ULTRA-ENHANCED PPM – Recommandation Vidéo

Modèle séquentiel avancé pour la prédiction d'interactions utilisateur–contenu (AUC : 0.645)

Équipe

- **Nom de l’équipe** : Sahel Data Innovators – Niger
- **Membres** : Aichatou Boubacar Soumana , Abdourahamane Ide Salifou

---

1. Contexte et objectif

Ce projet a été réalisé dans le cadre de la **finale Internationale du DataTour Afrique 2025**, dédiée à la **recommandation de contenus vidéo sur plateformes sociales**.

Le but est de prédire la **probabilité qu'un utilisateur interagisse** (label binaire) avec une vidéo qui lui est proposée dans un futur proche, en exploitant :

- Les caractéristiques des utilisateurs (âge, genre, historique d'engagement).
- Les propriétés des vidéos (durée, source, embeddings de contenu).
- L'historique d'interactions passées.
- Des signaux contextuels (temps, type de contenu, etc.).

La métrique d'évaluation est l'**AUC ROC**, standard pour les problèmes de classification déséquilibrée et de ranking probabiliste.

Notre objectif : développer un modèle performant, robuste et cohérent avec les pratiques industrielles des systèmes de recommandation modernes (YouTube, TikTok, Netflix, etc.).

Soumission et reproductibilité

Le notebook principal s’exécute sur **GPU NVIDIA T4** avec un temps d’entraînement d’environ **~3h10** (1 epoch) pour Ultra Enhanced PPM.

2. Vue d'ensemble de l'approche

Notre approche repose sur une architecture **hybride, profonde et séquentielle**, inspirée des meilleurs modèles de recommandation industriels (YouTube DNN, architectures PPM/Next-Item, DCN, FiBiNet) :

Architecture ULTRA-ENHANCED PPM

Le modèle combine plusieurs composantes avancées :

1. **Embeddings d'entités multiples** :
    - Utilisateur, vidéo, source/canal, âge, genre, durée.

    - Embeddings séparés pour les branches Wide et Deep.

2. **Module séquentiel Multi-Head PPM** :
    - GRU multi-couches pour capturer la dynamique temporelle.
    - 4 têtes d'attention pour diversifier les patterns appris.
    - Fusion des représentations pour générer un vecteur contextualisé de l'historique utilisateur.

3. **Interactions bilinéaires FiBiNet** :
    - 21 interactions bilinéaires entre tous les couples de features (C(7,2)).
    - Capture des patterns de second ordre (ex : "utilisateur × catégorie", "âge × source").

4. **SENet (Squeeze-and-Excitation)** :
    - Pondération dynamique de l'importance des différentes composantes.
    - Adapte automatiquement le modèle au contexte de chaque prédiction.

5. **Deep & Cross Network (DCN)** :
    - 4 couches cross pour modéliser des interactions explicites de haut niveau.
    - MLP profond avec architecture .
    - Fusion finale Wide + Cross + Deep.

6. **Features statistiques avancées** :
    - Niveau vidéo : CTR, popularité, écart-type d'engagement, taux de conversion.
    - Niveau utilisateur : activité, engagement, affinités source/contexte.

Philosophie du modèle : Apprendre un score de propension à l'interaction pour chaque paire `(user, video)` en combinant profil statique, historique séquentiel et relations complexes entre dimensions.

---

3. Données et prétraitement

3.1. Organisation des fichiers

Le challenge fournit :

- **Fichier d'entraînement** : paires `(utilisateur, contenu)` avec label d'interaction binaire.
- **Fichier de test** : même structure sans labels.
- **Fichier d'exemple de soumission** : format attendu (`id`, `target`).
- **Métadonnées** : `items_meta.parquet` (vidéos) et `users_meta.parquet` (utilisateurs).

Les chemins sont paramétrables via le dictionnaire `CONFIG` :

```
CONFIG = {
    'data_folder': '/kaggle/input',
    'output_folder': '/kaggle/working/',
    'train_path': '/interactions-data/train_interactions.parquet',
    'test_path': '/new-test-dataset/test_pairs_public.parquet',
    'items_meta_path': '/interactions-data/items_meta.parquet',
    'users_meta_path': '/interactions-data/users_meta.parquet',
    ...
}
```

3.2. Construction de la cible

La cible est un label binaire construit à partir de plusieurs signaux d'interaction :

```
train['target'] = ((train['int1'] + train['int2'] + train['int3'] +
train['int4']) > 0).astype('int8')
```

- `target = 1` : interaction positive (clic, vue complète, engagement).
- `target = 0` : pas d'interaction ou interaction faible.

