Documentation Technique - API de Prédiction d'Attrition

Projet 5 - Parcours Data Scientist

Auteur : Anh Tuan KIEU

Date : Février 2025

Version : 1.0.0

---

1. Présentation du Projet

1.1 Contexte

TechNova Partners, une ESN spécialisée en transformation digitale, fait face à un turnover élevé. Au Projet 4, un modèle de classification a été construit pour prédire le risque de départ des employés. Le Projet 5 consiste à déployer ce modèle en production via une API REST et une interface web.

1.2 Objectifs

- Exposer le modèle via une API documentée (Swagger/OpenAPI)
- Assurer la traçabilité des prédictions en base de données
- Automatiser le déploiement via un pipeline CI/CD
- Fournir une interface utilisateur intuitive pour les équipes RH

---

2. Le Modèle de Machine Learning

2.1 Description

Propriété	Valeur
Type	LogisticRegression (Scikit-learn)

Propriété	Valeur
Format	Pipeline pickle (`lr_pipeline.pkl`)
Date d'export	2026-02-05
Taille du dataset	1 470 employés
Split	Train: 1 176 (80%) / Test: 294 (20%)
Variable cible	`attrition` (Oui/Non)
Déséquilibre	83.9% Non / 16.1% Oui

2.2 Hyperparamètres

Paramètre	Valeur	Justification
`class_weight`	`balanced`	Compense le déséquilibre des classes (pénalise 5x les erreurs sur la classe minoritaire)
`max_iter`	1000	Convergence garantie
`random_state`	42	Reproductibilité

2.3 Performances (jeu de test)

Métrique	Valeur	Interprétation
AUC-ROC	81.6%	Bonne capacité de discrimination
Recall	70.2%	Détecte 7 départs sur 10
Precision	37.1%	1 alerte sur 3 est un vrai départ
F1-Score	48.5%	Compromis precision/recall
Accuracy	76.2%	Performance globale

Choix du recall comme métrique prioritaire : dans le contexte RH, un faux négatif (employé à risque non détecté) coûte entre 6 et 9 mois de salaire. Un faux positif (fausse alerte) ne coûte qu'un entretien RH supplémentaire.

2.4 Pipeline de preprocessing

Le modèle est un pipeline Scikit-learn intégrant le preprocessing :

```
Pipeline
  |-- preprocessor (ColumnTransformer)
  |     |-- num: StandardScaler (27 features numériques)
  |     |-- cat: OneHotEncoder(drop='first') (8 features catégorielles)
  |-- classifier: LogisticRegression(class_weight='balanced')
```

Après encodage : 48 features (one-hot encoding des variables catégorielles).

2.5 Top 5 des facteurs d'attrition (SHAP)

Rang	Feature	Impact SHAP	Interprétation
1	`heure_supplementaires`	0.6516	Risque x3 si heures sup
2	`frequence_deplacement` (Fréquent)	0.5153	Déplacements fréquents = risque accru
3	`ratio_poste_entreprise`	0.4797	Stagnation dans le poste
4	`annees_dans_le_poste_actuel`	0.4760	Peu d'ancienneté = risque
5	`duree_moyenne_poste`	0.4594	Mobilité professionnelle élevée

---

3. Feature Engineering

3.1 Features calculées (5 variables)

Le modèle utilise 30 features brutes + 5 features calculées = 35 features au total.

Les features dérivées sont calculées **côté serveur** dans le module `app/feature_engineering.py`. Le client n'a pas besoin de les calculer.

Feature	Formule	Signification
`ratio_poste_entreprise`	`années_poste / (années_entreprise + 1)`	Stagnation dans le poste. Ratio proche de 1 = même poste depuis l'arrivée

Feature	Formule	Signification
`evolution_evaluation`	`note_actuelle - note_précédente`	Progression ou régression de la performance. Négatif = régression
`satisfaction_globale`	`moyenne(4 scores de satisfaction)`	Score synthétique de bien-être au travail
`salaire_par_experience`	`revenu / (expérience + 1)`	Détecte les employés sous-payés par rapport à leur profil
`duree_moyenne_poste`	`expérience / (nb_postes + 1)`	Mesure la mobilité professionnelle

Note technique : le `+1` au dénominateur évite les divisions par zéro pour les employés nouvellement arrivés (`années_entreprise = 0`).

