Cahier des Charges

Application de Prédiction de Séries Temporelles par Réseaux de Neurones

MLApp

Projet Génie Logiciel

Membres du groupe:
Gatien Séguy - Serveur IA
Maxime Degraeve - User Interface
Sofiane Boucherba - Serveur Data

15 janvier 2026

Table des matières

1 Introduction **3**
- 1.1 Contexte du Projet . . . 3
- 1.2 Méthodologie de Conception . . . 3
  - 1.2.1 Approche Orientée Objet . . . 3

2 Spécifications Fonctionnelles **4**
- 2.1 F-1 : Gestion des Données . . . 4
  - 2.1.1 F-1.1 : Importation de Données . . . 4
  - 2.1.2 F-1.2 : Sélection de Dataset . . . 5
  - 2.1.3 F-1.3 : Filtrage Temporel . . . 5
- 2.2 F-2 : Configuration du Modèle . . . 6
  - 2.2.1 F-2.1 : Choix du Type de Réseau . . . 6
  - 2.2.2 F-2.2 : Configuration Architecture MLP . . . 6
  - 2.2.3 F-2.3 : Configuration Architecture CNN . . . 7
  - 2.2.4 F-2.4 : Configuration Architecture LSTM . . . 7
- 2.3 F-3 : Configuration de l'Entraînement . . . 7
  - 2.3.1 F-3.1 : Paramètres Temporels . . . 7
  - 2.3.2 F-3.2 : Fonction de Perte . . . 8
  - 2.3.3 F-3.3 : Optimiseur . . . 8
  - 2.3.4 F-3.4 : Paramètres d'Entraînement . . . 8
- 2.4 F-4 : Entraînement et Monitoring . . . 8
  - 2.4.1 F-4.1 : Lancement de l'Entraînement . . . 8
  - 2.4.2 F-4.2 : Visualisation en Temps Réel . . . 9
  - 2.4.3 F-4.3 : Annulation de l'Entraînement . . . 9
- 2.5 F-5 : Évaluation et Test . . . 10
  - 2.5.1 F-5.1 : Phase de Test Automatique . . . 10
  - 2.5.2 F-5.2 : Métriques Calculées . . . 10
- 2.6 F-6 : Prédiction Future . . . 11
  - 2.6.1 F-6.1 : Prédiction sur Horizon Défini . . . 11
- 2.7 F-7 : Sauvegarde et Chargement . . . 11
  - 2.7.1 F-7.1 : Sauvegarde du Modèle et du Contexte . . . 11
  - 2.7.2 F-7.2 : Chargement d'un Modèle Existant . . . 12

3 Spécifications Non-Fonctionnelles **13**
- 3.1 Performance . . . 13
  - 3.1.1 Utilisation des Ressources . . . 13
- 3.2 Fiabilité . . . 13

3.2.1 Gestion des Erreurs . . . 13
3.3 Utilisabilité . . . 13
3.3.1 Expérience Utilisateur . . . 14
3.4 Sécurité . . . 14
3.4.1 Protection des Données . . . 14

4 Spécification architecturale **15**
4.1 Architecture Globale . . . 15
4.1.1 Vue d'Ensemble . . . 15
4.2 Composants Principaux . . . 15
4.2.1 Interface Utilisateur . . . 15
4.2.2 Serveur IA . . . 16
4.2.3 Serveur Data . . . 17

5 Spécification détaillé **18**
5.1 Diagramme de classe UI . . . 19
5.2 Diagramme de classe Serveur IA . . . 20
5.3 Diagramme de classe Serveur IA simplifié . . . 21
5.4 Diagramme de classe Serveur Data simplifié . . . 21
5.5 Diagrammes de Séquence — Entraînement Complet . . . 22

Chapitre 1

Introduction

1.1 Contexte du Projet

L'application MLApp permet de prédire des séries temporelles à l'aide de réseaux de neurones. Elle offre une interface simple pour importer des données, configurer un modèle (MLP, CNN ou LSTM), lancer l'entraînement, visualiser les performances en temps réel et effectuer des prédictions futures.

1.2 Méthodologie de Conception

1.2.1 Approche Orientée Objet

Ce projet suit une **méthodologie de conception orientée objet** en utilisant UML (Unified Modeling Language) pour spécifier l'architecture et la structure du système.

Phases de Conception

1. Conception Architecturale

— Découpe du système en sous système faiblement couplés

— Spécification des responsabilités de chaque composant

2. Conception Détaillée

— Structure orientée objet (classes, objets, associations)

— Hiérarchie des classes

Chapitre 2

Spécifications Fonctionnelles

Ce chapitre détaille 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.

Pré-conditions :

— Le fichier doit être au format JSON

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

— Les longueurs des deux listes doivent être identiques

Post-conditions :

— Les données sont stockées dans le Serveur Data

— Un nom unique est attribué au dataset

— Un identifiant unique (UUID) est généré

— Le dataset devient disponible pour la sélection

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
5. Le système demande un nom pour le dataset
6. Le système vérifie l'unicité du nom
7. Les données sont envoyées au Serveur Data via le Serveur IA

8. Le système confirme l'importation

Flux alternatifs :

— **4a.** Format invalide → Affichage d'un message d'erreur précis

— **6a.** Nom déjà existant → Demande d'un nouveau nom unique

Format JSON attendu :

```
{
  "timestamps": [
    "2020-01-01 00:00:00",
    "2020-01-02 00:00:00",
    ...
  ],
  "values": [
    10.5,
    11.2,
    ...
  ]
}
```

Listing 2.1 – Format de dataset

2.1.2 F-1.2 : Sélection de Dataset

Description

L'utilisateur définit la plage de dates (`date_debut`, `date_fin`) depuis l'interface. Ces bornes sont transmises au Serveur IA, qui demande ensuite au Serveur Data le dataset déjà croppé sur cet intervalle via l'endpoint `GET /timeseries?name={dataset_name}start={date_debut}end={date_fin}`.

Pré-conditions :

— Au moins un dataset doit être disponible dans le Serveur Data

Post-conditions :

— Le dataset sélectionné est chargé en mémoire

— Les informations du dataset sont affichées (taille, plage de dates)

— Le `dataset_name` sélectionné est transmis au Serveur IA lors des requêtes d'entraînement et de prédiction.

2.1.3 F-1.3 : Filtrage Temporel

Description

L'utilisateur peut définir une plage de dates pour filtrer les données utilisées.

Paramètres :

— **Date de début :** Format YYYY-MM-DD

— **Date de fin :** Format YYYY-MM-DD

Contraintes :

— Date de fin > Date de début

— Les dates doivent être dans la plage du dataset

2.2 F-2 : Configuration du Modèle

2.2.1 F-2.1 : Choix du Type de Réseau

Description

L'utilisateur sélectionne le type d'architecture neuronale parmi trois options.

