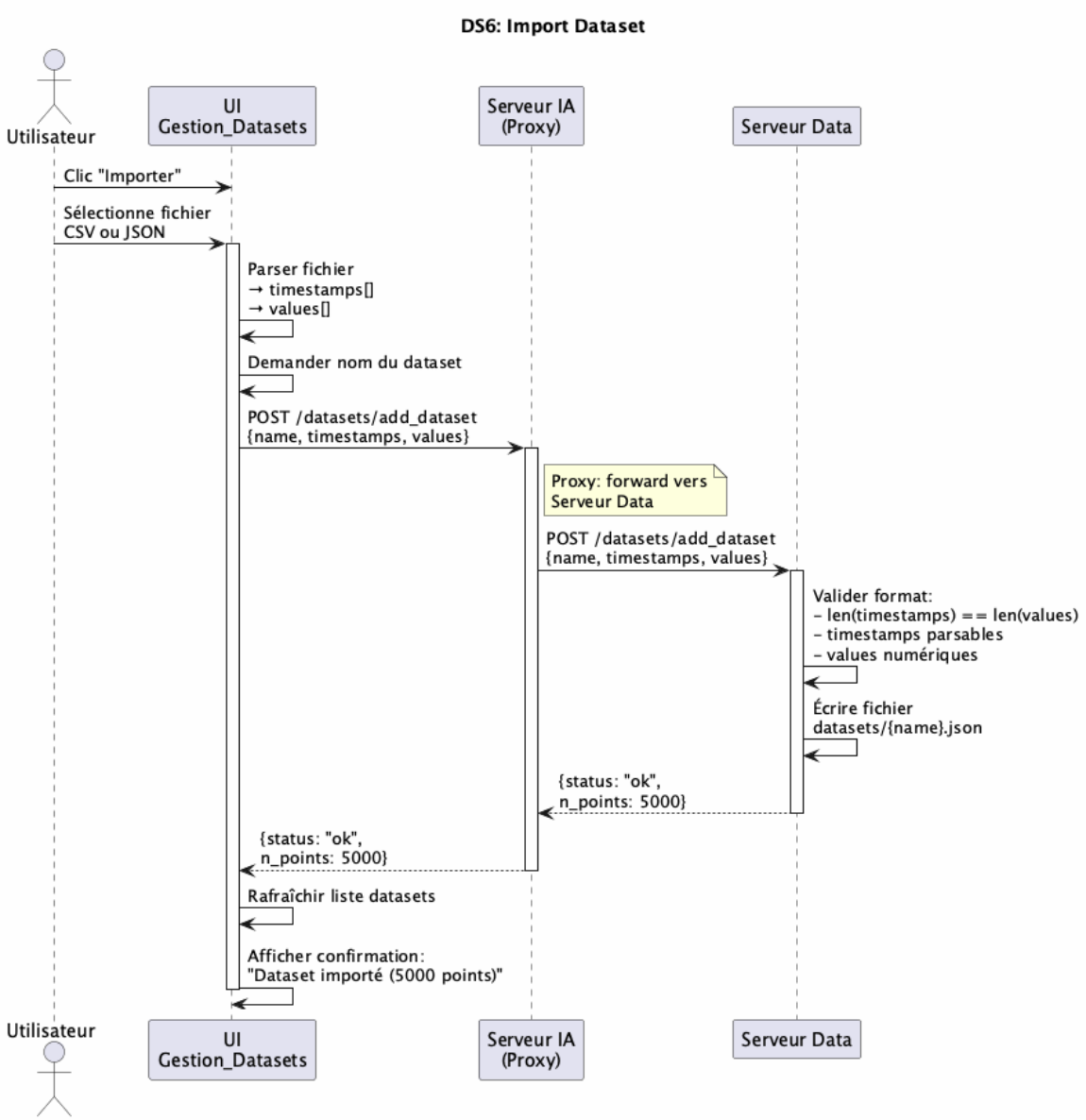


FIGURE 5.6 – Import d'un dataset via le proxy du Serveur IA

Description :

1. L'utilisateur sélectionne un fichier CSV ou JSON
2. L'UI parse le fichier et extrait `timestamps` et `values`
3. `POST /datasets/add_dataset` via proxy IA → Serveur Data
4. Le Serveur Data valide et écrit `datasets/{name}.json`