Distribution : Dataset fortement déséquilibré (~5.4% de positifs), d'où l'importance de l'AUC ROC comme métrique.

3.3. Normalisation et encodage des métadonnées

Variables catégorielles encodées en entiers puis transformées en embeddings :

- `user_id`, `item_id` (vidéo), `source_id`, `age`, `gender`, `duration`.

Variables continues prétraitées :

- Recentrage (ex : `age = age - 18`, `duration = duration - 5`).
- Log-transformation pour les distributions asymétriques.
- Troncature par percentiles (1%–99%) pour réduire l'impact des outliers.
- Normalisation dans.[1]

Embeddings pré-calculés :

- Embeddings de contenu vidéo (dimension 32, issus de modèles pré-entraînés).
- Projetés dans l'espace commun (192D) via couche linéaire.

3.4. Séquences utilisateur

Point clé de l'approche : modélisation séquentielle de l'historique de visionnage.

Pour chaque utilisateur :

1. Reconstruction de la séquence ordonnée temporellement des `video_id` consultés.
2. Stockage dans un dictionnaire `user_sequences[uid] = {'items': [...], 'targets': [...], 'length': ...}`.
3. Troncature/padding à une **longueur maximale de 20** (derniers contenus).
4. Création d'un **masque de séquence** pour distinguer positions réelles du padding.

Ces séquences alimentent le module Multi-Head PPM et donnent au modèle une vue contextualisée du comportement récent.

4. Feature engineering avancé

4.1. Statistiques niveau vidéo (item)

Agrégation par `item_id` sur le train :

- **Volume** : `item_total_count` (nombre d'impressions).
- **Engagement** : `item_pos_count` (nombre d'interactions positives).
- **Taux de clic** : `item_ctr = pos_count / total_count`.
- **Variabilité** : `item_ctr_std` (écart-type du CTR).
- **Popularité** : `item_popularity = log(1 + total_count)`.
- **Conversion** : `item_conversion_rate = pos_count / (total_count + 10)`.

Normalisation robuste :

- Troncature par percentiles (1%–99%).
- Normalisation dans.[1]
- Imputation des valeurs manquantes.

Ces 4 features sont projetées dans un espace latent via un MLP (`item_stat_proj`).

4.2. Statistiques niveau utilisateur

Agrégation par `user_id` sur le train :

- **Volume** : `user_total_count` (nombre de contenus vus).
- **Engagement** : `user_pos_count` (nombre d'interactions positives).
- **Taux d'engagement** : `user_ctr = pos_count / total_count`.
- **Variabilité** : `user_ctr_std` (écart-type du CTR).
- **Activité** : `user_activity = log(1 + total_count)`.
- **Propension** : `user_engagement_rate = pos_count / (total_count + 10)`.

Même normalisation robuste que pour les items.

Ces 6 features distinguent par exemple :

- Utilisateur **très actif mais peu engagé** (consomme beaucoup, clique peu).
- Utilisateur **peu actif mais très engagé** (consomme peu, clique souvent).

4.3. Affinités utilisateur–source/contexte

Modélisation des préférences fines par source de contenu :

1. Jointure `train` × `items_meta` pour obtenir `source_id`.
2. Agrégation par `(user_id, source_id)` : calcul de `us_affinity = us_pos / (us_total + 5)`.
3. Agrégation par utilisateur :
    - `user_avg_source_affinity` : affinité moyenne sur toutes les sources.
    - `user_max_source_affinity` : affinité maximale (source préférée).

Ces signaux capturent par exemple qu'un utilisateur interagit fortement avec un type de créateur ou de catégorie spécifique.

Total : 10 features statistiques (4 item + 6 user) projetées dans des espaces latents de dimension 64.