Options disponibles :

— **MLP** (Multi-Layer Perceptron) : Réseau fully-connected classique

— **CNN** (Convolutional Neural Network) : Réseau à convolutions 1D

— **LSTM** (Long Short-Term Memory) : Réseau récurrent avec mémoire

2.2.2 F-2.2 : Configuration Architecture MLP

TABLE 2.1 – Hyperparamètres du MLP

Paramètre	Type	Défaut	Contraintes
`nb_couches`	int	2	≥ 1
`hidden_size`	int	64	≥ 1
`dropout_rate`	float	0.0	$\in [0.0, 1.0]$
`fonction_activation`	str	ReLU	{ReLU, GELU, tanh, sigmoid, leaky_relu}
`use_batchnorm`	bool	False	{True, False}

Description des paramètres :

— **nb_couches :** Nombre de couches cachées dans le réseau

— **hidden_size :** Nombre de neurones par couche cachée

— **dropout_rate :** Taux de dropout pour la régularisation (0 = pas de dropout)

— **fonction_activation :** Fonction d'activation appliquée après chaque couche

— **use_batchnorm :** Utilisation de la normalisation par batch

2.2.3 F-2.3 : Configuration Architecture CNN

TABLE 2.2 – Hyperparamètres du CNN

Paramètre	Type	Défaut	Contraintes
`nb_couches`	int	2	≥ 1
`hidden_size`	int	64	≥ 1
`kernel_size`	int	3	≥ 1, impair recommandé
`stride`	int	1	≥ 1
`padding`	int	0	≥ 0
`fonction_activation`	str	ReLU	{ReLU, GELU, tanh, sigmoid}
`dropout_rate`	float	0.0	$\in [0.0, 1.0]$

2.2.4 F-2.4 : Configuration Architecture LSTM

TABLE 2.3 – Hyperparamètres du LSTM

Paramètre	Type	Défaut	Contraintes
`nb_couches`	int	2	≥ 1
`hidden_size`	int	64	≥ 1
`bidirectional`	bool	False	
`batch_first`	bool	True	

2.3 F-3 : Configuration de l'Entraînement

2.3.1 F-3.1 : Paramètres Temporels

TABLE 2.4 – Paramètres de la série temporelle

Paramètre	Type	Défaut	Contraintes
`horizon`	int	1	≥ 1
`pas_temporel`	int	1	≥ 1
`portion_decoupage`	float	0.8	$(0.0, 1.0)$
`dates`	list[str]	-	[début, fin]

Description :

— **horizon :** Nombre de pas de temps à prédire dans le futur

— **pas_temporel :** Pas d'échantillonnage des données (1 = toutes les données)

— **portion_decoupage :** Ratio de données pour l'entraînement (ex : 0.8 = 80% train, 20% test)

— **dates :** Plage temporelle à considérer [date_début, date_fin]

2.3.2 F-3.2 : Fonction de Perte

Description

Sélection de la fonction de perte pour l'optimisation du modèle.

Options :

— **MSE** (Mean Squared Error) : $L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ — défaut

— **MAE** (Mean Absolute Error) : $L = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

— **Huber Loss** : Combinaison de MSE et MAE, robuste aux outliers

2.3.3 F-3.3 : Optimiseur

TABLE 2.5 – Paramètres de l'optimiseur

Paramètre	Type	Défaut	Contraintes
`optimisateur`	str	Adam	{Adam, SGD, RMSprop, Adagrad, Adadelta}
`learning_rate`	float	0.001	> 0
`decroissance`	float	0.0	≥ 0 (weight decay)
`scheduler`	str	None	{None, Plateau, Cosine, OneCycle}
`patience`	int	5	≥ 1 (pour early stopping)

2.3.4 F-3.4 : Paramètres d'Entraînement

TABLE 2.6 – Paramètres du processus d'entraînement

Paramètre	Type	Défaut	Contraintes
`nb_epochs`	int	1000	≥ 1
`batch_size`	int	4	≥ 1
`clip_gradient`	float	None	> 0 si défini

2.4 F-4 : Entraînement et Monitoring

2.4.1 F-4.1 : Lancement de l'Entraînement

Description

Démarrage du processus d'entraînement avec streaming en temps réel des métriques.

Pré-conditions :

— Un dataset est sélectionné

— Les paramètres du modèle sont configurés

— Les paramètres d'entraînement sont définis

Post-conditions :

— Le modèle est entraîné

— Les métriques d'entraînement sont sauvegardées

— Le modèle et le contexte sont stockés dans le Serveur Data

Événements streamés (SSE) :

```
{
  "type": "train",
  "epoch": 1,
  "loss": 0.4532,
  "gradient_norm": 1.234,
  "learning_rate": 0.001
}
```

Listing 2.2 – Événements pendant l'entraînement

2.4.2 F-4.2 : Visualisation en Temps Réel

Métriques Affichées

L'interface affiche dynamiquement les métriques suivantes pendant l'entraînement :

— **Loss vs Époques :** Graphique linéaire mis à jour à chaque époque

— **Valeur actuelle de la loss :** Affichage numérique

— **Numéro de l'époque courante :** Progression

— **Barre de progression :** Pourcentage d'avancement

Métriques proposées (à implémenter) :

— **Norme du gradient :** Détection des problèmes de vanishing/exploding gradients

— **Learning rate :** Suivi du learning rate si scheduler actif

— **Temps par époque :** Estimation du temps restant

— **Utilisation mémoire :** Surveillance GPU/CPU

— **Loss de validation :** Si ensemble de validation créé

2.4.3 F-4.3 : Annulation de l'Entraînement

Description

L'utilisateur peut interrompre l'entraînement en cours à tout moment.

Comportement :

— L'entraînement s'arrête à la fin de l'époque courante

— Les résultats partiels sont conservés

— Possibilité de sauvegarder l'état intermédiaire (checkpoint)

2.5 F-5 : Évaluation et Test

2.5.1 F-5.1 : Phase de Test Automatique

Description

Après l'entraînement, le modèle est automatiquement testé sur l'ensemble de test.

Processus :

1. Le Serveur IA utilise l'ensemble X_{test} et y_{test}
2. Le modèle effectue des prédictions par batch
3. Les prédictions sont dénormalisées via la fonction inverse
4. Les métriques sont calculées
5. Les résultats sont streamés événement par événement

Événements streamés :

```
{
  "type": "test_prediction",
  "y_true": 10.5,
  "y_pred": 10.3
}

{
  "type": "test_metrics",
  "mse": 0.0521,
  "mae": 0.1832,
  "rmse": 0.2283,
  "mape": 1.745,
  "r2": 0.9412
}
```

Listing 2.3 – Événements de test

2.5.2 F-5.2 : Métriques Calculées

TABLE 2.7 – Métriques d'évaluation

Métrique	Formule	Interprétation
MSE	$\frac{1}{n}\sum(y_i - \hat{y}_i)^2$	Plus faible = meilleur
MAE	$\frac{1}{n}\sum\lvert y_i - \hat{y}_i\rvert$	Plus faible = meilleur
RMSE	$\sqrt{\text{MSE}}$	Même unité que y
MAPE	$\frac{100\%}{n}\sum\frac{\lvert y_i - \hat{y}_i\rvert}{\lvert y_i\rvert}$	Erreur en pourcentage
R^2	$1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$	$\in [-\infty, 1]$, 1 = parfait

2.6 F-6 : Prédiction Future

2.6.1 F-6.1 : Prédiction sur Horizon Défini

Description

Utilisation du modèle entraîné pour prédire des valeurs futures sur l'horizon défini.

Pré-conditions :

— Un modèle est entraîné ou chargé

— L'horizon de prédiction est défini

Flux de données : L'interface utilisateur envoie au Serveur IA le `dataset_name` et l'`horizon` souhaité. Le Serveur IA interroge alors le Serveur Data pour obtenir les dernières valeurs nécessaires à la prédiction.

Post-conditions :

— Les prédictions futures sont affichées dans l'onglet « Prediction »

— Les prédictions sont exportables (CSV, JSON)

Méthodes de prédiction :

— **Prédiction directe :** Utilise les dernières valeurs historiques

— **Prédiction itérative :** Utilise les prédictions précédentes comme input

2.7 F-7 : Sauvegarde et Chargement

2.7.1 F-7.1 : Sauvegarde du Modèle et du Contexte

Description

Après l'entraînement et le test, le système sauvegarde automatiquement tous les éléments nécessaires à la reproductibilité.