5.7 DS7 : Chargement Dataset (Fetch)

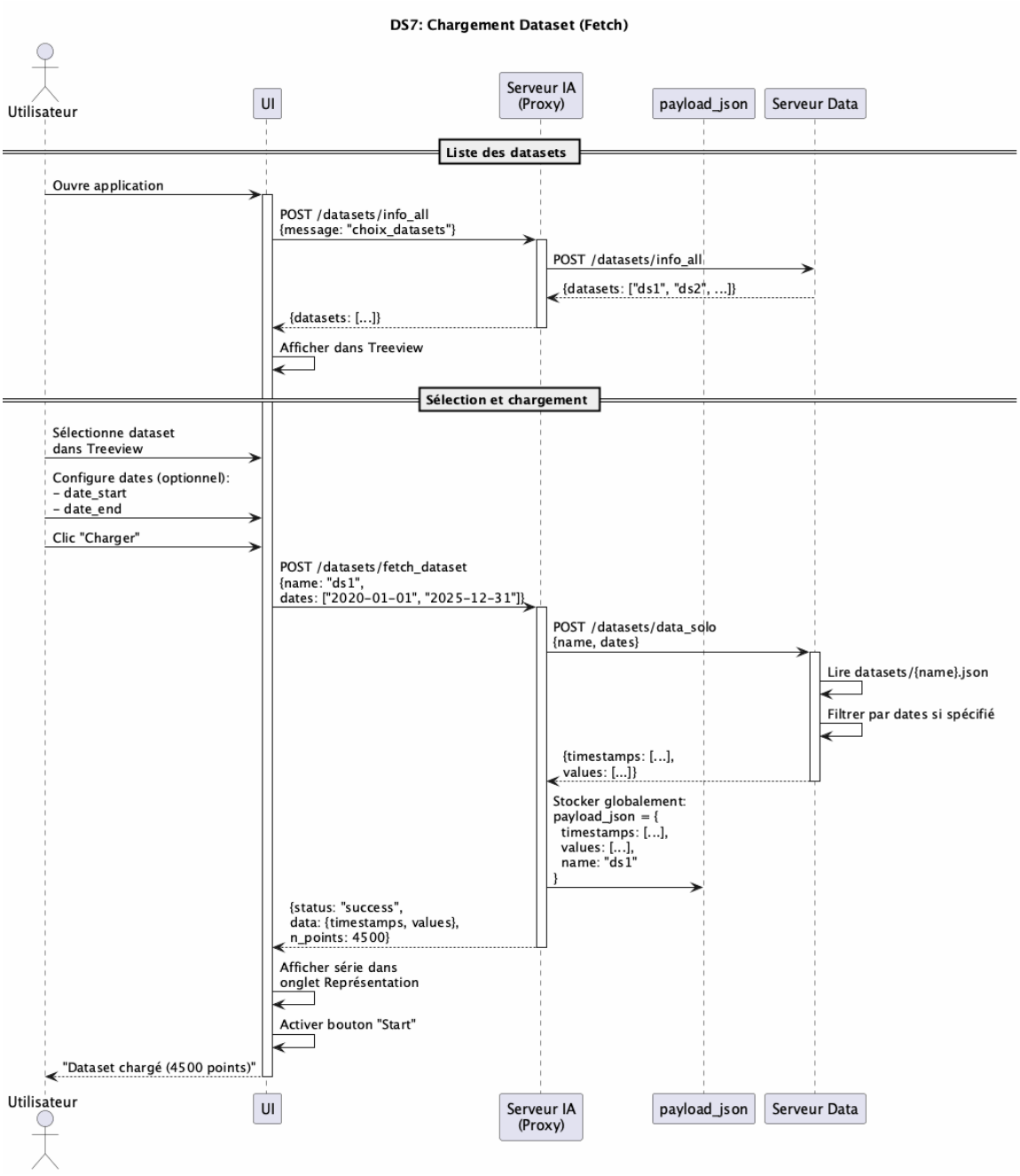


FIGURE 5.7 – Sélection et chargement d'un dataset avec filtrage par dates

Description :

1. Liste des datasets via `POST /datasets/info_all`
2. Sélection dans le Treeview + configuration dates (optionnel)
3. `POST /datasets/fetch_dataset` → Serveur Data
4. Données stockées dans `payload_json` (variable globale)
5. Affichage dans l'onglet Représentation

5.8 DS8 : Sauvegarde du Modèle

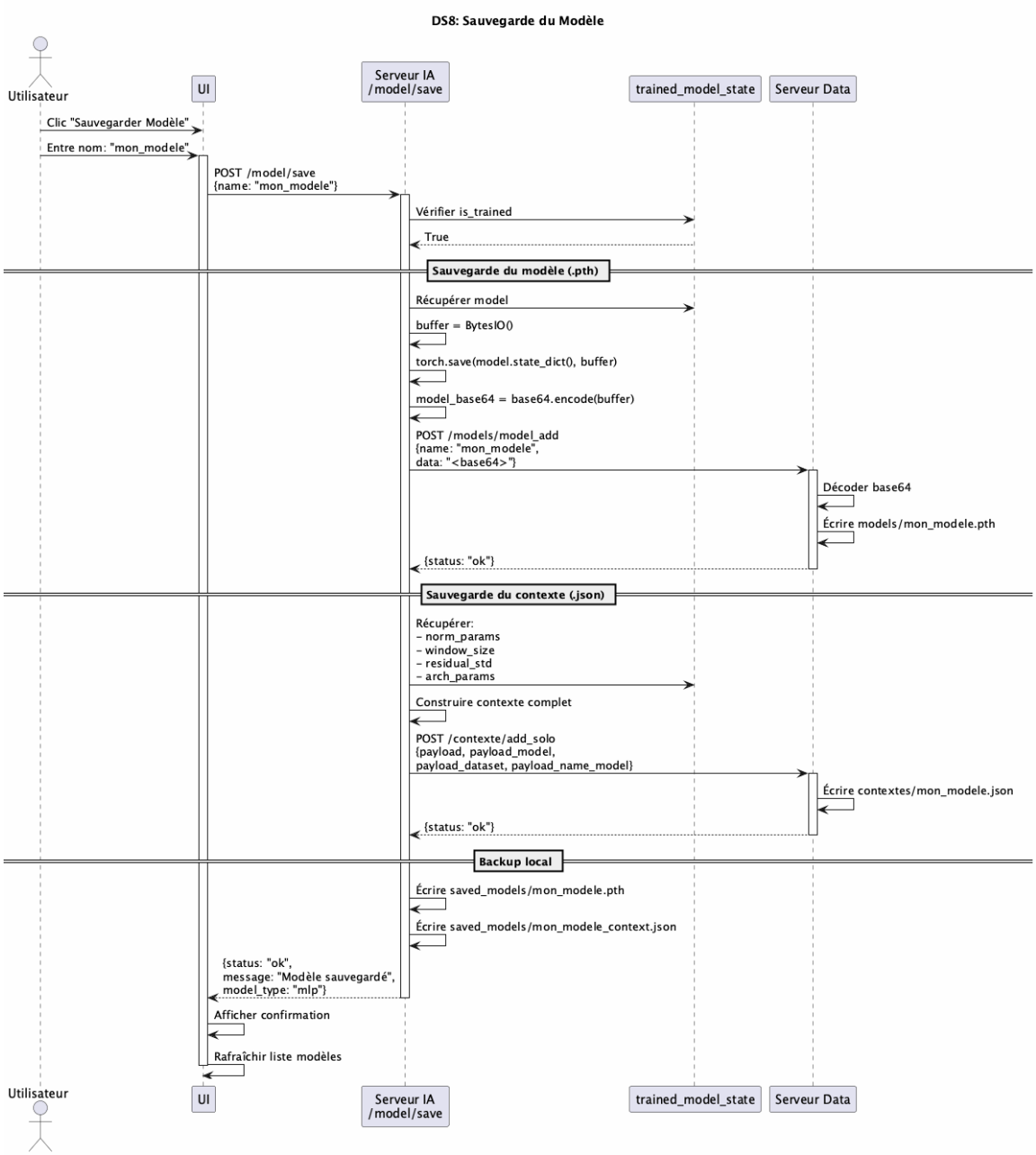


FIGURE 5.8 – Sauvegarde du modèle et du contexte vers le Serveur Data

Description de `POST /model/save` :