---

5. Architecture détaillée du modèle

5.1. Embeddings (Wide & Deep)

Chaque entité possède **deux embeddings** :

- **Wide** : pour la branche linéaire et les interactions bilinéaires.
- **Deep** : pour les branches profondes (DCN, MLP).

Dimension uniforme : **192D** pour tous les embeddings.

Entités :

- `user_id` (183,404 utilisateurs).
- `item_id` (337,727 vidéos).
- `source_id` (sources de contenu).
- `age_id` (tranches d'âge).
- `gender_id` (2 classes).
- `duration_id` (durées de vidéo).
- `item_embeddings` pré-calculés (32D → 192D via projection linéaire).

Total : 7 champs catégoriels + 2 champs statistiques (item, user).

5.2. Module séquentiel Multi-Head PPM

Architecture :

```
Input: current_item_emb [batch, 192] + sequence_embs [batch, seq_len, 192] +
mask [batch, seq_len]
       ↓
[HEAD 1]   [HEAD 2]   [HEAD 3]   [HEAD 4]
   ↓          ↓          ↓          ↓
Encoder    Encoder    Encoder    Encoder  (MLP: 192 → 80)
   ↓          ↓          ↓          ↓
  GRU        GRU        GRU        GRU     (2 layers, hidden: 80)
   ↓          ↓          ↓          ↓
Attention  Attention  Attention  Attention (learnable attention weights)
   ↓          ↓          ↓          ↓
Concat current_enc + seq_repr [batch, 160] per head
       ↓
Concatenate all heads [batch, 640]
       ↓
Fusion MLP [640 → 320]
       ↓
Output: ppm_features [batch, 320]
```

Fonctionnement :

1. Chaque tête encode indépendamment la séquence d'historique.
2. Un GRU multi-couches capture la dynamique temporelle.

3. Un mécanisme d'attention pondère les positions selon leur pertinence.
4. La représentation agrégée est concaténée avec l'encodage de la vidéo cible.
5. Les 4 têtes sont fusionnées pour produire un vecteur contextuel riche.

Avantage : Les têtes multiples permettent d'apprendre des patterns d'attention diversifiés (court terme vs long terme, contenus similaires vs complémentaires, etc.).

5.3. SENet (Squeeze-and-Excitation)

```
Input: Stack de 7 embeddings deep [batch, 7, 192]
       ↓
Global Average Pooling (dim=2) → [batch, 7]
       ↓
MLP: [7 → 2 → 7]
       ↓
Sigmoid → attention_weights [batch, 7]
       ↓
Reweight: embeddings × attention_weights.unsqueeze(2)
       ↓
Output: Embeddings repondérés [batch, 7, 192]
```

Avantage : Adapte dynamiquement l'importance relative de chaque feature selon le contexte (ex : pour certaines prédictions, l'âge est plus important ; pour d'autres, c'est la source).

5.4. FiBiNet – Interactions bilinéaires

```
Input: Stack de 7 embeddings wide [batch, 7, 192]
       ↓
Pour chaque paire (i, j) avec i < j:
    interaction_ij = sum(emb_i^T × W × emb_j)
       ↓
Output: 21 interactions bilinéaires [batch, 21]
```

Matrice `W` apprise de dimension .

Avantage : Capture des interactions de second ordre riches (ex : "jeunes utilisateurs × vidéos courtes", "utilisateurs engagés × sources premium").

5.5. DCN (Deep & Cross Network) + MLP profond

Branche DCN :

```
Input: deep_concat [batch, dcn_input]
       ↓
x0 = deep_concat
for i in [0, 1, 2, 3]:
    x = x0 × (x @ W_i) + b_i + x
       ↓
Output: cross_out [batch, dcn_input]
```

4 couches cross permettent de modéliser des interactions explicites de haut niveau.