Éléments sauvegardés :

— **Poids du modèle :** `state_dict` PyTorch (fichier .pth)

— **Architecture :** Hyperparamètres du modèle

— **Contexte d'entraînement :** Tous les paramètres utilisés

— **Métriques de test :** Résultats finaux

— **Historique :** Loss par époque

— **Normalisation :** Paramètres pour dénormalisation

Contexte complet : Envoyé au Serveur Data via l'endpoint `POST /context`

Format de stockage :

```
{
  "model_id": "uuid-1234-5678",
  "model_type": "MLP",
  "architecture": {
    "nb_couches": 2,
    "hidden_size": 64,
    ...
  },
  "training_params": {
    "nb_epochs": 1000,
    "batch_size": 4,
    ...
  },
  "normalization": {
    "method": "standardization",
    "mean": 10.5,
    "std": 2.3
  },
  "test_metrics": {
    "mse": 0.052,
    "mae": 0.183,
    "r2": 0.941
  },
  "training_history": [
    {"epoch": 1, "loss": 0.45},
    {"epoch": 2, "loss": 0.38},
    ...
  ]
}
```

Listing 2.4 – Structure du contexte

2.7.2 F-7.2 : Chargement d'un Modèle Existant

Description

L'utilisateur peut charger un modèle précédemment entraîné pour effectuer de nouvelles prédictions.

Flux :

1. Clic sur « Charger Modèle »
2. Affichage de la liste des modèles disponibles
3. Sélection d'un modèle
4. Chargement des poids et du contexte
5. Activation de l'onglet « Prediction »

Chapitre 3

Spécifications Non-Fonctionnelles

3.1 Performance

3.1.1 Utilisation des Ressources

— **CPU :** Optimisé pour multi-threading
— **GPU :** Support MPS (Apple Silicon) et CUDA (NVIDIA)
— **Stockage :** Compression des modèles sauvegardés

3.2 Fiabilité

3.2.1 Gestion des Erreurs

Stratégie de Gestion des Erreurs (Pydantic)

1. Validation des entrées à tous les niveaux
2. Messages d'erreur clairs et actionnables
3. Logging détaillé côté serveur
4. Pas d'exposition de détails sensibles

Types d'erreurs gérées :

— Erreurs de validation des paramètres
— Erreurs de format de données
— Erreurs d'entraînement (NaN, explosion du gradient)
— Erreurs de communication réseau
— Erreurs de stockage

3.3 Utilisabilité

3.3.1 Expérience Utilisateur

— **Courbe d'apprentissage :** Utilisable après 15 minutes de découverte
— **Documentation :** Help contextuelle pour chaque paramètre
— **Tooltips :** Explications au survol des éléments

3.4 Sécurité

3.4.1 Protection des Données

— **Chiffrement :** pour toutes les communications

Chapitre 4

Spécification architecturale

4.1 Architecture Globale

4.1.1 Vue d'Ensemble

L'application MLApp suit une **architecture trois-tiers** avec séparation claire des responsabilités. On peut retrouver ci-dessous le diagrammes de composants UML

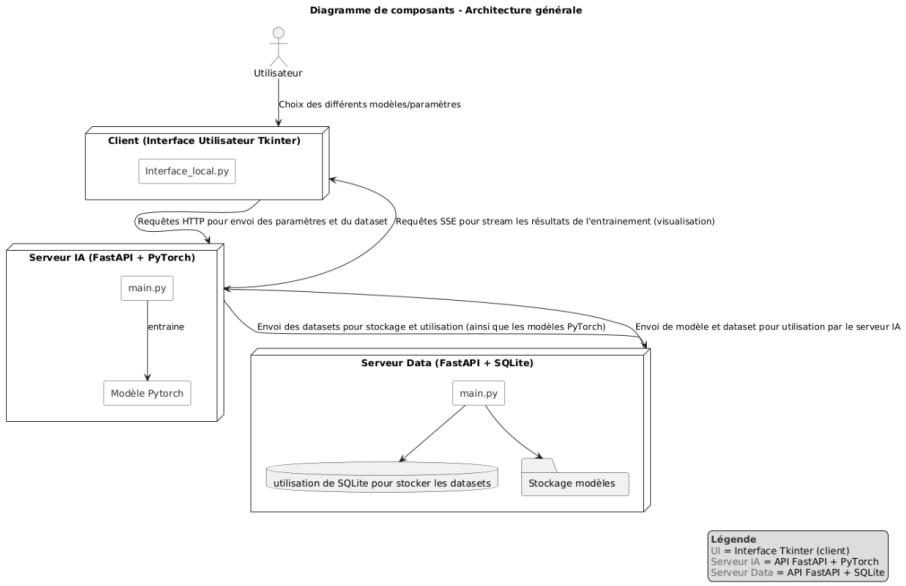


FIGURE 4.1 – Diagramme de composant

4.2 Composants Principaux

4.2.1 Interface Utilisateur

Technologies :

— Framework : Tkinter

— Graphiques : Matplotlib avec backend TkAgg
— Threading : Pour le streaming non-bloquant
— HTTP Client : requests library

Structure :

— `Fenetre_Acceuil` : Fenêtre principale
— `Cadre_Entrainement` : Onglet Training
— `Cadre_Testing` : Onglet Testing
— `Cadre_Metrics` : Onglet Metrics
— `Cadre_Prediction` : Onglet Prediction
— `Fenetre_Params` : Configuration du modèle
— `Fenetre_Params_horizon` : Paramètres temporels
— `Fenetre_Choix_datasets` : Sélection de dataset

Responsabilités :

— Collecte des paramètres utilisateur
— Envoi des requêtes au Serveur IA
— Réception et affichage du streaming
— Mise à jour des graphiques en temps réel
— Gestion de l'état de l'application

4.2.2 Serveur IA

Technologies :

— Framework : FastAPI
— ML Framework : PyTorch
— Streaming : Server-Sent Events (SSE)
— Validation : Pydantic models

Endpoints principaux :

— `POST /train_full` : Entraînement complet avec streaming
— `GET /` : Health check
— `POST /predict` (futur) : Prédiction sur nouvelles données
— `GET /models` (futur) : Liste des modèles disponibles

Modules :

— `main.py` : Point d'entrée FastAPI
— `classes.py` : Modèles Pydantic
— `trains/` : Modules d'entraînement (MLP, CNN, LSTM)
— `test/` : Module de testing
— `fonctions_pour_main.py` : Fonctions utilitaires

4.2.3 Serveur Data

Technologies proposées :

— Framework : FastAPI

— Base de données : SQLite

— Stockage : Système de fichiers pour modèles

— Validation : Pydantic models

Endpoints proposés :

— `POST /datasets` : Créer un nouveau dataset

— `GET /datasets` : Lister tous les datasets

— `GET /datasets/{id}` : Récupérer un dataset spécifique

— `DELETE /datasets/{id}` : Supprimer un dataset

— `POST /models` : Sauvegarder un modèle

— `GET /models` : Lister tous les modèles

— `GET /models/{id}` : Récupérer un modèle

— `DELETE /models/{id}` : Supprimer un modèle

— `GET /timeseries?name={dataset_name}start={date_debut}end={date_fin}` : Récupérer un dataset croppé

— `POST /context` : Sauvegarder le contexte complet d'entraînement

— `POST /metrics` : Sauvegarder les métriques associées à un modèle

— `POST /predictions` : Sauvegarder les prédictions produites

Chapitre 5

Spécification détaillé

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, en représentant les classes principales, leurs attributs, leurs méthodes et les relations qui assurent la cohérence de l'ensemble.