1. Sérialisation du `state_dict` en Base64
2. `POST /models/model_add` avec le modèle encodé
3. Sauvegarde du contexte complet (norm_params, window_size, residual_std, arch_params)
4. `POST /contexte/add_solo` avec le contexte
5. Backup local dans `./saved_models/`

5.9 DS9 : Chargement du Modèle

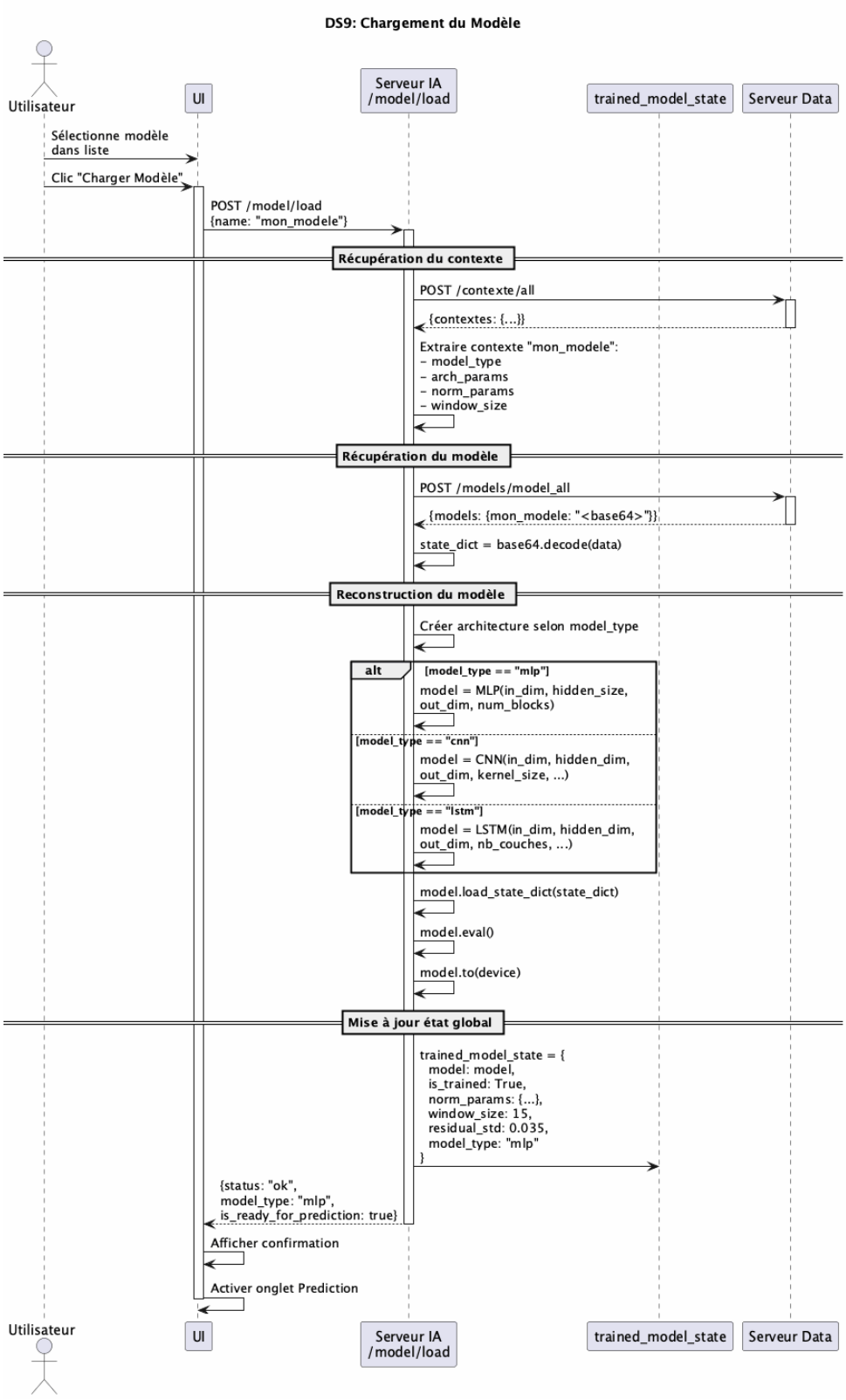


FIGURE 5.9 – Chargement d'un modèle sauvegardé depuis le Serveur Data

Description de `POST /model/load` :

1. Récupération du contexte via `POST /contexte/all`
2. Récupération du modèle encodé via `POST /models/model_all`
3. Décodage Base64 → `state_dict`
4. Reconstruction de l'architecture selon `model_type`
5. `model.load_state_dict()` + `model.eval()`
6. Mise à jour de `trained_model_state`

