Miguel Breault

# DÉFI CAUSAL TRANSFORMER

pour la classification de qualité EEG (EEGMMIDB)

Présenté à

Usef Faghihi

UQTR

Département de mathématique et informatique

30 janvier 2026

# Préface

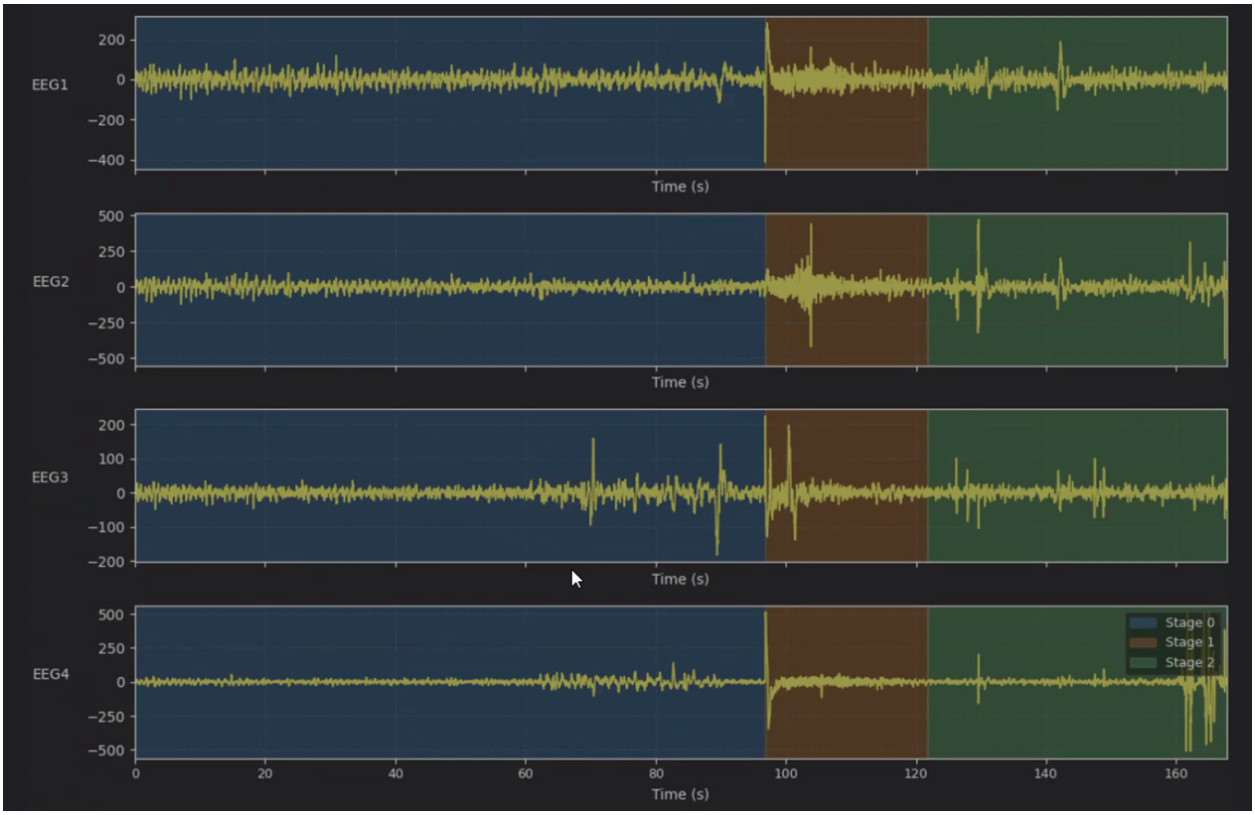



FIGURE 1 – EEG : signal brut et contrôle qualité (illustration).

Ce projet est né d'une volonté d'appliquer la rigueur du *CausalTransformer* [4] à un cas concret d'analyse EEG. Le dépôt initial étant incomplet pour cette tâche, j'ai développé le pipeline nécessaire pour passer de la donnée brute à la classification, en prenant soin de désactiver proprement la causalité «  médicale » au profit d'une approche temporelle.

Dans ce rapport, je présente l'implémentation sur EEGMMIDB [1] et insiste particulièrement sur les mécanismes anti-fuite (*anti-leakage*). J'y détaille comment j'ai sécurisé l'apprentissage (normalisation isolée, séparation par sujet) et validé les résultats par des contrôles stricts (permutations, tests inter-batch), le tout tracé via MLflow.

