Cahier des charges – Application de prédiction de ventes

Équipe Projet GL

16 janvier 2026

Table des matières

1 Introduction

1.1 Contexte du Projet

Dans un contexte économique où l'optimisation des ressources est cruciale, les entreprises cherchent à anticiper leurs volumes de ventes avec précision. Ce projet vise à développer une application de bureau (basée sur des technologies Web packagées) permettant aux opérateurs métier et aux data analysts de configurer, entraîner et évaluer des modèles de Deep Learning (réseaux de neurones) sur des séries temporelles.

L'application doit abstraire la complexité du code Python sous-jacent (PyTorch/TensorFlow) via une interface graphique intuitive, offrant un retour visuel en temps réel lors de l'entraînement et des outils d'analyse post-entraînement.

1.2 Méthodologie de Conception

Le projet suit un cycle de développement itératif. L'architecture logicielle repose sur une séparation claire entre le Frontend (Interface Utilisateur) et le Backend (Calculs et Données).

1.2.1 Approche Orientée Objet

Bien que le Frontend utilise une approche fonctionnelle moderne (React Hooks), la modélisation des données et l'architecture globale suivent les principes de conception orientée objet pour assurer la modularité :

— Encapsulation des états (Store global).

— Ségrégation des interfaces (Composants UI distincts pour la configuration, la visualisation, le contrôle).

— Abstraction des services API (DatasetAPI, TrainingAPI).

2 Spécifications Fonctionnelles

2.1 F-1 : Gestion des Données

Le système doit permettre la manipulation des jeux de données temporels qui serviront de base à l'apprentissage.

2.1.1 F-1.1 : Importation de Données

L'utilisateur doit pouvoir charger des données depuis une source externe.

— Le système interroge l'API `datasetAPI` pour récupérer les métadonnées disponibles.

— Gestion des erreurs si le format du fichier source est invalide.

2.1.2 F-1.2 : Sélection de Dataset

L'interface doit présenter une liste des datasets disponibles (ex : `DatasetList`).

— Affichage du nom du dataset.

— Validation obligatoire : l'entraînement ne peut démarrer sans sélection active.

2.1.3 F-1.3 : Filtrage Temporel

L'utilisateur doit pouvoir restreindre la plage de données utilisée.

— Sélection de la date de début et de fin.

— Définition du pas temporel (ex : journalier, hebdomadaire).

— Découpage automatique ou manuel en ensembles d'entraînement, de validation et de test.

2.2 F-2 : Configuration du Modèle

Le cœur de l'application réside dans la personnalisation de l'architecture du réseau de neurones.

2.2.1 F-2.1 : Choix du Type de Réseau

L'utilisateur sélectionne la famille de modèle via un composant `ModelSelector`.

— Types supportés : MLP (Perceptron Multicouche), CNN (Réseau Convolutif), LSTM (Long Short-Term Memory).

— Mise à jour dynamique des paramètres affichés selon le type choisi.

2.2.2 F-2.2 : Configuration Architecture MLP

Si MLP est sélectionné :

— Définition du nombre de couches cachées.

— Définition de la taille des couches (`hidden_size`).

— Paramètre de `dropout` et fonction d'activation (ReLU, Tanh, Sigmoid).

2.2.3 F-2.3 : Configuration Architecture CNN

Si CNN est sélectionné :

— Paramètres spécifiques aux convolutions (taille du noyau, nombre de filtres).

— Paramètres de pooling.

2.2.4 F-2.4 : Configuration Architecture LSTM

Si LSTM est sélectionné :

— Nombre de couches récurrentes (`num_layers`).

— Taille de l'état caché (`hidden_size`).

— Bidirectionnalité (optionnel).

2.3 F-3 : Configuration de l'Entraînement

L'utilisateur doit pouvoir ajuster les hyperparamètres régissant l'algorithme d'apprentissage.

2.3.1 F-3.1 : Paramètres Temporels

Définition de l'horizon de prédiction (combien de pas de temps dans le futur) et de la fenêtre d'observation (lookback).