5.10 DS10 : Suppression Dataset

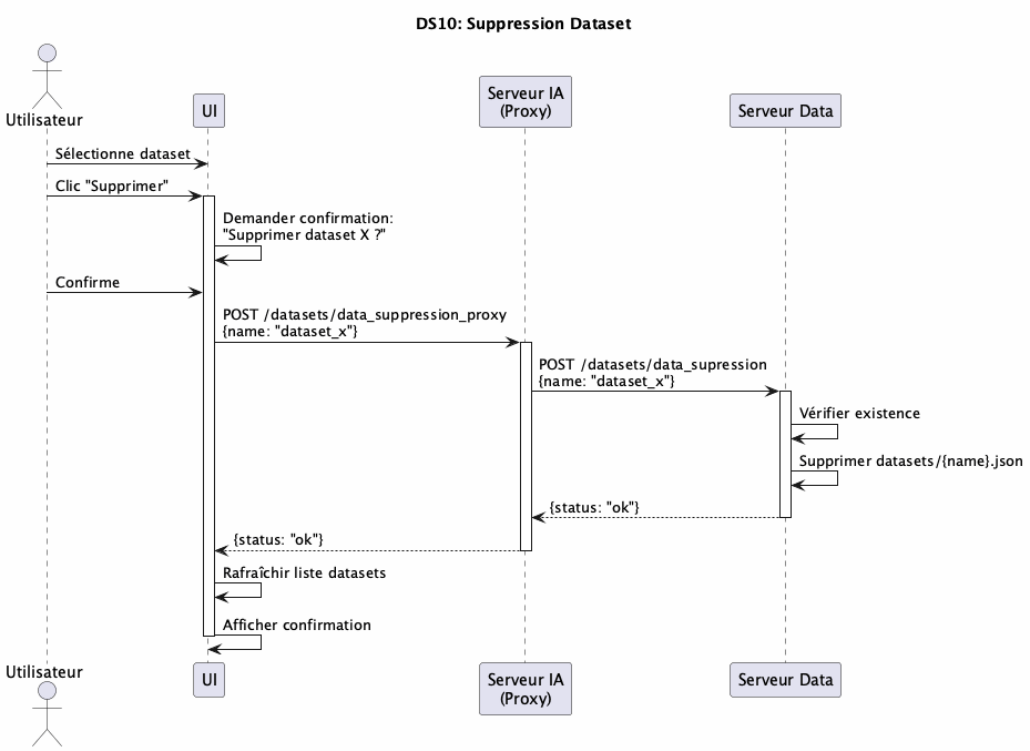


FIGURE 5.10 – Suppression d'un dataset

Pattern Proxy

Toutes les opérations sur les datasets passent par le Serveur IA (proxy) :

— `/datasets/info_all` → Liste des datasets

— `/datasets/fetch_dataset` → Récupération avec filtrage

— `/datasets/add_dataset` → Import

— `/datasets/data_suppression_proxy` → Suppression

5.11 Diagramme d'Activité : Pipeline Complet

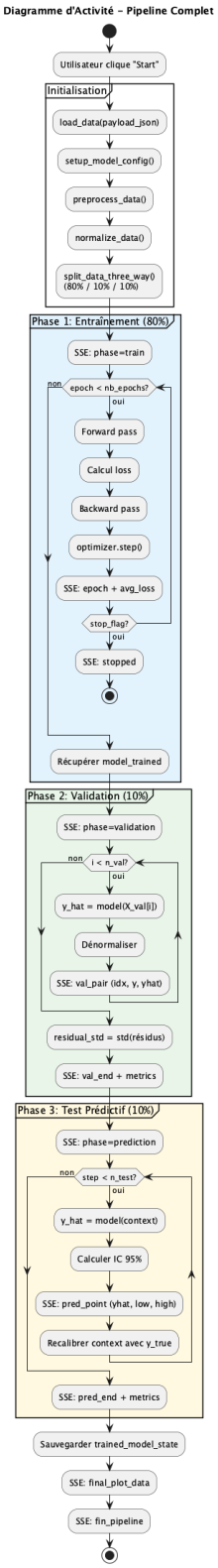


FIGURE 5.11 – Diagramme d'activité – Flux du pipeline en 3 phases

5.12 Diagramme de Déploiement

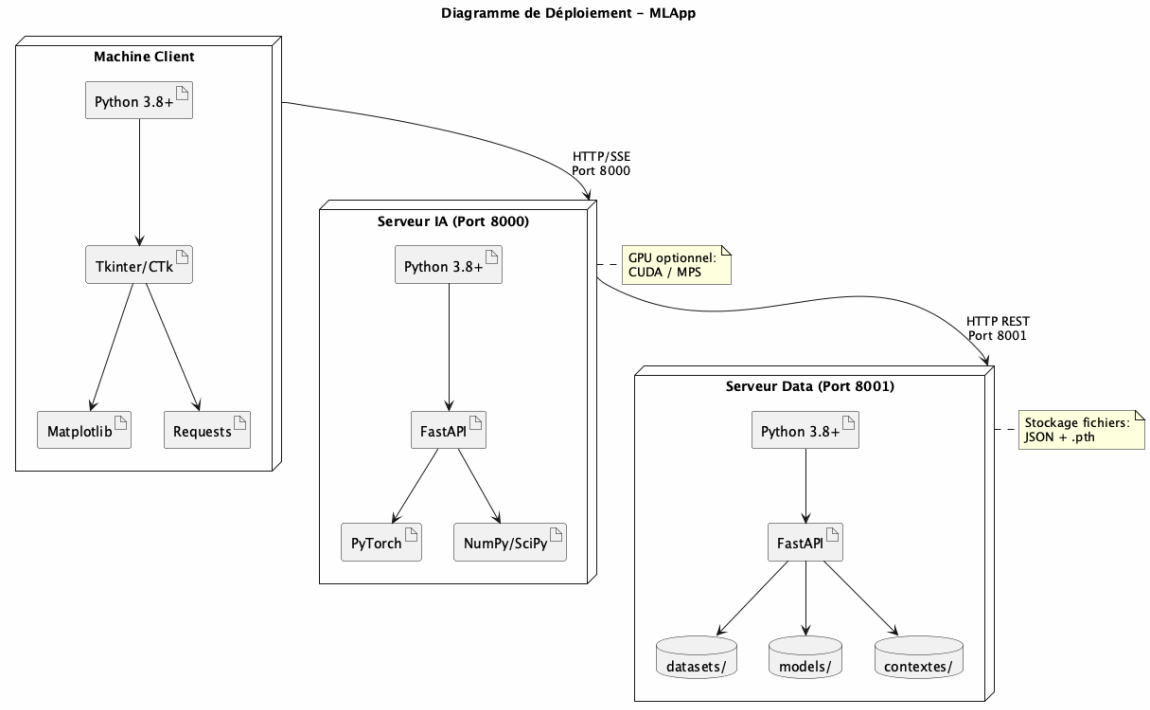


FIGURE 5.12 – Diagramme de déploiement – Architecture réseau Client / IA / Data

Architecture de déploiement :

— **Client :** Python 3.8+, Tkinter/CTk, Matplotlib, Requests

— **Serveur IA (Port 8000) :** FastAPI, PyTorch, NumPy/SciPy, GPU optionnel (CUDA/MPS)

— **Serveur Data (Port 8001) :** FastAPI, stockage fichiers JSON et .pth

Chapitre 6

Spécifications Non-Fonctionnelles

6.1 Performance

— Le streaming SSE doit permettre une mise à jour de l'interface à chaque époque sans lag perceptible
— Le temps de réponse des endpoints proxy doit être inférieur à 500ms
— L'interface doit rester responsive pendant l'entraînement grâce au threading
— Support CPU + GPU (CUDA) et Apple Silicon (MPS) pour accélérer l'entraînement