3.2 Features supprimées (redondantes)

Feature	Raison de suppression
`faible_satisfaction`	Seuil de `satisfaction_globale` — le modèle peut apprendre ce seuil
`surcharge_travail`	Identique à `heure_supplementaires` après one-hot encoding

---

4. Architecture Technique

4.1 Vue d'ensemble

```
Frontend (Vercel)          Backend (HF Spaces)          Database (Neon)
+------------------+       +----------------------+       +----------------+
| React + Vite     | ----> | FastAPI              | ----> | PostgreSQL     |
| Interface RH     | JSON  | Feature Engineering  | SQL   | employees      |
|                  | <---- | ML Model (pickle)    | <---- | predictions    |
+------------------+       +----------------------+       +----------------+
```

4.2 Choix techniques

Composant	Technologie	Justification
Frontend	React + Vite	Build rapide, hot reload, écosystème riche
Backend	FastAPI	Performance async, validation Pydantic, Swagger auto-généré
Database	PostgreSQL (Neon)	Données structurées, relations, SQL standard, tier gratuit serverless
Hébergement API	Hugging Face Spaces	Gratuit, optimisé ML, support Docker natif
Hébergement Frontend	Vercel	Gratuit, CDN mondial, déploiement automatique
CI/CD	GitHub Actions	Intégré à GitHub, workflows YAML, secrets chiffrés
Conteneurisation	Docker	Reproductibilité, isolation, déploiement simplifié

4.3 Niveaux de risque

L'API attribue un niveau de risque basé sur la probabilité de départ :

Probabilité	Niveau	Action suggérée
< 20%	**low** (faible)	Suivi standard
20% – 45%	**medium** (moyen)	Entretien préventif
>= 45%	**high** (élevé)	Action urgente

Les seuils sont conservateurs (inférieurs au seuil standard de 50%) car le taux d'attrition de base est de 16%. Un employé à 20% de probabilité a déjà un risque supérieur à la moyenne.

---

5. API — Documentation des Endpoints

5.1 Base URL

- **Production** : `https://passkey1510-employee-attrition-api.hf.space`
- **Documentation Swagger** : `https://passkey1510-employee-attrition-api.hf.space/docs`
- **Documentation ReDoc** : `https://passkey1510-employee-attrition-api.hf.space/redoc`

5.2 Endpoints

Méthode	Endpoint	Description
GET	`/`	Informations de l'API
GET	`/health`	Health check (statut + modèle chargé)
POST	`/predict`	Prédiction pour un employé (30 champs)
POST	`/predict/batch`	Prédictions en lot
GET	`/employees`	Liste paginée des employés (`?skip=0&limit=100`)
GET	`/employees/{id}`	Détail d'un employé
GET	`/employees/{id}/predict`	Prédiction pour un employé en base
GET	`/predictions`	Historique des prédictions
GET	`/predictions/{id}`	Détail d'une prédiction
GET	`/model/info`	Métadonnées du modèle
GET	`/model/features`	Liste des features

5.3 Exemple d'utilisation — POST /predict

Requête :

```
curl -X POST "https://passkey1510-employee-attrition-api.hf.space/predict" \
  -H "Content-Type: application/json" \
  -d '{
    "genre": "M",
    "statut_marital": "Marie(e)",
    "departement": "Commercial",
```

```
    "poste": "Representant Commercial",
    "heure_supplementaires": "Non",
    "augementation_salaire_precedente": 0.13,
    "domaine_etude": "Infra & Cloud",
    "ayant_enfants": "Y",
    "frequence_deplacement": "Occasionnel",
    "age": 35,
    "revenu_mensuel": 5000,
    "nombre_experiences_precedentes": 2,
    "nombre_heures_travailless": 80,
    "annee_experience_totale": 10,
    "annees_dans_l_entreprise": 5,
    "annees_dans_le_poste_actuel": 3,
    "satisfaction_employee_environnement": 3,
    "note_evaluation_precedente": 3,
    "niveau_hierarchique_poste": 2,
    "satisfaction_employee_nature_travail": 4,
    "satisfaction_employee_equipe": 3,
    "satisfaction_employee_equilibre_pro_perso": 3,
    "note_evaluation_actuelle": 4,
    "nombre_participation_pee": 1,
    "nb_formations_suivies": 3,
    "nombre_employee_sous_responsabilite": 0,
    "distance_domicile_travail": 10,
    "niveau_education": 3,
    "annees_depuis_la_derniere_promotion": 1,
    "annes_sous_responsable_actuel": 3
  }'
```

Réponse :

```
{
  "prediction_id": 1,
  "result": {
    "prediction": 0,
    "probability": 0.1542,
    "risk_level": "low",
    "attrition_label": "Non"
  },
  "engineered_features": {
    "ratio_poste_entreprise": 0.5,
    "evolution_evaluation": 1,
    "satisfaction_globale": 3.25,
    "salaire_par_experience": 454.55,
    "duree_moyenne_poste": 3.33
  },
```

```
  "timestamp": "2026-02-05T16:45:00"
}
```