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 des métriques d'entraînement, de test et de prédiction en temps réel.

— **Le Serveur IA** constitue le cœur du système : il orchestre le flux de données entre l'UI et le Serveur Data, assure le prétraitement des séries temporelles, l'entraînement des modèles de réseaux de neurones (MLP, CNN, LSTM), l'évaluation automatique et la génération de prédictions. Il repose sur `FastAPI` pour l'exposition des routes et sur `PyTorch` pour la partie apprentissage.

— **Le Serveur Data** centralise la gestion et la persistance des données : il stocke les datasets, les modèles entraînés, les contextes d'apprentissage, les métriques et les prédictions. Il communique exclusivement avec le Serveur IA via des endpoints REST sécurisés.

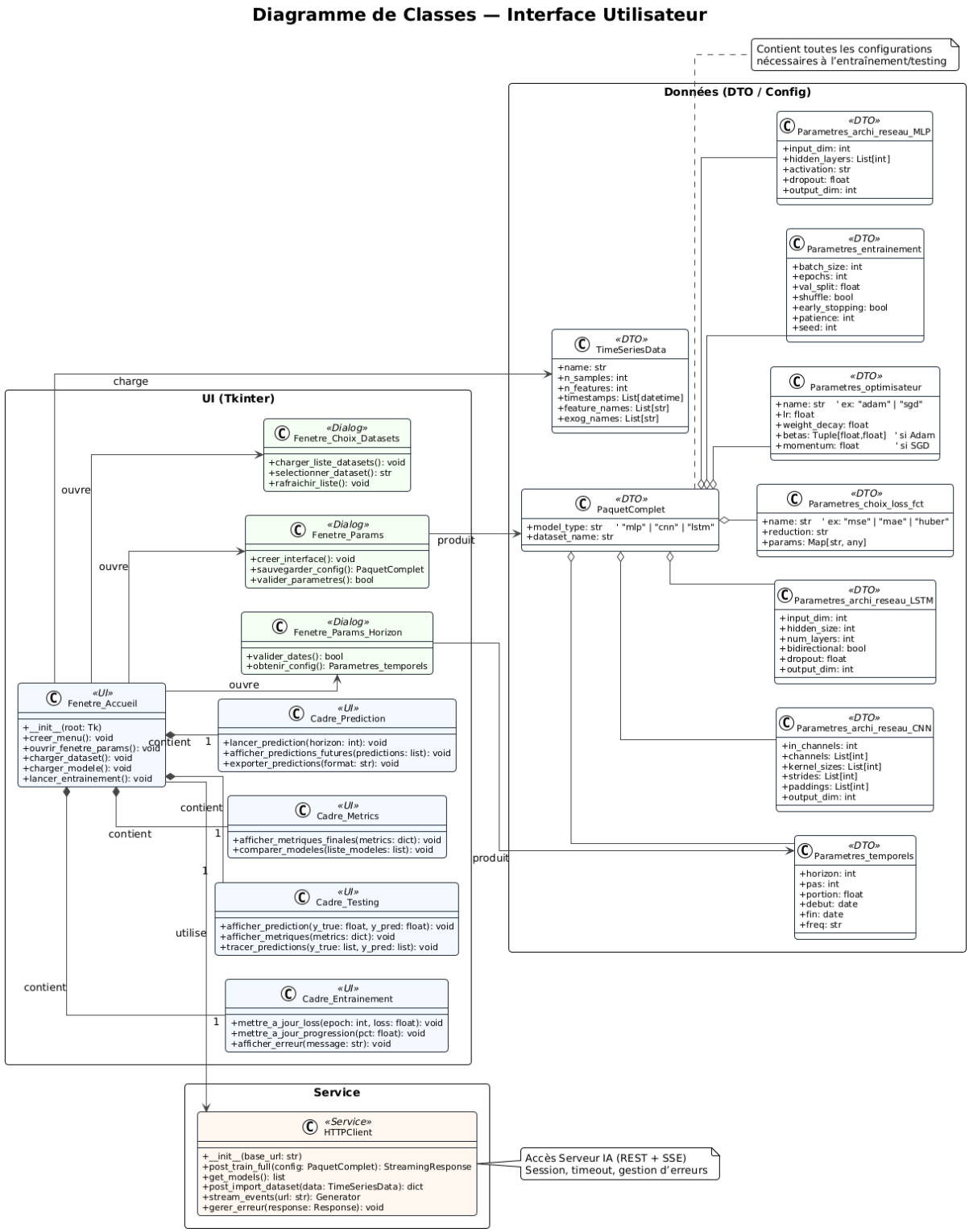
L'ensemble de ces diagrammes permet de visualiser clairement :

1. la répartition des responsabilités entre les composants,
2. les dépendances fonctionnelles entre les modules (UI, IA, Data),
3. la circulation des informations — depuis la configuration utilisateur jusqu'à la sauvegarde des résultats.

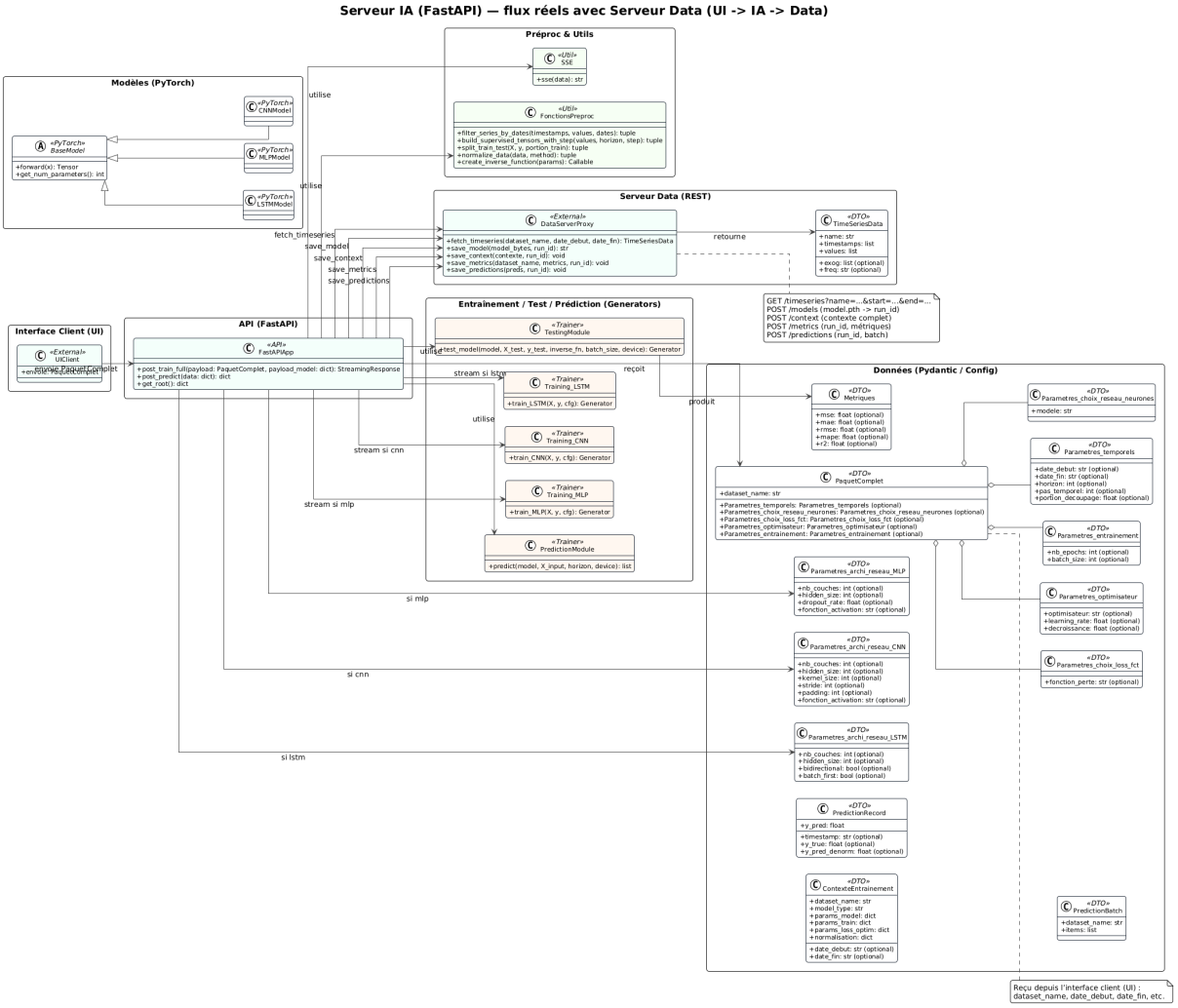
Les diagrammes suivants détaillent successivement :