6.2 Fiabilité

— Les données doivent être validées par Pydantic avant tout traitement
— Les erreurs d'entraînement doivent être propagées proprement à l'interface
— La fonction d'arrêt d'entraînement doit être réactive (vérification à chaque époque)
— Gestion des NaN dans les séries temporelles (forward/backward fill)

6.3 Maintenabilité

— Architecture modulaire avec séparation des responsabilités (trois-tiers)
— Code documenté avec docstrings Python
— Tests unitaires pour les fonctions critiques
— Utilisation de générateurs Python pour un code lisible et efficace

6.4 Portabilité

— Compatible Python 3.8+
— Indépendant du système d'exploitation (Windows, macOS, Linux)
— GPU optionnel via PyTorch CUDA/MPS
— Communication REST standard entre composants

6.5 Sécurité

— Validation stricte des entrées via Pydantic
— Pas d'exécution de code arbitraire
— Communication locale uniquement (pas d'exposition Internet par défaut)

BONUS : Interface Web JavaScript

Contexte et Motivation

L'interface Python Tkinter/CustomTkinter, bien que fonctionnelle, présentait une limitation majeure : la difficulté de visualiser l'architecture interne des réseaux de neurones de manière interactive et pédagogique.

Face à cette contrainte technique, nous avons décidé de **refaire l'interface frontend en JavaScript**, en utilisant l'assistance de l'intelligence artificielle pour accélérer le développement (sachant que le backend était déjà fait). Cette nouvelle interface conserve **exactement les mêmes fonctionnalités** que l'interface Python, tout en ajoutant une dimension pédagogique essentielle : la **visualisation en temps réel de l'architecture du réseau**.

Pourquoi JavaScript ?

- **Visualisation avancée :** Les bibliothèques JavaScript (D3.js, Three.js, Canvas API) offrent des capacités de rendu graphique supérieures
- **Interactivité :** Manipulation intuitive des éléments visuels (zoom, pan, survol)
- **Portabilité :** Exécution dans n'importe quel navigateur moderne, sans installation
- **Écosystème riche :** React, Tailwind CSS, bibliothèques de visualisation de réseaux de neurones

Captures d'Écran de l'Interface MLApp

Onglet Architecture – Visualisation du Réseau

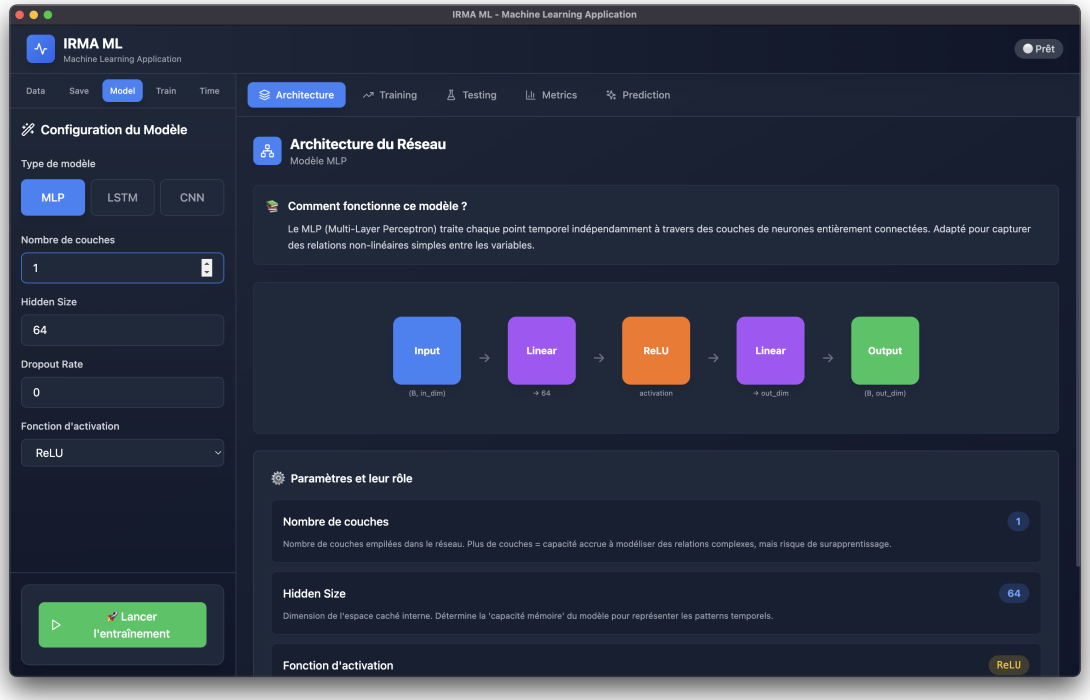


FIGURE 6.1 – Onglet Architecture – Visualisation interactive du pipeline MLP

L'onglet **Architecture** constitue l'apport majeur de l'interface JavaScript. Il affiche :

— **Le pipeline du modèle :** Input → Linear → ReLU → Linear → Output avec les dimensions

— **Explication pédagogique :** Description du fonctionnement du modèle MLP

— **Paramètres et leur rôle :** Explication de chaque paramètre (nombre de couches, hidden size, fonction d'activation)