5.4 Validation des données

La validation est assurée par Pydantic avec des contraintes métier :

Champ	Contrainte	Message d'erreur
`age`	18 – 70	L'âge doit être compris entre 18 et 70 ans
`revenu_mensuel`	>= 0	Le revenu mensuel doit être positif
`satisfaction_*`	1 – 5	Les scores de satisfaction sont entre 1 et 5
`niveau_education`	1 – 5	Le niveau d'éducation est entre 1 et 5
`heure_supplementaires`	Oui/Non	Valeur attendue : Oui ou Non

Les erreurs de validation retournent un code HTTP 422 avec un message en français.

---

6. Base de Données

6.1 Schéma

```
employees                          predictions
+---------------------------------+  +---------------------------------+
| id (PK, auto)                   |  | id (PK, auto)                   |
| employee_id (UNIQUE)            |<-| employee_id (FK, nullable)      |
| genre                           |  | input_data (JSONB)              |
| age                             |  | prediction (INTEGER, 0 ou 1)    |
| statut_marital                  |  | probability (FLOAT)             |
| departement                     |  | risk_level (VARCHAR)            |
| poste                           |  | model_version (VARCHAR)         |
| revenu_mensuel                  |  | created_at (TIMESTAMP, auto)    |
| ... (30 features au total)      |  +---------------------------------+
| attrition_actual (VARCHAR)      |
| dataset_type (train/test)       |
+---------------------------------+
```

6.2 Relation nullable

La clé étrangère `employee_id` dans `predictions` est **nullable** pour supporter deux scénarios :

- **Employé existant** : `employee_id` pointe vers la table `employees` (prédiction depuis la liste)
- **Saisie manuelle** : `employee_id = NULL` (prédiction pour un candidat ou simulation)

6.3 Traçabilité

Chaque prédiction est enregistrée avec :

- Les données d'entrée complètes (`input_data` en JSONB)
- Le résultat de la prédiction
- La version du modèle utilisée
- Un timestamp automatique

Cela permet l'audit, l'analyse rétrospective, et la comparaison de performances entre versions du modèle.

---

7. Tests et Qualité du Code

7.1 Couverture

Module	Lignes	Couverture	Rôle
`schemas.py`	73	**100%**	Validation des données
`feature_engineering.py`	37	**100%**	Calcul des features
`database.py`	77	**94%**	Accès base de données
`model.py`	59	**93%**	Chargement et prédiction
`main.py`	103	77%	Endpoints API
TOTAL	**350**	**91%**	

7.2 Types de tests

Tests unitaires (`tests/test_model.py`) :

- Validation Pydantic : rejet des valeurs hors bornes
- Feature engineering : vérification des formules
- Chargement du modèle : pipeline pickle valide

Tests fonctionnels (`tests/test_api.py`) :

- POST /predict : prédiction valide, profil à haut risque
- GET /employees : liste paginée
- GET /predictions : historique
- Batch predictions

7.3 Exécution

```
# Tous les tests
pytest tests/ -v

# Avec couverture
pytest tests/ --cov=app --cov-report=term-missing
```

---

8. CI/CD et Déploiement

8.1 Pipeline

```
Push to main
    |
    +-- Path filter : backend modifié ?
    |       |
    |       +-- pytest (tests)
    |       |       |
    |       |       +-- PASS --> Build Docker --> Deploy HF Spaces
    |       |       |
    |       |       +-- FAIL --> Déploiement bloqué
    |
    +-- Path filter : frontend modifié ?
            |
            +-- Build Vite --> Deploy Vercel
```

8.2 Gestion des secrets

Les secrets sont gérés via **GitHub Actions Secrets** :

Secret	Usage
`DATABASE_URL`	Connection string PostgreSQL (Neon)
`HF_TOKEN`	Token Hugging Face pour le déploiement

Flux : GitHub Secrets → GitHub Actions → Variables d'environnement HF Spaces

Les secrets ne sont **jamais** dans le code source.