1. le diagramme de classes de l'**Interface Utilisateur**,
2. le diagramme de classes complet du **Serveur IA**,
3. et enfin un diagramme **simplifié du Serveur IA**, illustrant le flux global entre l'UI, le module d'entraînement et le Serveur Data.

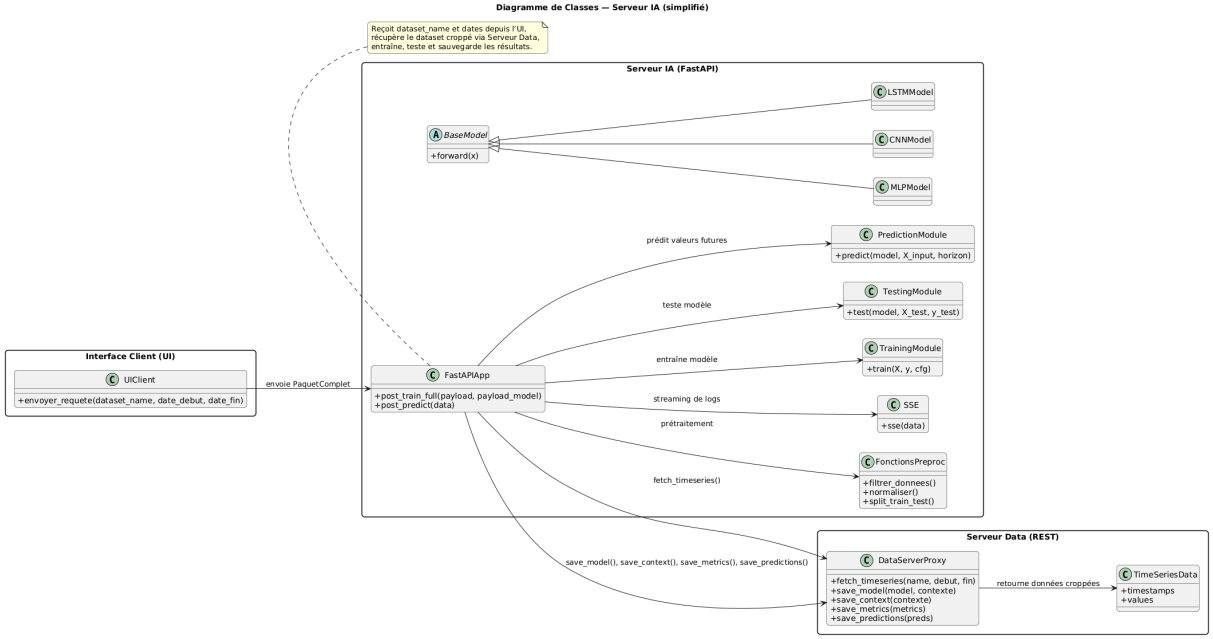
5.1 Diagramme de classe UI



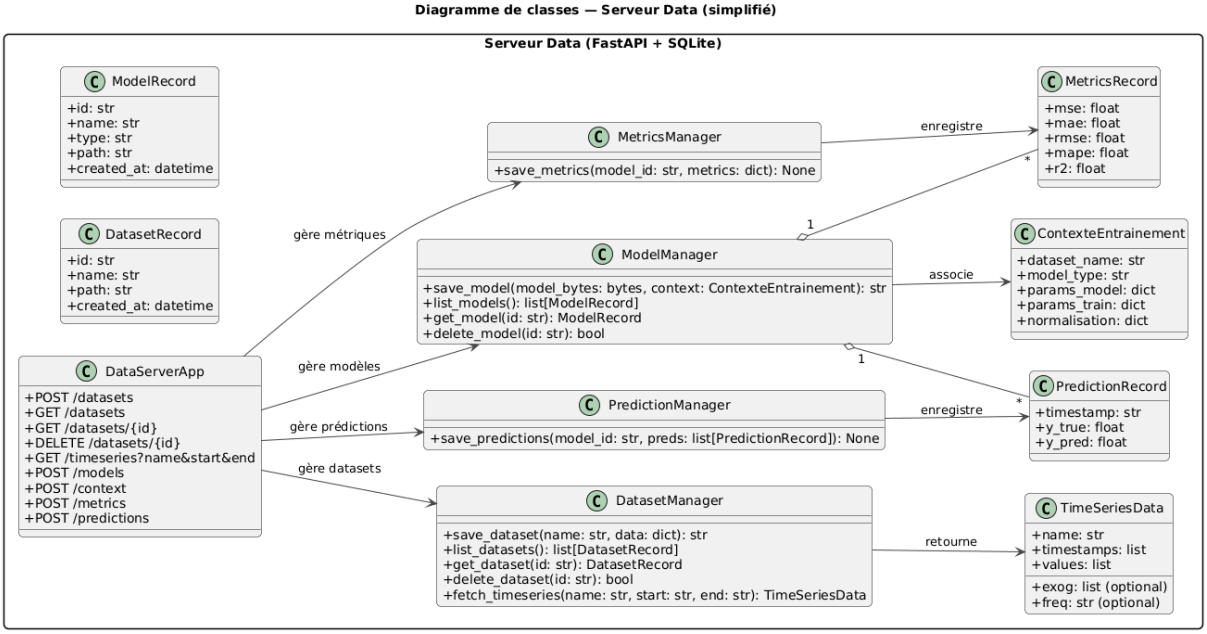
5.2 Diagramme de classe Serveur IA



5.3 Diagramme de classe Serveur IA simplifié



5.4 Diagramme de classe Serveur Data simplifié



5.5 Diagrammes de Séquence — Entraînement Complet

Les diagrammes suivants illustrent le fonctionnement séquentiel du système lors d'un **cycle complet d'entraînement et de prédiction**. Chaque diagramme correspond à une phase spécifique du processus, depuis la configuration initiale jusqu'à la sauvegarde finale du modèle.