Onglet Training – Évolution de la Loss

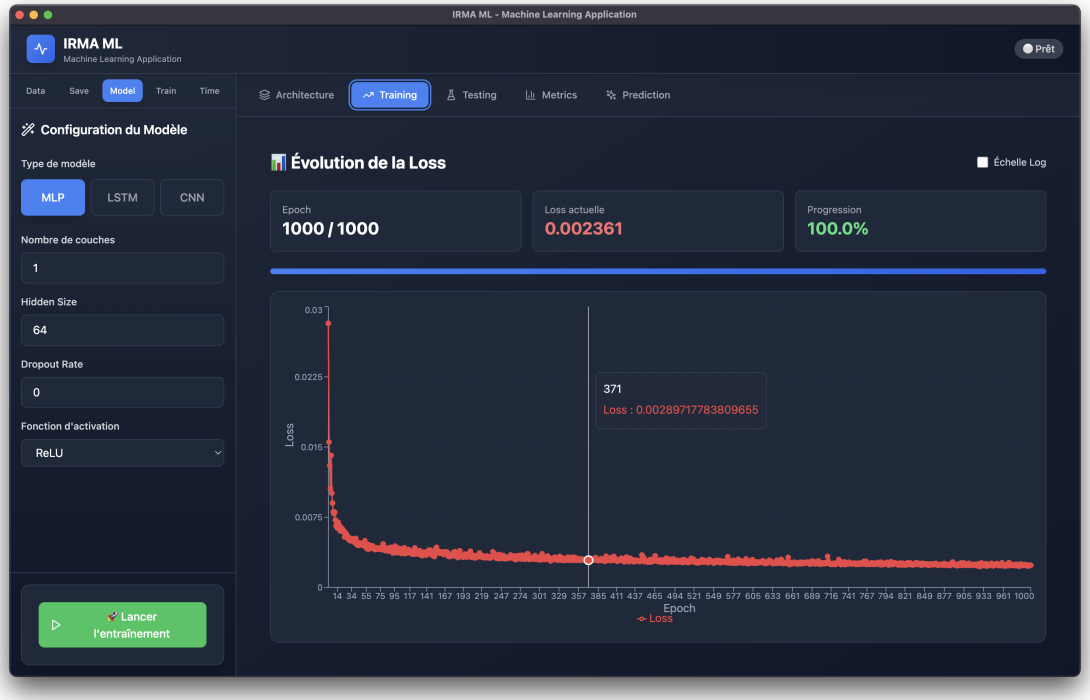


FIGURE 6.2 – Onglet Training – Courbe de loss en temps réel (1000 epochs)

L'onglet **Training** affiche en temps réel :

— **Progression :** Epoch actuelle / Total (ici 1000/1000)

— **Loss actuelle :** Valeur de la loss (ici 0.002361)

— **Barre de progression :** Pourcentage d'avancement (100%)

— **Graphique interactif :** Courbe de loss avec tooltip au survol

— **Option échelle logarithmique :** Pour mieux visualiser les variations

Onglet Testing – Résultats de Validation et Test

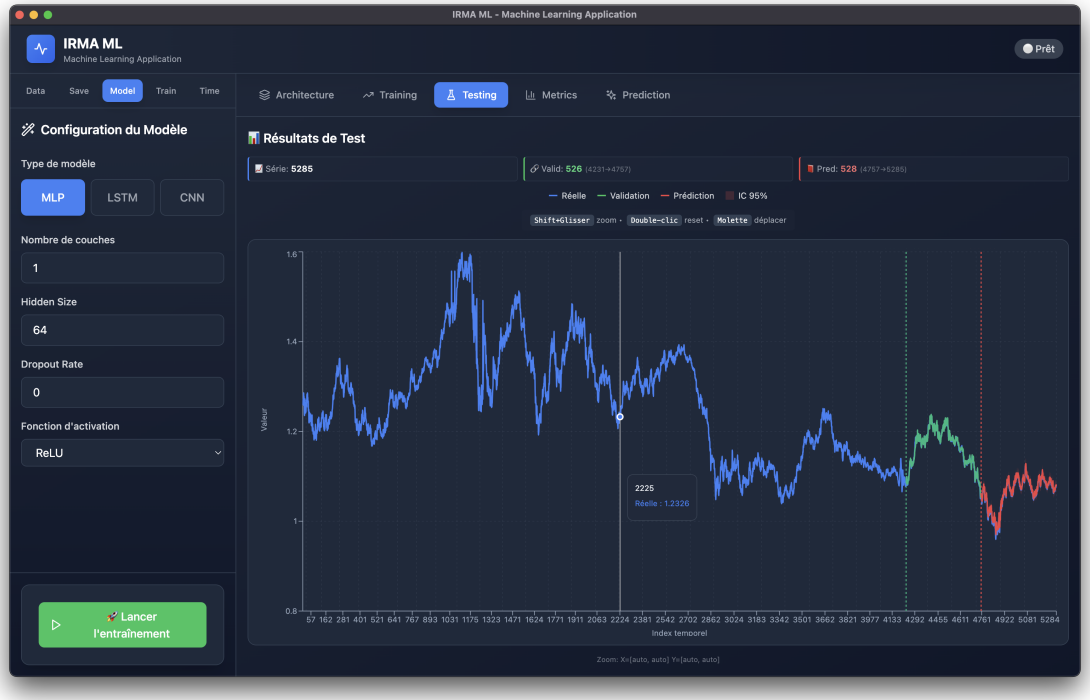


FIGURE 6.3 – Onglet Testing – Courbes de validation (vert), prédiction (rouge) et IC 95%

L'onglet **Testing** présente les trois courbes superposées :

— **Série réelle (bleu) :** 5285 points de données

— **Validation (vert) :** 526 points (indices 4231→4757) – Teacher forcing

— **Prédiction (rouge + halo) :** 528 points (indices 4757→5285) – One-step avec IC 95%

— **Interactions :** Shift+Glisser pour zoom, Double-clic pour reset, Molette pour déplacer