Je retiens surtout de cette expérience qu'un score élevé ne vaut rien s'il repose sur une fuite de données : la méthode doit être inattaquable. Face à l'envergure du projet, j'ai utilisé *OpenAI Codex CLI* [6] comme accélérateur global de développement et d'apprentissage, tout en exerçant une validation critique constante sur le code et les résultats pour en assurer la maîtrise.

# 1 Résumé exécutif

**Résumé de la section** — Cette section présente l'objectif du projet, l'approche retenue pour transformer *CausalTransformer* en pipeline de classification EEG à 3 classes, les résultats chiffrés issus des exports MLflow, ainsi que les principales limites (labels heuristiques, portée non clinique).

**Objectif.** Le dépôt *CausalTransformer* (initialement conçu pour l'inférence causale et l'estimation de résultats contrefactuels) a été adapté en un pipeline de **classification de qualité EEG** en **3 classes** {mauvais, moyen, excellent} sur EEGMMIDB/EEGBCI (PhysioNet), avec une contrainte forte de **zéro fuite de données** et de traçabilité via MLflow.

**Approche.** L'adaptation repose sur (i) un pipeline de données EEG (lecture EDF, fenêtrage, extraction de caractéristiques, normalisation) et (ii) l'utilisation du Transformer causal en mode *classification* (désactivation de la branche traitement : `treatment_mode=none`, `dim_treatments=0`) tout en conservant les mécanismes de masquage causal décrits dans l'architecture de référence [5].

**Résultats (exports MLflow).** Sur la configuration candidate (`raw8` + `manualcw`, seed=600, 5 folds), le modèle CT obtient en test une **Balanced Accuracy (sujet)** de **0.906 ± 0.056** et un **Macro-F1 (enregistrement)** de **0.840 ± 0.026** (voir Tables 1 et 4). Les baselines internes (EEGNet, ShallowConvNet, SimpleCNN, CSP+LDA) sont récapitulées dans la Table 4.

**Contrôles de validité.** Deux contrôles clés sont présentés : (i) *permutation des labels* (performance attendue proche du hasard) et (ii) *découplage entrées/labels* via permutation inter-batch (`CT_SHUFFLE_TRAIN_INPUTS_MODE=inter_batch_decouple`), attendu comme test « fort » anti-autopilote. Un résumé figure en Table 9.

**Limites.** Les labels {mauvais, moyen, excellent} sont **heuristiques** (non cliniques) et dérivés de métriques de qualité du signal (Section 3.3). Les informations de citation du dataset (DOI **10.13026/C28G6P**, licence **ODC-By 1.0**) sont incluses en bibliographie.

# Table des matières

# 2 Contexte et Objectifs

**Résumé de la section** — Cette section situe le problème (variabilité de qualité des signaux EEG), décrit l'objectif de classification en trois classes et explique le choix d'une adaptation du *Causal Transformer* (Transformer causal initialement destiné aux résultats contrefactuels) vers une tâche de classification temporelle.

## 2.1 La Problématique EEG

La qualité des signaux EEG varie fortement selon les sujets, les sessions et les conditions expérimentales. Les artefacts (bruit de ligne, saturation, segments plats, variations d'amplitude) peuvent dégrader l'analyse en aval (p. ex. classification de tâches, détection d'événements). Dans ce projet, la qualité est formulée comme une tâche de **classification 3 classes** {mauvais, moyen, excellent} afin de fournir un module de contrôle qualité (QC) reproductible.

Contrainte clé : EEGMMIDB ne fournit pas de labels cliniques de qualité. Les labels utilisés ici sont donc **heuristiques** et doivent être interprétés comme une approximation statistique, non comme une vérité terrain médicale (voir Section 3.3).

## 2.2 L'Approche Causal Transformer

Le *Causal Transformer* [5] est conçu pour modéliser des séries temporelles avec un masquage causal (pas d'accès au futur) et des entrées structurées en covariables et traitements, afin d'estimer des résultats (potentiellement contrefactuels). Le dépôt étudié implémente cette famille de modèles (notamment via les modules Transformer et leurs masques d'attention).

**Détournement vers la classification EEG.** L'adaptation consiste à utiliser le backbone Transformer pour prédire une classe de qualité à partir de séquences de fenêtres EEG, tout en désactivant la logique « traitement » lorsque la tâche n'en nécessite pas :

- **Désactivation des traitements** : configuration EEG avec `treatment_mode=none` et `dim_treatments=0` (confirmé par la configuration exportée dans les `params.json`).
- **Tête de classification** : sortie en 3 classes (`dim_outcomes=3`) et apprentissage via une loss de type cross-entropy (voir `config/dataset/eegmmidb.yaml` et `src/models/time_varying_model.py`).
- **Agrégations d'évaluation** : métriques calculées aux niveaux fenêtre (*window*), enregistrement (*record*) et sujet (*subject*), exportées sous forme de `metrics.json`, `predictions_*.csv`, matrices de confusion et rapports de classification.