Branche Deep :

```
Input: deep_concat [batch, dcn_input]
       ↓
Linear(896) → BatchNorm → ReLU → Dropout(0.18)
       ↓
Linear(448) → BatchNorm → ReLU → Dropout(0.18)
       ↓
Linear(224) → BatchNorm → ReLU → Dropout(0.18)
       ↓
Output: deep_out [batch, 224]
```

Branche Wide :

```
Input: wide_concat [batch, wide_base]
       (contient embeddings wide + stats + ppm + bilinear)
       ↓
Linear(1) → wide_out [batch, 1]
```

Fusion finale :

```
combined = Concat(wide_out, cross_out, deep_out)
       ↓
BatchNorm [batch, final_dim]
       ↓
Linear(1) → logit [batch, 1]
       ↓
Sigmoid → probability
```

5.6. Dimensions du modèle

- **Nombre de paramètres** : **223,352,442** (~223M).
- **Dimension d'entrée DCN/Deep** : 1,632 (7×192 embeddings + 64+64 stats + 320 ppm).
- **Dimension Wide** : 1,632 + 21 interactions bilinéaires = 1,653.
- **Dimension finale** : 1 + 1,632 + 224 = 1,857 → Linear(1) → probabilité.

---

6. Stratégie d'entraînement

6.1. Hyperparamètres

```
CONFIG = {
    'emb_size': 192,
    'stat_emb_size': 64,
    'deep_layers': [896, 448, 224],
    'num_cross_layers': 4,
    'dropout': 0.18,
    'ppm_hidden_dim': 320,
    'ppm_num_layers': 2,
    'ppm_num_heads': 4,
    'sequence_length': 20,
    'BATCH_SIZE': 14336,
    'LR': 0.0022,
    'LR_MIN': 0.00003,
    'weight_decay': 6e-6,
    'EPOCHS': 1,
    'GRAD_CLIP': 1.0,
    'label_smoothing': 0.01,
}
```

6.2. Optimisation

- **Optimiseur** : AdamW avec weight decay (6e-6).
- **Scheduler** : CosineAnnealingLR (LR : 0.0022 → 0.00003).
- **Loss** : BCEWithLogitsLoss avec label smoothing (0.01).
- **Gradient clipping** : norme max = 1.0.
- **Régularisation** : Dropout (0.18), BatchNorm, Weight decay.

6.3. Validation temporelle

Split temporal pour respecter la nature séquentielle :

```
val_size = int(len(train) * 0.05)  # 5%
train_data = train.iloc[:-val_size]
val_data = train.iloc[-val_size:]
```

Évite les fuites d'information (ne pas utiliser le futur pour prédire le passé).

Métrique de validation : AUC ROC.

Meilleur modèle : sauvegardé à chaque amélioration de l'AUC validation.

6.4. Résultats d'entraînement

```
Epoch 1/1
Train Loss: 0.170403
Val AUC: 0.890975
```

Note : AUC de validation très élevée (0.89) car calculée sur le split interne binaire. L'AUC sur le leaderboard public (target finale) est de **0.64007**, ce qui est attendu et cohérent.

---

7. Génération de la soumission

7.1. Pipeline d'inférence

1. Rechargement du meilleur checkpoint (`ultra_enhanced_best.pth`).
2. Passage du test dans le même pipeline de preprocessing :
    - Extraction des métadonnées.
    - Calcul des features statistiques.
    - Génération des séquences utilisateur (historique du train).
3. Inférence par batch (mode `model.eval()`, `torch.no_grad()`).
4. Transformation des logits en probabilités via `torch.sigmoid`.

7.2. Calibration des probabilités

Une calibration simple est appliquée pour aligner la distribution prédite avec la distribution observée sur le train :

```
train_mean = train['target'].mean()  # ~0.054
pred_mean = np.mean(outputs_list)
calibration_factor = train_mean / (pred_mean + 1e-8)
outputs_list = np.clip(outputs_list * calibration_factor, 0.0, 1.0)
```

Avantage : Réduit le biais systématique et améliore la calibration des scores.

7.3. Format de soumission

```
submission = pd.DataFrame({
    'id': test['uid'].astype(str) + '_' + test['iid'].astype(str),
    'target': outputs_list
})
submission.to_parquet('submission.parquet', index=False)
```

---

8. Justification technique et apport du modèle

Cette architecture se situe dans la lignée des **systèmes de recommandation industriels de pointe** :

8.1. Inspirations industrielles

- **YouTube DNN** : Embeddings + Deep Network pour ranking.
- **FiBiNet (Alibaba)** : Interactions bilinéaires pour e-commerce.
- **DCN (Google)** : Cross layers pour interactions explicites.
- **SENet** : Attention sur features pour computer vision, adapté ici aux embeddings.
- **Multi-Head Attention** : Transformers, BERT, GPT (diversité des patterns).