Onglet Metrics – Métriques de Performance

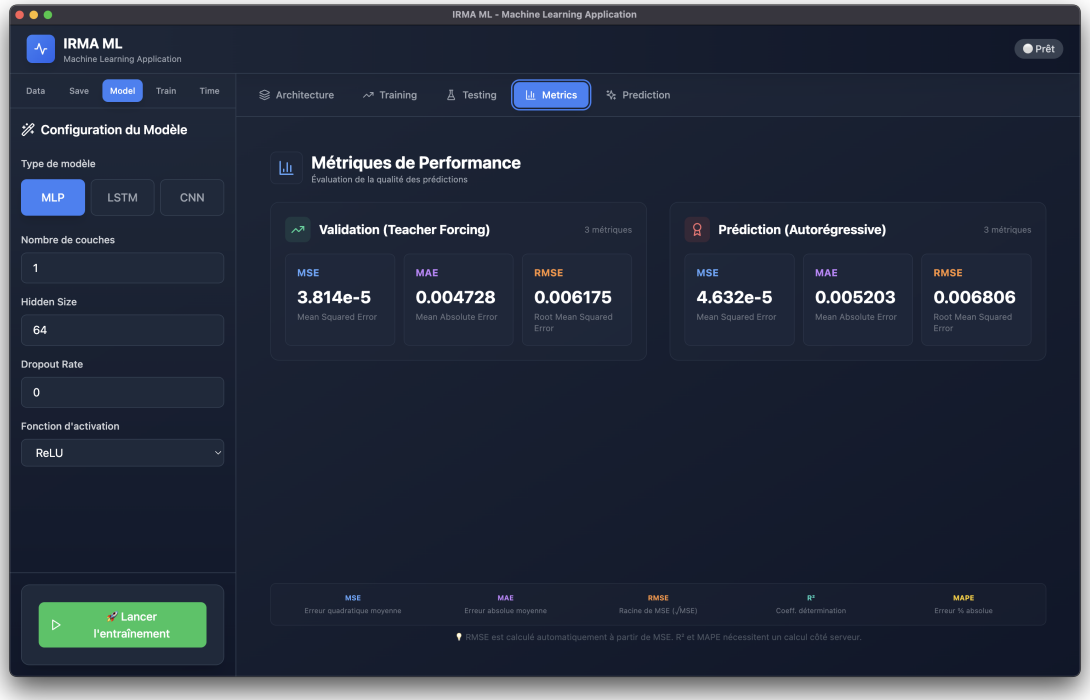


FIGURE 6.4 – Onglet Metrics – Comparaison Validation vs Prédiction

L'onglet **Metrics** affiche les métriques pour les deux phases :

— **Validation (Teacher Forcing) :** MSE = 3.814e-5, MAE = 0.004728, RMSE = 0.006175

— **Prédiction (Autorégressive) :** MSE = 4.632e-5, MAE = 0.005203, RMSE = 0.006806

— **Légende explicative :** Description de chaque métrique en bas de page

Onglet Prediction – Prédiction Future

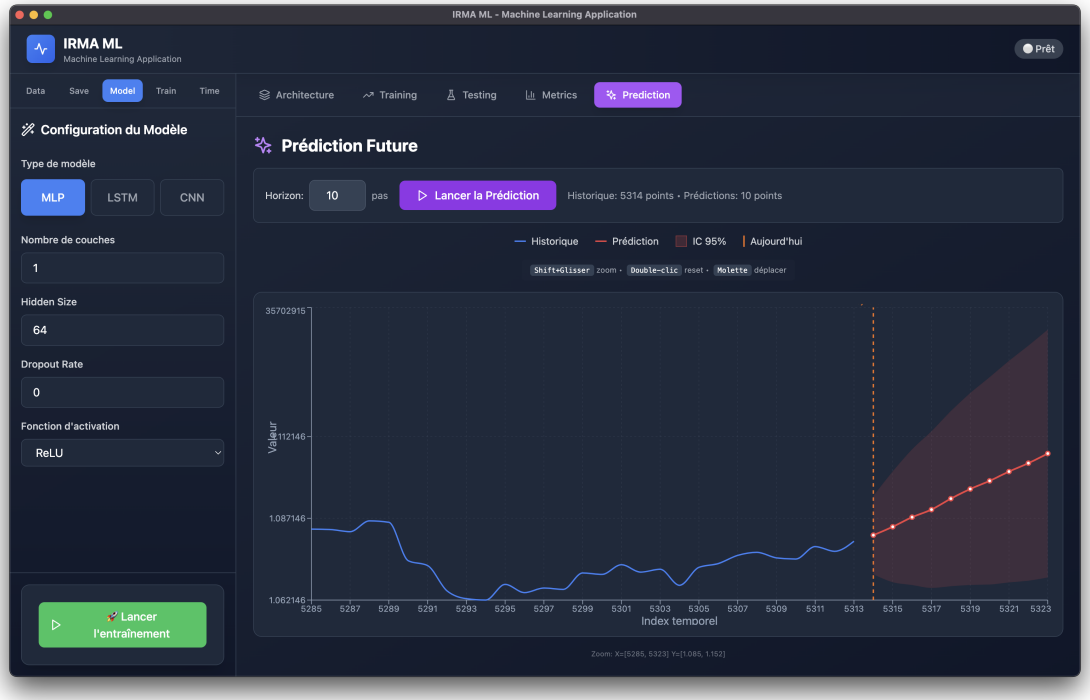


FIGURE 6.5 – Onglet Prediction – Prédiction sur horizon H=10 avec IC croissant

L'onglet **Prediction** permet la prédiction future :

— **Configuration de l'horizon :** Ici H = 10 pas

— **Historique (bleu) :** 5314 points de données connues

— **Prédictions (rouge) :** 10 points futurs

— **Intervalle de confiance 95% (halo rouge) :** IC croissant avec $\sqrt{step + 1}$

— **Ligne "Aujourd'hui" :** Séparation visuelle entre historique et prédictions

Architecture de l'Interface Web

L'interface JavaScript communique avec les mêmes serveurs backend (Serveur IA sur le port 8000, Serveur Data sur le port 8001) via des requêtes HTTP et Server-Sent Events (SSE).

```
INTERFACE_WEB/
|-- src/
|   |-- App.jsx                    # Composant principal React
|   |-- components/
|   |   |-- DatasetManager.jsx     # Gestion des datasets
|   |   |-- ModelConfig.jsx        # Configuration du modele
|   |   |-- TrainingPanel.jsx      # Lancement et monitoring
|   |   |-- MetricsDisplay.jsx     # Affichage des metriques
```

```
|   |   |-- PredictionPanel.jsx     # Predictions futures
|   |   |-- NetworkVisualizer.jsx   # VISUALISATION ARCHITECTURE
|   |   +-- TestingChart.jsx        # Graphiques validation/test
|   |-- hooks/
|   |   |-- useSSE.js               # Hook pour Server-Sent Events
|   |   +-- useAPI.js               # Hook pour requetes REST
|   |-- utils/
|   |   +-- networkRenderer.js      # Rendu du reseau de neurones
|   +-- styles/
|       +-- tailwind.css
|-- package.json
+-- vite.config.js
```

Listing 6.1 – Structure du projet JavaScript

Fonctionnalités de l'Interface

TABLE 6.1 – Onglets de l'interface IRMA ML

Onglet	Fonctionnalité
Architecture	Visualisation du pipeline du réseau avec blocs colorés (Input, Linear, ReLU, Output) et explication des paramètres
Training	Courbe de loss en temps réel avec tooltip interactif et option échelle logarithmique
Testing	Superposition des courbes réelle/validation/prédiction avec zones d'IC et interactions zoom/pan
Metrics	Tableau comparatif des métriques (MSE, MAE, RMSE) pour validation et prédiction
Prediction	Prédiction future sur horizon H configurable avec IC croissant