**Contributions techniques :** D'après l'inspection du code, des configurations Hydra et des artefacts MLflow exportés (sans diff git) :

- ajout d'un dataset EEGMMIDB (lecture EDF, fenêtrage, extraction de features, labellisation) ;

— suppression/neutralisation de la logique « shock target » et focalisation sur la classification ;
— instrumentation de contrôles de validité (shuffle labels et shuffle inputs « fort » inter-batch) ;
— ajout de baselines EEG (EEGNet, ShallowConvNet, SimpleCNN, CSP+LDA) et d'exports MLflow structurés.

# 3 Ingénierie MLOps et Rigueur Scientifique (Pipeline)

**Résumé de la section** — Cette section décrit le pipeline de données EEGMMIDB (lecture EDF, fenêtrage, extraction de features), la stratégie de split sujet-disjoint et les mécanismes anti-fuite (normalisation train-only, labellisation train-only, contrôles de validité). Elle explicite également comment les classes {mauvais, moyen, excellent} sont construites.

## 3.1 Diagrammes du système

Cette sous-section fournit une vue d'ensemble compacte (fidèle à l'implémentation) de l'adaptation du Causal Transformer utilisée pour la classification de qualité EEG sur EEGMMIDB, ainsi que du pipeline de données à zéro fuite implémenté dans ce dépôt.

La Figure 2 résume comment le CT multi-entrées est utilisé ici avec la branche des traitements désactivée (pas de traitements, covariables EEG uniquement) pour produire une prédiction de qualité à 3 classes.

La Figure 3 met en évidence les garanties anti-fuite essentielles : split sujet-disjoint et ajustement sur le train uniquement, à la fois pour la normalisation et la labellisation par quantiles.

Enfin, la Figure 4 donne une vue simplifiée des classes principales impliquées (PyTorch Lightning, composants CT/EDCT, et dataset collection EEGMMIDB).

## 3.2 Pipeline de Données "Zero-Leakage"

### Dataset EEGMMIDB / EEGBCI (PhysioNet) : protocole et format

**Source et format.** Les données EEG proviennent de EEGMMIDB/EEGBCI (PhysioNet) [1] et sont stockées localement sous forme de fichiers `.edf` (EDF+). Dans ce dépôt, les chemins attendus/acceptés pour la découverte des fichiers EDF sont implémentés dans `src/data/physionet_eegmmidb/dataset.py` (détection de layout + scan récursif).