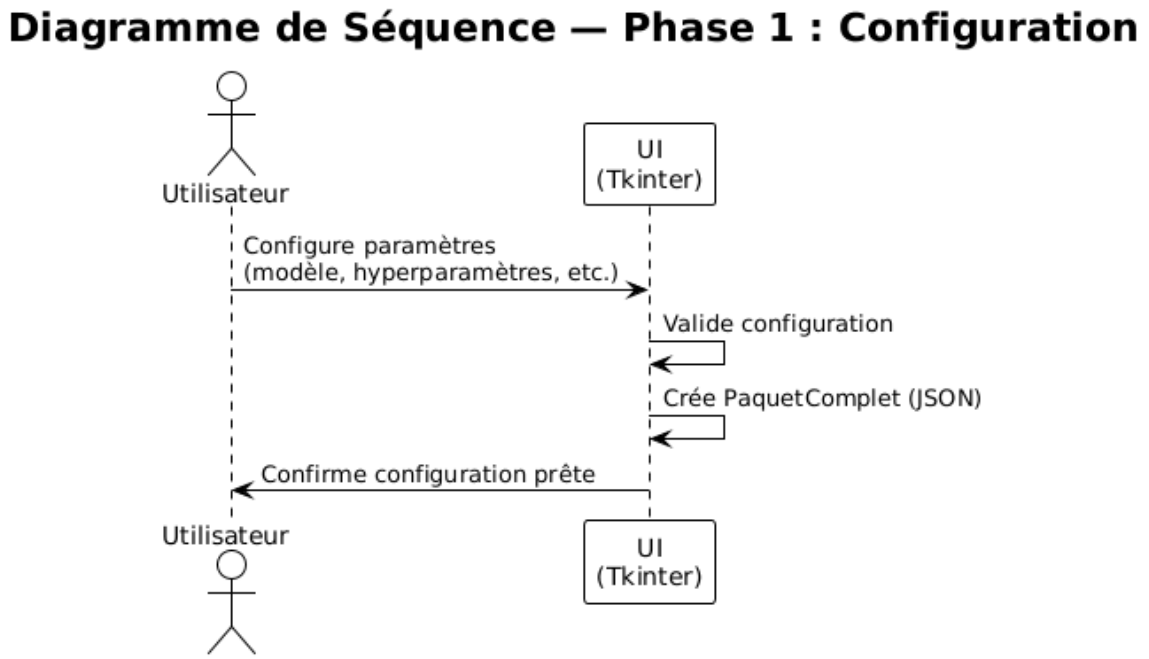


FIGURE 5.1 – Phase 1 — Configuration de l'entraînement (création du PaquetComplet)

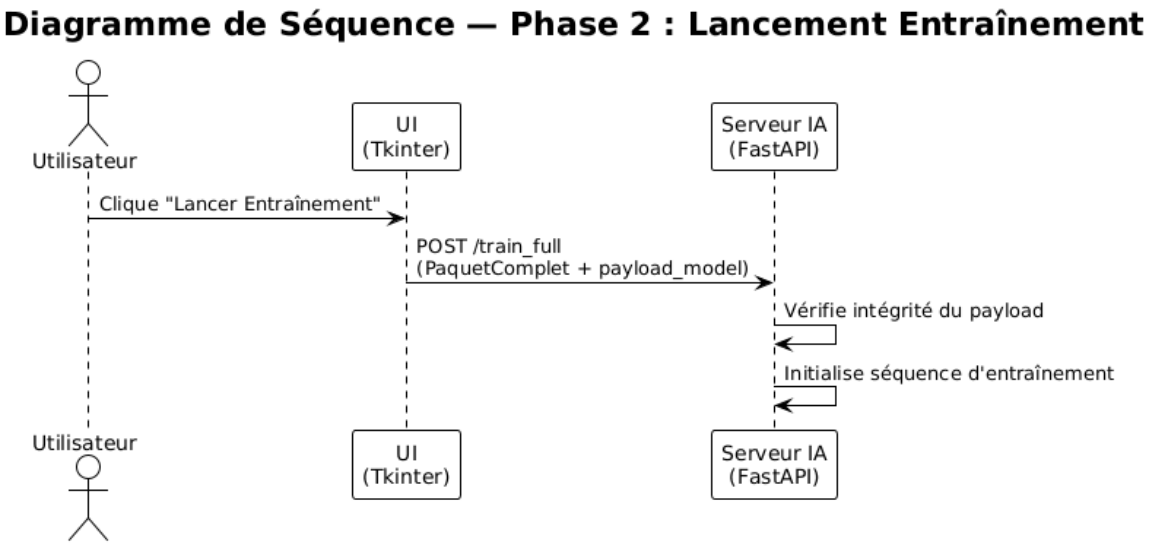


FIGURE 5.2 – Phase 2 — Lancement de l'entraînement (requête POST /train_full)