Développement Assisté par IA

Le développement de cette interface a été réalisé avec l'assistance de Claude (Anthropic). Cette approche a permis de :

1. **Accélérer le prototypage :** Génération rapide des composants React et de la logique de rendu
2. **Résoudre des problèmes complexes :** Algorithmes de placement automatique des neurones, calcul des courbes de Bézier pour les connexions
3. **Maintenir la cohérence :** S'assurer que l'interface JS respecte exactement les mêmes endpoints et formats de données que l'interface Python
4. **Design moderne :** Interface sombre professionnelle avec Tailwind CSS

Retour d'Expérience

L'utilisation de l'IA pour le développement frontend s'est avérée particulièrement efficace pour les tâches de visualisation complexes. La génération de code pour le rendu des architectures de réseaux de neurones et les graphiques interactifs, qui aurait nécessité plusieurs jours de développement manuel, a été réalisée en quelques heures d'itérations avec Claude.

Comparaison des Interfaces

TABLE 6.2 – Comparaison Python Tkinter vs JavaScript React

Aspect	**Python (Tkinter/CTk)**	**JavaScript (React)**
Installation	Requiert Python + packages	Navigateur uniquement
Performance graphique	Limitée (Matplotlib)	Excellente (Canvas/SVG)
Visualisation réseau	Non disponible	**Complète et interactive**
Design	CustomTkinter basique	Moderne (Tailwind, dark mode)
Interactivité graphiques	Limitée	Zoom, pan, tooltips natifs
SSE streaming	Threading manuel	Natif (EventSource API)

Conclusion

L'interface JavaScript IRMA ML représente une évolution significative du projet, ajoutant une dimension pédagogique qui manquait à l'interface Python originale. La visualisation de l'architecture des réseaux de neurones et les graphiques interactifs permettent aux utilisateurs de mieux comprendre les modèles qu'ils configurent et entraînent.

Les deux interfaces restent disponibles et interchangeables, partageant les mêmes serveurs backend. L'utilisateur peut choisir celle qui correspond le mieux à ses besoins :

— **Interface Python :** Pour les utilisateurs familiers avec l'écosystème Python

— **Interface JavaScript :** Pour la visualisation pédagogique et l'accessibilité web

Chapitre 7

Annexes

7.1 Structure des Fichiers du Projet

```
MLApp/
|-- INTERFACE/
|   |-- interface_local_ctk.py       # Interface principale
|   |-- Themes/
|   |   |-- blue.json                # Theme CustomTkinter
|   |   +-- theme_ttk.json
|   +-- Docs/
|       |-- DOCUMENTATION_MODERNE.md
|       +-- REFERENCE_RAPIDE.md
|
|-- SERVEUR_IA/
|   |-- TEST_main_pred.py            # Point d'entree FastAPI
|   |-- classes.py                   # Modeles Pydantic
|   |-- test_fonctions_pour_main.py  # Fonctions utilitaires
|   |-- __init__.py
|   |-- trains/
|   |   |-- training_MLP.py          # Generateur entrainement MLP
|   |   |-- training_CNN.py          # Generateur entrainement CNN
|   |   |-- training_LSTM.py         # Generateur entrainement LSTM
|   |   +-- __init__.py
|   |-- test/
|   |   |-- test_testing.py          # Module validation
|   |   |-- test_prediction_strategie.py  # 4 strategies prediction
|   |   |-- testing_pred.py
|   |   +-- __init__.py
|   +-- models/
|       |-- model_MLP.py             # Architecture MLP PyTorch
|       |-- model_CNN.py             # Architecture CNN PyTorch
|       |-- model_LSTM.py            # Architecture LSTM PyTorch
|       |-- optim.py                 # Optimizers et schedulers
|       +-- __init__.py
|
+-- SERVEUR_DATA/
    |-- main2.py                     # Point d'entree FastAPI
    |-- datasets/                    # Stockage JSON series temporelles
    |   |-- EURO.json
    |   |-- CACAO.json
```

```
    |   +-- marees_saint_jean_de_luz.json
    |-- models/                        # Stockage modeles .pth
    |   |-- mon_modele.pth
    |   +-- mon_modele2.pth
    +-- contextes/                     # Stockage contextes JSON
        |-- contexte_model_mlp.json
        +-- contexte_model_lstm.json
```

Listing 7.1 – Arborescence complète du projet

7.2 Dépendances Python

TABLE 7.1 – Dépendances Python du projet

Package	Usage	Version
fastapi	Framework API REST asynchrone	≥ 0.100
uvicorn	Serveur ASGI pour FastAPI	≥ 0.20
torch	Framework Deep Learning (PyTorch)	≥ 2.0
pydantic	Validation et sérialisation de données	≥ 2.0
customtkinter	Interface graphique moderne	≥ 5.0
matplotlib	Visualisation et graphiques	≥ 3.5
numpy	Calcul numérique	≥ 1.24
pandas	Manipulation de données (NaN handling)	≥ 2.0
requests	Client HTTP pour l'UI	≥ 2.28

7.3 Commandes de Lancement

```
# Terminal 1: Serveur Data (port 8001)
cd SERVEUR_DATA
python -m uvicorn main2:app --host 0.0.0.0 --port 8001 --reload

# Terminal 2: Serveur IA (port 8000)
cd SERVEUR_IA
python -m uvicorn TEST_main_pred:app --host 0.0.0.0 --port 8000 --reload

# Terminal 3: Interface Utilisateur
cd INTERFACE
python interface_local_ctk.py
```

Listing 7.2 – Lancement des serveurs

7.4 Références Bibliographiques

— Taieb, S. B., & Hyndman, R. J. (2014). A gradient boosting approach to the Kaggle load forecasting competition. *International Journal of Forecasting*, 30(2), 382-394.

— Chevillon, G. (2007). Direct multi-step estimation and forecasting. *Journal of Economic Surveys*, 21(4), 746-785.

— Benidis, K., et al. (2022). Deep Learning for Time Series Forecasting : Tutorial and Literature Survey. *ACM Computing Surveys*.

— FastAPI Documentation : https://fastapi.tiangolo.com/

— PyTorch Documentation : https://pytorch.org/docs/

— CustomTkinter Documentation : https://customtkinter.tomschimansky.com/