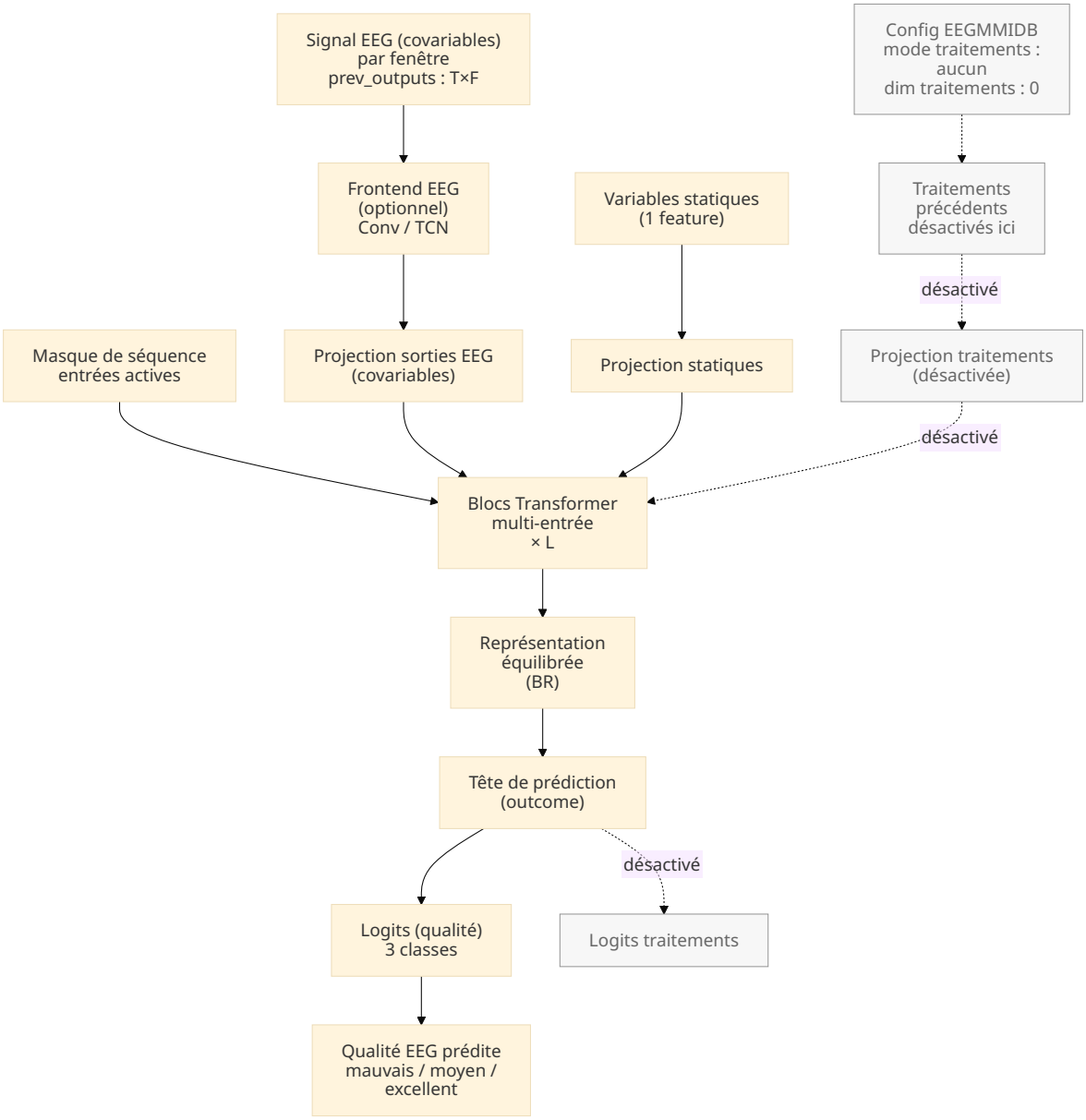



FIGURE 2 – Architecture du Causal Transformer adaptée à la classification de qualité EEG. La branche des traitements est désactivée (treatment_mode=none, dim_treatments=0) afin de n'utiliser que les covariables EEG.

**Version, DOI et licence.** EEGMMIDB/EEGBCI (PhysioNet) est référencé comme version **1.0.0** (9 sept. 2009), DOI **10.13026/C28G6P**, sous licence **Open Data Commons Attribution License v1.0 (ODC-By)** [1].

**Citations recommandées.** La même documentation demande de citer (i) BCI2000 [7] et (ii) la référence standard PhysioNet [2].

**Sujets et enregistrements.** La copie locale utilisée dans ce projet contient **109** dossiers sujets (`S001` à `S109`) dans `data/eegmmidb/MNE-eegbci-data/files/`. EEGMMIDB est décrit