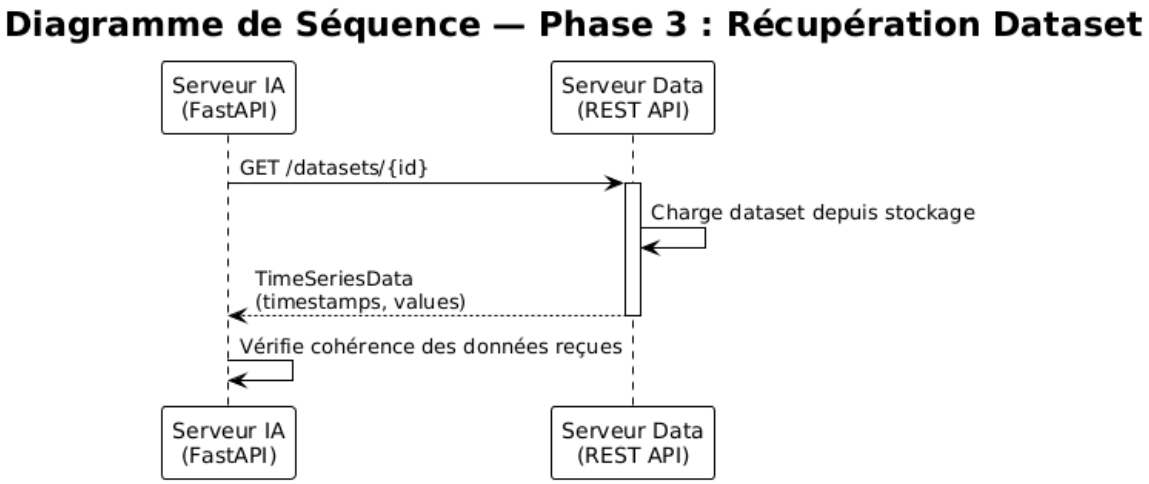


FIGURE 5.3 – Phase 3 — Récupération du dataset auprès du Serveur Data

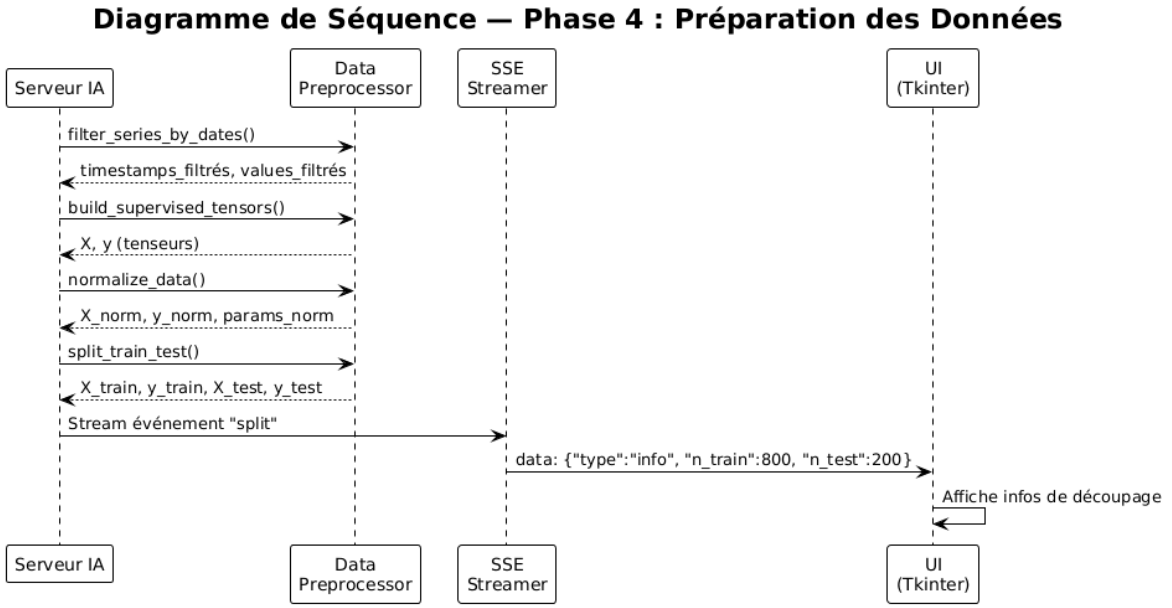


FIGURE 5.4 – Phase 4 — Préparation des données (filtrage, tenseurs, normalisation, split)

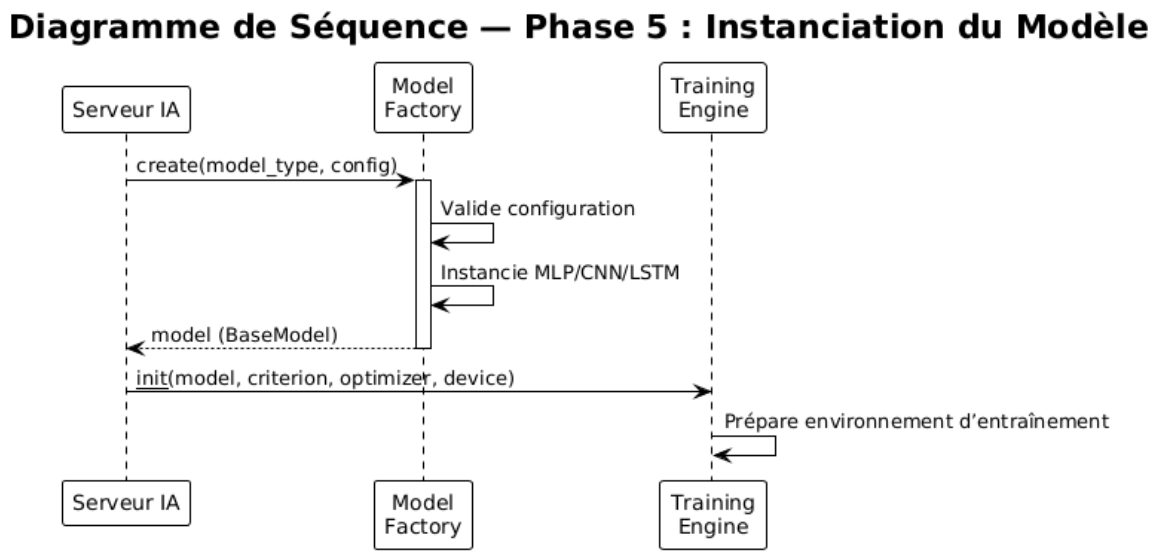


FIGURE 5.5 – Phase 5 — Instanciation du modèle (création via ModelFactory)

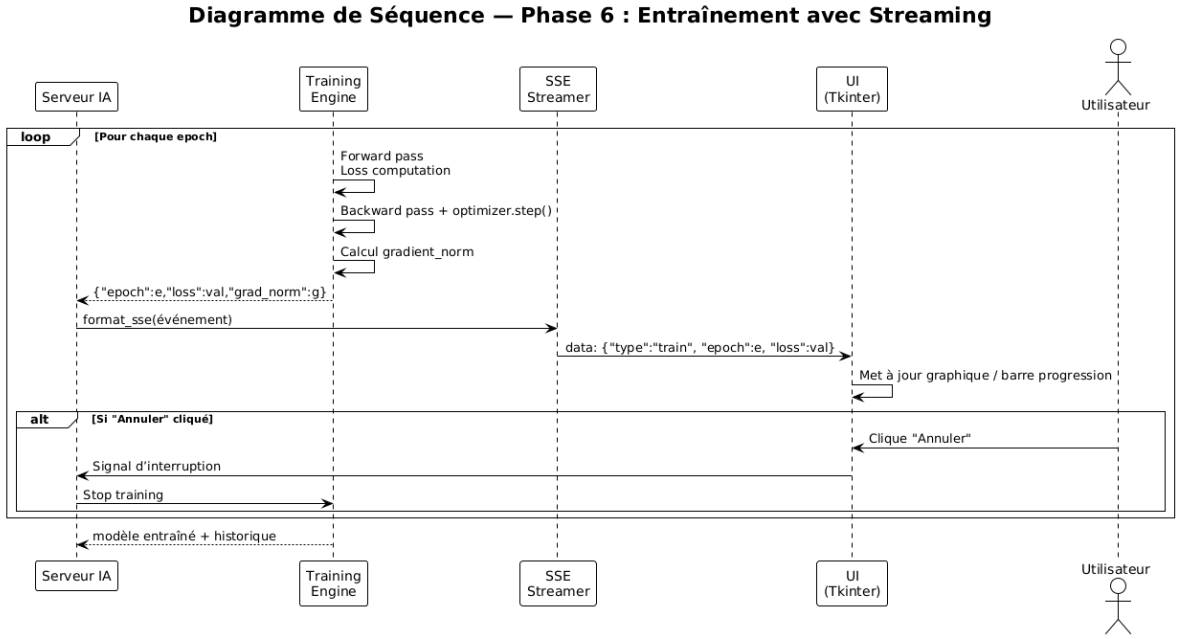


FIGURE 5.6 – Phase 6 — Entraînement avec streaming SSE (Server-Sent Events)