8.2. Points forts

✅ **Modélisation séquentielle** : Capture la dynamique temporelle via GRU + attention multi-têtes.
✅ **Interactions riches** : Bilinéaires (FiBiNet) + cross (DCN) + deep (MLP).
✅ **Adaptabilité contextuelle** : SENet pour pondération dynamique.
✅ **Features interprétables** : CTR, popularité, affinités permettent l'analyse post-hoc.

✅ **Robustesse** : Régularisation forte (dropout, weight decay, gradient clipping, label smoothing).
✅ **Scalabilité** : Architecture modulaire, parallélisable, adaptable à plus de données.

8.3. Pipeline de bout en bout

```
Raw Data → Preprocessing → Feature Engineering → Séquences
    ↓
Embeddings (Wide & Deep) + Stats + Séquences
    ↓
Multi-Head PPM + SENet + FiBiNet + DCN + Deep
    ↓
Fusion → Logit → Calibration → Probabilité
    ↓
Soumission
```

Tout le pipeline est reproductible et documenté.

---

9. Reproductibilité

9.1. Environnement

- **Python** : 3.12.12
- **PyTorch** : 2.8.0+cu126
- **GPU** : T4, V100, ou équivalent (16+ GB VRAM recommandé).
- **Packages** : `pandas`, `numpy`, `torch`, `sklearn`, `pyarrow`, `tqdm`.

9.2. Instructions

1. Placer les fichiers de données dans la structure attendue par `CONFIG`.
2. Ouvrir le notebook `ultra_enhanced_ppm.ipynb`.
3. Vérifier la configuration GPU (`DEVICE = 'cuda'`).
4. Exécuter le notebook Run All:
    - Chargement des données.
    - Construction des séquences.
    - Feature engineering.
    - Définition du modèle.
    - Entraînement.
    - Génération de la soumission.
5. Inference sur le private dataset :
    - Il suffit simplement de remplacer le chemin du fichier test par le nouveau et de relancer l'execution du notebook.

Temps d'exécution : ~3 heures sur GPU T4 x2.

9.3. Seed fixe

```
SEED = 42
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
np.random.seed(SEED)
```

Garantit la reproductibilité.

---

10. Conclusion

Ce projet présente une architecture de recommandation vidéo **state-of-the-art**, combinant :

- Modélisation séquentielle avancée (Multi-Head PPM avec GRU + Attention).
- Interactions riches multi-niveaux (FiBiNet bilinéaires, DCN cross layers).
- Mécanismes d'attention adaptatifs (SENet pour pondération dynamique).
- Feature engineering statistique robuste (10 features normalisées).
- Pipeline de bout en bout reproductible et documenté.

Performance finale :

- **Modèle unique (soumis)** : AUC = **0.64007**
- **Ensemble (expérimental)** : AUC = **0.645**

Choix stratégique : Soumission du **modèle unique** pour un rapport optimal entre performance, reproductibilité et robustesse.

11. Références

1. **PPM: A Pre-trained Plug-in Model for Click-through Rate Prediction**
   Gao, Y., Lin, P. — *PPM: A Pre-trained Plug-in Model for Click-through Rate Prediction* (WWW).
   → Base conceptuelle de la brique PPM (modélisation séquentielle + attention).

2. **FiBiNET: Combining Feature Importance and Bilinear Feature Interaction**
   Huang, T., Zhang, Z., Zhang, J. (2019) — *FiBiNET* (RecSys).
   → Importance des champs (SENet) + interactions bilinéaires explicites.

3. **DCN V2: Improved Deep & Cross Network**
   Wang, R., et al. (2020) — *DCN V2* (WWW).
   → Cross network pour modéliser des interactions d’ordre élevé.

4. **Wide & Deep Learning for Recommender Systems**
   Cheng, H. T., et al. (2016).
   → Paradigme “mémorisation + généralisation” (wide + deep), largement repris en CTR.

Auteur : Abdourahamane Ide Salifou , Engineering student in Artificial Intelligence at Carnegie Mellon University Africa

Compétition : DataTour 2025 – Finale Internationale
Date : Décembre 2025

Contact : aidesali@andrew.cmu.edu