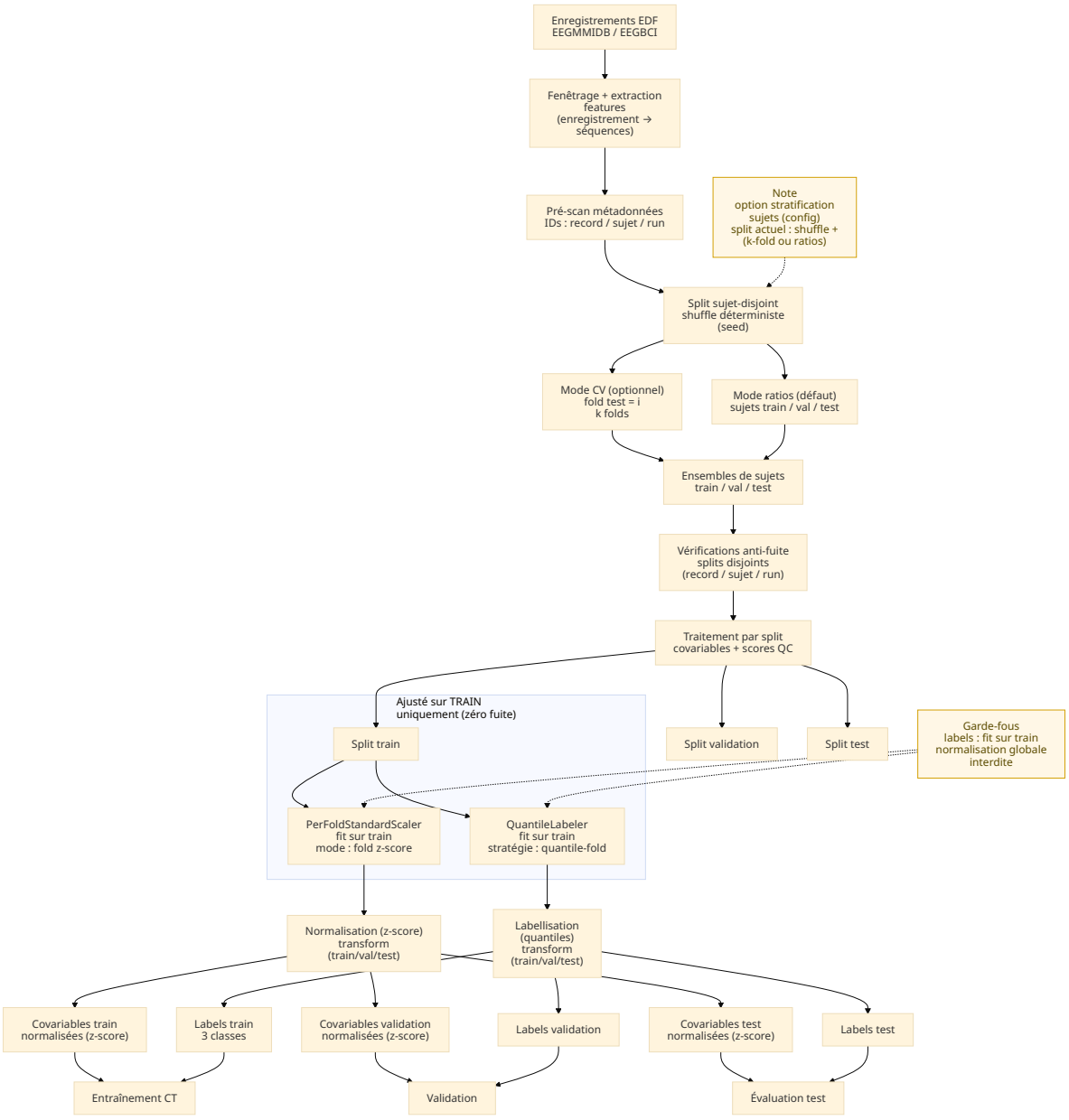



FIGURE 3 – Pipeline de cross-validation à zéro fuite. Les statistiques de normalisation et les seuils de labellisation par quantiles sont ajustés sur les sujets du train uniquement, puis appliqués aux sujets de validation/test.

comme contenant **plus de 1500** enregistrements EEG de **1 à 2 minutes**, acquis chez **109 volontaires** [1].

**Annotations.** Des fichiers `.edf.event` sont présents à côté des EDF (ex. : `data/eegmmidb/MNE-eegbci-data/files/S001/S001R04.edf.event`) et contiennent des marqueurs `T0/T1/T2`. EEGMMIDB précise que ces annotations sont identiques à celles présentes dans le canal d'annotation EDF+ et décrit les codes [1] :