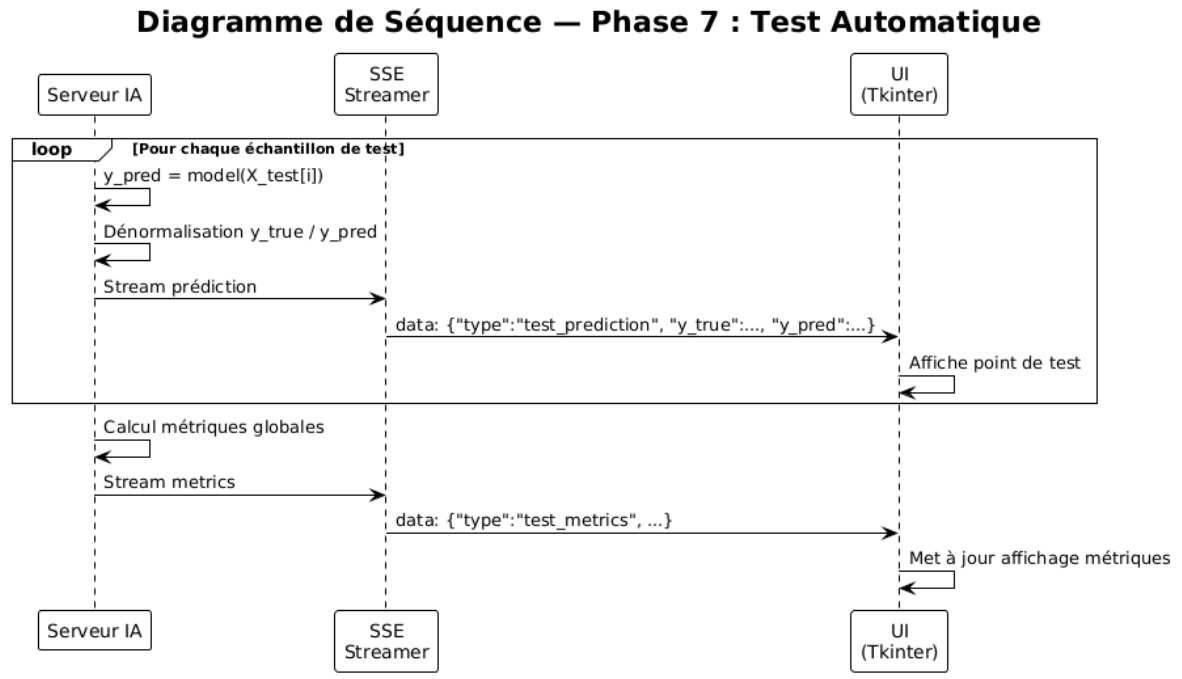


FIGURE 5.7 – Phase 7 — Test automatique du modèle sur l'ensemble de test

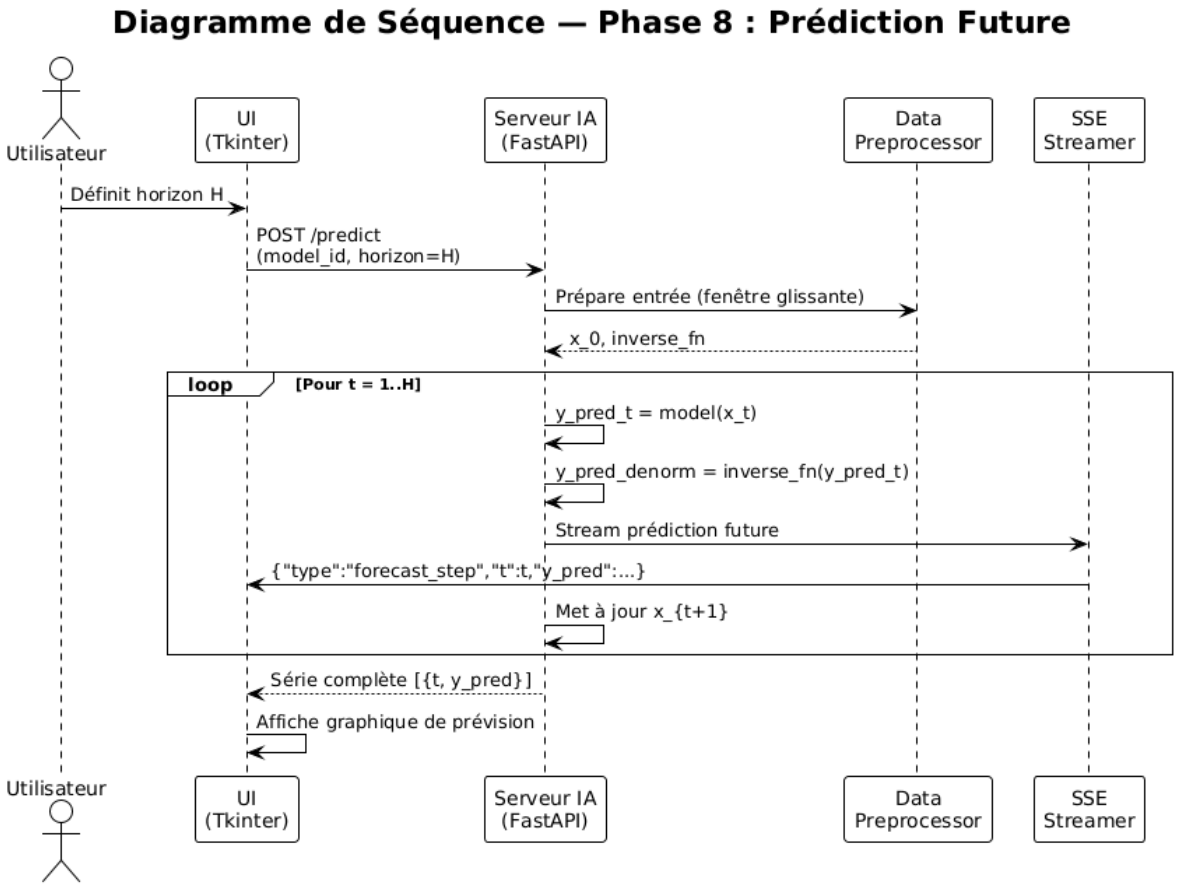


FIGURE 5.8 – Phase 8 — Prédiction future sur un horizon défini

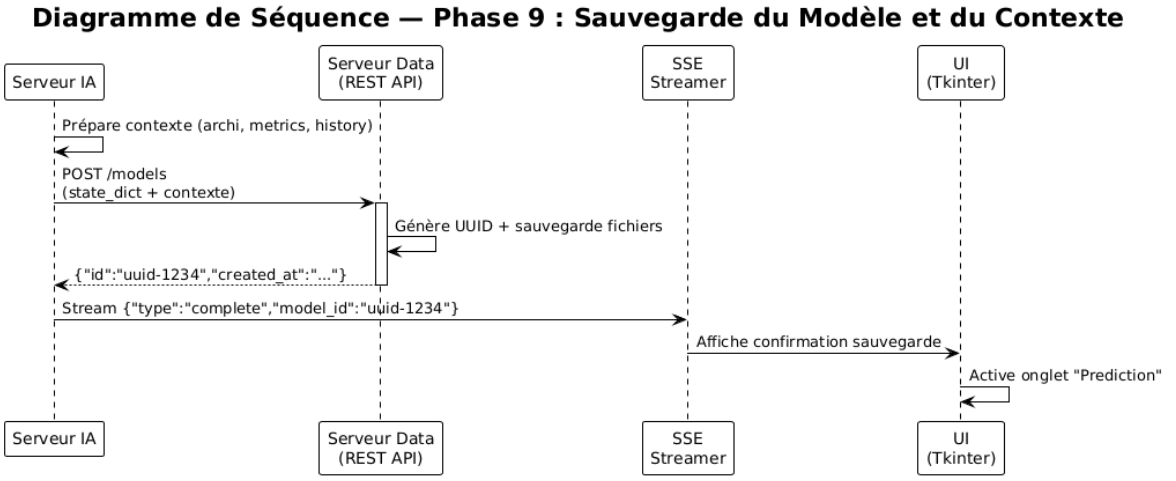


FIGURE 5.9 – Phase 9 — Sauvegarde du modèle et du contexte complet dans le Serveur Data

Ces neuf diagrammes de séquence détaillent le comportement dynamique de l'application **MLApp**. Ils montrent :

— les interactions entre les trois couches principales : **UI**, **Serveur IA** et **Serveur Data** ;

— la gestion progressive des flux de données et des événements SSE ;

— la séparation logique entre préparation, apprentissage, test et sauvegarde.