2.3.2 F-3.2 : Fonction de Perte

Sélection de la métrique à minimiser :
— MSE (Mean Squared Error).
— MAE (Mean Absolute Error).
— Huber Loss (robuste aux outliers).

2.3.3 F-3.3 : Optimiseur

Choix de l'algorithme d'optimisation (Adam, SGD, RMSprop, Adagrad, Adadelta) et configuration du Scheduler (Plateau, Cosine, OneCycle) pour l'ajustement dynamique du taux d'apprentissage.

2.3.4 F-3.4 : Paramètres d'Entraînement

Réglage des paramètres scalaires :
— Nombre d'époques (`nb_epochs`).
— Taille du lot (`batch_size`).
— Taux d'apprentissage initial (`learning_rate`).
— `clip_gradient` pour éviter l'explosion du gradient.

2.4 F-4 : Entraînement et Monitoring

Le système doit fournir un contrôle total et une visibilité sur le processus d'entraînement.

2.4.1 F-4.1 : Lancement de l'Entraînement

Action déclenchée par un bouton unique. Le système doit vérifier la cohérence de la configuration avant d'envoyer la requête au backend via `trainingAPI.startTraining`.

2.4.2 F-4.2 : Visualisation en Temps Réel

L'interface doit afficher la courbe de perte (`TrainingChart`) mise à jour dynamiquement à chaque époque reçue via le flux d'événements (SSE).
— Réception de l'événement `epoch` contenant `avg_loss`.
— Mise à jour du graphique sans rechargement de page.

2.4.3 F-4.3 : Annulation de l'Entraînement

Possibilité d'interrompre le processus à tout moment via un bouton "Arrêter". Le système doit envoyer une requête d'annulation au backend et réinitialiser l'état de l'interface.

2.5 F-5 : Evaluation et Test

Une fois l'entraînement terminé, le modèle doit être évalué.

2.5.1 F-5.1 : Phase de Test Automatique

À la fin du pipeline (`fin_pipeline`), le système affiche les résultats sur le jeu de test (`TestingChart`).
— Superposition de la série réelle et des prédictions.
— Affichage des intervalles de confiance (IC low/high).
— Outils d'interaction : Zoom X/Y, Tooltip, Légende.

2.5.2 F-5.2 : Métriques Calculées

Affichage des indicateurs de performance reçus du backend :

— Métriques de validation et de prédiction.

— Moyenne globale (`overall_mean`).

2.6 F-6 : Prédiction Future

2.6.1 F-6.1 : Prédiction sur Horizon Défini

Le système doit permettre de générer des prédictions au-delà des données connues (forecast), basées sur l'horizon configuré.

2.7 F-7 : Sauvegarde et Chargement

2.7.1 F-7.1 : Sauvegarde du Modèle et du Contexte

L'utilisateur doit pouvoir sauvegarder la configuration actuelle (hyperparamètres) et potentiellement le modèle entraîné pour une réutilisation ultérieure.

2.7.2 F-7.2 : Chargement d'un Modèle Existant

Capacité de recharger une configuration précédente pour reproduire un entraînement ou affiner un modèle.

3 Spécifications Non-Fonctionnelles

3.1 Performance

3.1.1 Utilisation des Ressources

L'application cliente (Electron) doit rester réactive même lors de la réception de flux de données rapides. Le rendu des graphiques (Recharts) doit être optimisé pour gérer des séries de plusieurs milliers de points sans latence perceptible.

3.2 Fiabilité

3.2.1 Gestion des Erreurs

Le système doit gérer gracieusement les échecs de communication avec l'API (timeout, erreur 500). Des messages d'erreur clairs doivent être affichés à l'utilisateur (via des composants `Alert`).

3.3 Utilisabilité

3.3.1 Expérience Utilisateur

L'interface doit être intuitive, avec un thème cohérent (support du mode sombre/clair via TailwindCSS). Les contrôles doivent fournir un feedback visuel immédiat (états loading, disabled).