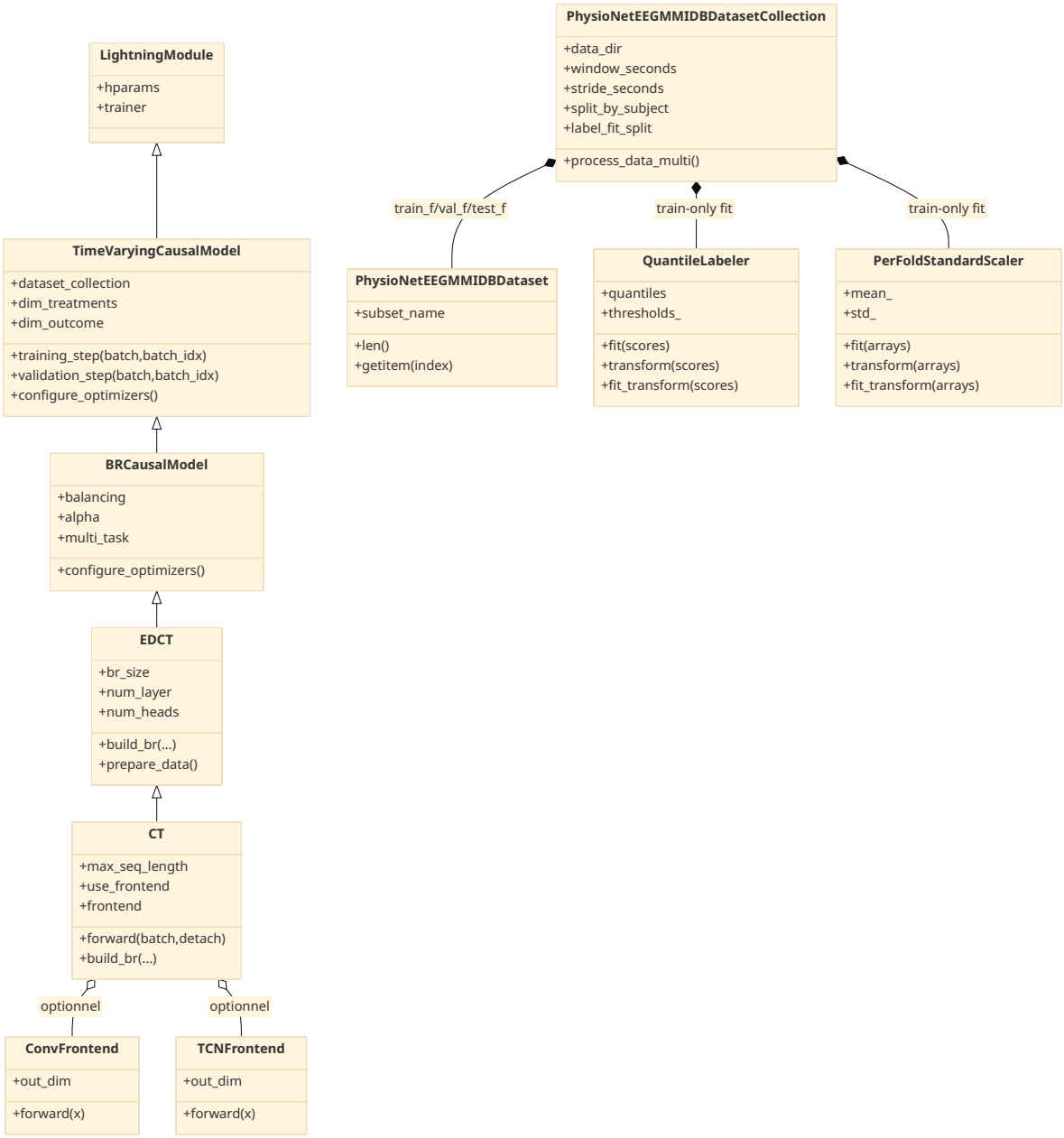



FIGURE 4 – Structure de classes simplifiée de l'implémentation (PyTorch Lightning + composants CausalTransformer + dataset collection EEGMMIDB).

— `T0` : repos ;
— `T1` : début du mouvement (réel ou imaginé) de la main **gauche** (runs 3, 4, 7, 8, 11, 12) ou des **deux poings** (runs 5, 6, 9, 10, 13, 14) ;
— `T2` : début du mouvement (réel ou imaginé) de la main **droite** (runs 3, 4, 7, 8, 11, 12) ou des **deux pieds** (runs 5, 6, 9, 10, 13, 14).

**Important :** ces marqueurs décrivent les *tâches motrices* du protocole EEGMMIDB et ne sont pas utilisés pour la définition des labels de *qualité* (Section 3.3).

**Protocole (runs).** Les enregistrements sont nommés `SXXXRYY.edf` (sujet `SXXX`, run `RYY`). La documentation EEGMMIDB indique que chaque sujet effectue **14 runs** : 2 baselines (1 minute) puis 4 tâches répétées 3 fois (2 minutes chacune). La séquence des runs est :

| Run | Description |
|---|---|
| 1 | Baseline, yeux ouverts |
| 2 | Baseline, yeux fermés |
| 3, 7, 11 | Tâche 1 : ouvrir/fermer le poing gauche ou droit (mouvement réel) |
| 4, 8, 12 | Tâche 2 : imaginer ouvrir/fermer le poing gauche ou droit |
| 5, 9, 13 | Tâche 3 : ouvrir/fermer les deux poings ou les deux pieds (mouvement réel) |
| 6, 10, 14 | Tâche 4 : imaginer ouvrir/fermer les deux poings ou les deux pieds |

**Fréquence d'échantillonnage et canaux.** Une lecture de l'en-tête EDF via MNE (`mne.io.read_raw_edf(...,preload=False)`) sur un enregistrement local (`S001R04.edf`) indique :

- fréquence d'échantillonnage : **160 Hz** ;
- nombre de canaux EEG : **64** ;
- exemples de noms de canaux : `Fc5.`, `Fc3.`, `Fc1.`, `Fcz.`, `Fc2.`, `Fc4.`, `Fc6.`, `C5..`, ....