8.3 Docker

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 7860
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "7860"]
```

8.4 URLs de production

Service	URL
Frontend	https://employee-attrition-frontend-ten.vercel.app
API Backend	https://passkey1510-employee-attrition-api.hf.space
Swagger UI	https://passkey1510-employee-attrition-api.hf.space/docs

---

9. Installation et Configuration

9.1 Prérequis

- Python 3.12+
- Docker (optionnel, pour PostgreSQL local)
- Compte Neon (base de données) ou PostgreSQL local

9.2 Installation locale

```
# Cloner le dépôt
git clone https://github.com/passkey1510/employee-attrition-api
cd project-5

# Installer les dépendances
pip install -r requirements.txt

# Configurer l'environnement
cp .env.example .env
# Éditer .env avec votre DATABASE_URL
```

9.3 Démarrage

```
# Option 1 : Docker Compose (recommandé)
docker compose up -d

# Option 2 : Manuel
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

9.4 Seeding de la base

```
python scripts/seed_db.py
```

Charge les 1 470 employés du dataset dans la table `employees`.

9.5 Dépendances principales

Package	Version	Rôle
fastapi	>= 0.109	Framework API
uvicorn	>= 0.27	Serveur ASGI

Package	Version	Rôle
pydantic	>= 2.5	Validation des données
scikit-learn	1.6.1	Pipeline ML (version fixée pour compatibilité pickle)
pandas	>= 2.1	Manipulation des données
sqlalchemy	>= 2.0	ORM base de données
psycopg2-binary	>= 2.9	Driver PostgreSQL
pytest	>= 8.0	Framework de tests

Important : la version de `scikit-learn` est fixée à `1.6.1` pour garantir la compatibilité avec le fichier pickle exporté au Projet 4.

---

10. Protocole de Maintenance et Mise à Jour

10.1 Mise à jour du modèle

1. **Entraîner** un nouveau modèle dans un notebook (mêmes features, même pipeline)
2. **Exporter** le pipeline en pickle : `joblib.dump(pipeline, 'models/lr_pipeline.pkl')`
3. **Mettre à jour** `models/model_metadata.json` avec les nouvelles métriques
4. **Committer** et pusher vers `main`
5. Le pipeline CI/CD **déploie automatiquement** la nouvelle version

Attention : vérifier que la version de `scikit-learn` utilisée pour l'entraînement est compatible avec celle en production (`1.6.1`).

10.2 Ajout de nouvelles features

1. Ajouter la feature dans `app/feature_engineering.py`
2. Mettre à jour `models/features.json`
3. Mettre à jour le schéma Pydantic dans `app/schemas.py` (si nouveau champ en entrée)

4. Ajouter des tests dans `tests/test_model.py`
5. Ré-entraîner le modèle avec les nouvelles features

10.3 Monitoring recommandé

Quoi surveiller	Comment	Fréquence
Latence API	Logs HF Spaces	Hebdomadaire
Distribution des prédictions	Table `predictions`	Mensuel
Data drift	Comparer input actuel vs dataset d'entraînement	Trimestriel
Performances modèle	Ré-évaluer sur nouvelles données	Semestriel

10.4 Ré-entraînement

Le modèle doit être ré-entraîné si :

- Le taux de prédictions correctes baisse significativement
- De nouvelles variables sont disponibles (ex : données de formations, mobilité interne)
- Le dataset s'enrichit de nouveaux employés
- Un data drift est détecté (les caractéristiques des employés changent)

10.5 Sécurité (recommandations production)

Pour un déploiement en production réelle :

- Ajouter une authentification JWT sur les endpoints
- Implémenter du rate limiting (ex : 100 requêtes/minute)
- Forcer HTTPS
- Ajouter des logs structurés (ex : logging JSON)
- Mettre en place des alertes sur les erreurs 5xx

---

11. Structure du Projet

```
project-5/
|-- app/
|   |-- __init__.py                # Version de l'API
|   |-- main.py                    # Endpoints FastAPI
|   |-- schemas.py                 # Schémas Pydantic (validation)
|   |-- database.py                # Modèles SQLAlchemy + fonctions DB
|   |-- model.py                   # Chargement et prédiction ML
|   |-- feature_engineering.py     # Calcul des 5 features dérivées
|-- models/
|   |-- lr_pipeline.pkl            # Pipeline entraîné (pickle)
|   |-- features.json              # Liste des features
|   |-- model_metadata.json        # Métriques et hyperparamètres
|-- data/
|   |-- employees.csv              # Dataset (1 470 employés)
|-- tests/
|   |-- conftest.py                # Fixtures pytest
|   |-- test_api.py                # Tests fonctionnels
|   |-- test_model.py              # Tests unitaires
|-- scripts/
|   |-- seed_db.py                 # Seeding de la base
|-- frontend/                      # Application React (Vercel)
|-- .github/workflows/ci.yml       # Pipeline CI/CD
|-- Dockerfile                     # Image Docker
|-- docker-compose.yml             # Orchestration locale
|-- requirements.txt               # Dépendances Python
|-- README.md                      # Documentation du repo
```