3.4 Sécurité

3.4.1 Protection des Données

Bien que projet scolaire, l'application ne doit pas exposer de données sensibles dans les logs ou via des canaux non sécurisés. Les communications API doivent être robustes.

4 Spécification Architecturale

4.1 Architecture Globale

4.1.1 Vue d'Ensemble

L'architecture suit un modèle Client-Serveur découplé.

— **Client** : Application Desktop cross-platform (Windows/Mac/Linux).

— **Serveur** : API RESTful et Streaming pour le calcul intensif.

4.2 Composants Principaux

4.2.1 Interface Utilisateur (Frontend)

Développée en **React** avec **TypeScript**, packagée via **Electron**.

— **Gestion d'état** : Zustand (Store global pour la config et les données).

— **Visualisation** : Recharts (Graphiques interactifs).

— **Style** : TailwindCSS.

4.2.2 Serveur IA (Backend)

Développé en **Python** (FastAPI).

— Responsable de l'instanciation des modèles (PyTorch/TensorFlow).

— Gère la boucle d'entraînement et le streaming des événements SSE.

4.2.3 Serveur Data

Module responsable de la lecture, du nettoyage et de la mise à disposition des datasets (fichiers CSV/JSON) via l'API `datasetAPI`.

5 Spécification Détaillée

5.1 Diagramme de Classe UI

Le diagramme suivant illustre la structure des composants React et du Store Zustand.

```
classDiagram
  class UseStore {
    +config: GlobalConfig
    +modelConfig: ModelConfig
    +isTraining: boolean
    +trainingData: EpochLoss[]
    +testingData: FinalPlotData
    +metrics: MetricsResponse
    +startTraining()
    +stopTraining()
    +addTrainingPoint()
    +setTestingData()
    +setMetrics()
  }
  class TrainingControl {
    +handleStart()
```

```
    +handleStop()
  }
  class TrainingChart {
    +render(lossSeries)
  }
  class TestingChart {
    +render(testingData)
    +zoomXY()
  }
  class ModelSelector
  class NetworkArchitecture
  class TrainingParams

  UseStore o-- TrainingControl
  UseStore o-- TrainingChart
  UseStore o-- TestingChart
  UseStore o-- ModelSelector
```

5.2 Diagramme de Classe Serveur IA

Représentation simplifiée des classes Python gérant les modèles.

```
classDiagram
    class ModelFactory {
        +create_model(type, config)
    }
    class Trainer {
        +train(model, data, params)
        +stream_events()
    }
    class NeuralNet {
        <<Abstract>>
        +forward()
    }
    class MLP
    class CNN
    class LSTM

    NeuralNet <|-- MLP
    NeuralNet <|-- CNN
    NeuralNet <|-- LSTM
    ModelFactory ..> NeuralNet
    Trainer --> NeuralNet
```

5.3 Diagrammes de Séquence - Entraînement Complet

Ce diagramme détaille les interactions entre l'utilisateur, l'UI, le Store et l'API lors d'une session d'entraînement.

```
sequenceDiagram
  participant User
  participant UI as Front React
  participant Store as useStore
```

```
participant TrainAPI as trainingAPI
participant Backend as Python Server

User->>UI: Clic "Lancer"
UI->>Store: startTraining() (reset data)
UI->>TrainAPI: startTraining(config)
TrainAPI->>Backend: POST /train_full

alt Erreur Dataset
  Backend-->>TrainAPI: event: error
  TrainAPI-->>UI: alert("Erreur chargement")
end

loop Streaming SSE
  Backend-->>TrainAPI: event: epoch {loss}
  TrainAPI-->>UI: callback onEvent(epoch)
  UI->>Store: addTrainingPoint()
  Store-->>UI: update TrainingChart
end

Backend-->>TrainAPI: event: fin_pipeline
TrainAPI-->>UI: callback onComplete()
UI->>Store: setTestingData()
Store-->>UI: update TestingChart
```