**Note.** La documentation EEGMMIDB décrit des fichiers EDF+ contenant 64 signaux EEG (160 Hz) et un canal d'annotation ; les fichiers `.event` contiennent les mêmes annotations.

### Montage (système 10–10)

La documentation EEGMMIDB indique un montage 64 électrodes selon le système international 10–10, en excluant : `Nz`, `F9`, `F10`, `FT9`, `FT10`, `A1`, `A2`, `TP9`, `TP10`, `P9`, `P10`. Une figure de référence est fournie avec la copie locale du dataset.

### Fenêtrage (windowing) et séquences

**Fenêtrage.** Le dataset convertit chaque enregistrement EDF en une séquence de fenêtres temporelles via :

- `dataset.window_seconds = 2.0` ;
- `dataset.stride_seconds = 1.0` ;
- `dataset.max_seq_length = 60` (tronquage si nécessaire).

Ces valeurs sont visibles dans les `params.json` des runs finaux (p. ex. fold 0, seed=600).

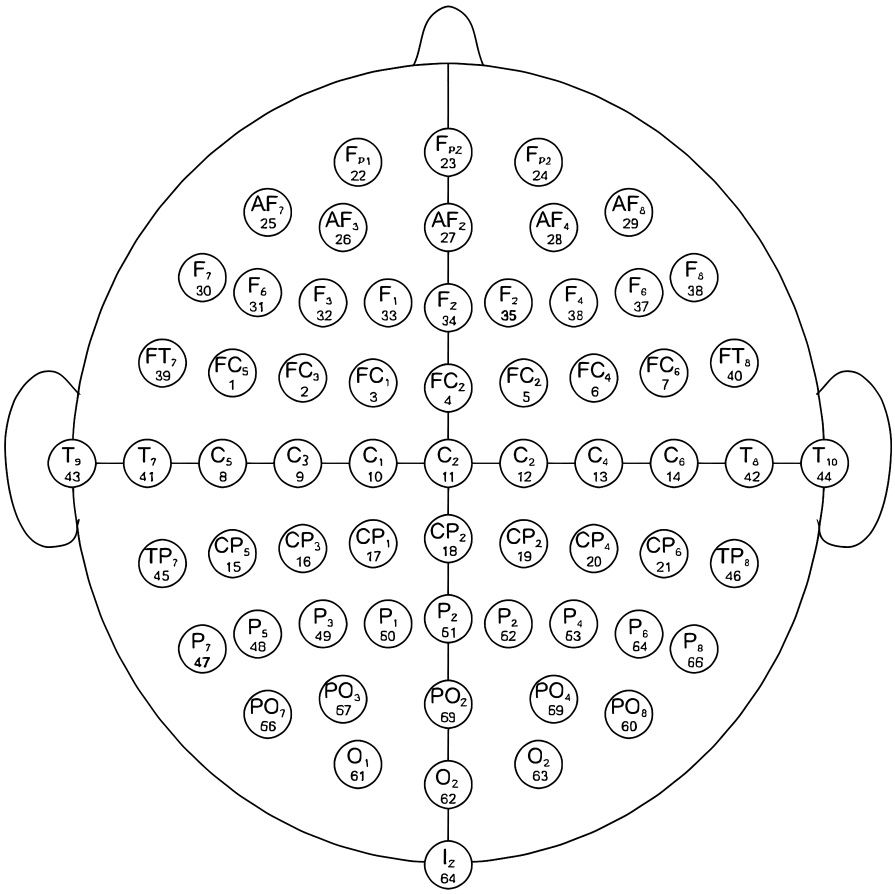



FIGURE 5 – Montage 64 canaux fourni dans la copie locale EEGMMIDB (`data/eegmmidb/MNE-eegbci-data/files/64_channel_sharbrough.png`).

### Features EEG (config “raw8”)

Les runs finaux présentés utilisent `dataset.feature_set=raw8`. Dans le code (`src/data/physionet_eegmmidb/dataset.py`), `raw8` correspond à 8 features calculées par fenêtre (agrégées sur les canaux) :

- `rms_mean`, `rms_std` ;
- `ptp_mean`, `ptp_std` ;
- `line_ratio_mean` (55–65 Hz, relatif) ;
- `alpha_ratio_mean`, `beta_ratio_mean`, `theta_ratio_mean`.

#### Splits sujet-disjoints et cross-validation

**Principe.** L'objectif est d'éviter la fuite "subject leakage" : aucune fenêtre d'un sujet ne doit être vue à la fois en train et en test. Le pipeline utilise `dataset.split_by_subject=True` et `dataset.fold_index` pour produire des splits **sujet-disjoints**.

**Mécanisme CV.** Dans l'implémentation actuelle (`src/data/physionet_eegmmidb/dataset.py::_split_subjects`), la CV :

- découpe les sujets en `k = dataset.n_folds` blocs (ici `k=5`) ;
- choisit le bloc `fold_index` comme **test** ;
- découpe les sujets restants en **train/val** selon `train_ratio` et `val_ratio`.

**Remarque importante.** Le paramètre `dataset.stratify_subject_split` est bien présent et exporté dans les `params.json`, mais il n'est pas utilisé explicitement par `_split_subjects` dans l'état actuel du code ; ceci est discuté comme limite et piste d'amélioration (Section 7).

**Garde-fous anti-fuite (assertions).** Après construction des splits, le code applique des vérifications de disjonction (`assert_disjoint_splits`) sur plusieurs identifiants (`record_id`, `subject_id`, `subject_run_id`) et lève une `RuntimeError` en cas de recouvrement (`src/data/physionet_eegmmidb/dataset.py`).

#### Normalisation "train-only" (anti-fuite)

La normalisation est configurée en `dataset.normalization_mode=fold_zscore` avec `dataset.normalization_scope=train`. Le code interdit explicitement `normalization_scope=global` (erreur fatale) car cela ferait fuiter des statistiques du val/test vers le train (`src/data/physionet_eegmmidb/dataset.py::_apply_normalization`).

### 3.3 Labellisation Heuristique (Ground Truth)

**Pourquoi une heuristique ?** EEGMMIDB est un dataset de tâches motrices (exécution / imagerie) et ne fournit pas de labels « qualité clinique » directement exploitables pour une tâche QC. Les classes {mauvais, moyen, excellent} sont donc construites automatiquement à partir de métriques objectives du signal.

**Score de qualité par fenêtre.** Un score scalaire est calculé par fenêtre à partir de métriques (amplitude RMS/PTP, composantes fréquentielles, bruit de ligne, etc.). La version "composite" (dans `src/data/utils/quality.py`) combine notamment un terme SNR et des pénalités de bruit de ligne/saturation/flatline.

**QuantileLabeler (3 classes).** Les seuils de classes proviennent d'un labeler par quantiles (`src/data/utils/labeling.py::QuantileLabeler`) :

- seuil bas = quantile `q_low` (par défaut 0.33) ;
- seuil haut = quantile `q_high` (par défaut 0.66) ;

— mapping : `score` $\leq$ `t_low` $\rightarrow$ **mauvais** (0), `t_low` < `score` < `t_high` $\rightarrow$ **moyen** (1), `score` $\geq$ `t_high` $\rightarrow$ **excellent** (2).

**Train-only (anti-fuite).** Dans la stratégie `label_strategy=quantile_fold`, les quantiles sont ajustés (`fit`) **uniquement sur les scores du train** du fold courant, puis appliqués (`transform`) à val/test. Les exports MLflow incluent des artefacts de labellisation (p. ex. `artifacts/labelers/quantile_labeler.json`).

# 4 Résultats de Performance (Benchmark)

**Résumé de la section** — Cette section présente les métriques test issues des exports MLflow : (i) la validation croisée 5-fold du modèle CT sur la configuration candidate, (ii) une comparaison quantitative avec plusieurs baselines EEG, et (iii) une analyse par classe aux niveaux window/record/subject.

## 4.1 Performance Comparée (SOTA Interne)

**Validation croisée (CT, seed=600).** La Table 1 récapitule les scores **test** par fold pour la configuration candidate (CT, `raw8` + `manualcw`). Les métriques sont calculées aux niveaux *window*, *record* et *subject*. Pour situer la sélection d'epoch et la stabilité sur validation, la Table 2 rapporte les mêmes métriques sur **val**.

TABLE 1 – Validation croisée (test) — métriques par fold (CT, raw8, manualcw, seed=600).

| Fold | Run ID | Run name | epoch | BalAcc (window) | Macro-F1 (window) | Acc (window) | BalAcc (record) | Macro-F1 (record) | Acc (record) | BalAcc (subject) | Macro-F1 (subject) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1ebc9c06 | dazzling-pig-862 | 55 | 0.823 | 0.825 | 0.825 | 0.870 | 0.867 | 0.867 | 0.900 | 0.900 |
| 1 | 16521b4c | bold-fawn-723 | 67 | 0.793 | 0.777 | 0.763 | 0.807 | 0.801 | 0.789 | 0.833 | 0.822 |
| 2 | a8f5bd3a | masked-foal-764 | 60 | 0.802 | 0.806 | 0.798 | 0.823 | 0.830 | 0.818 | 0.944 | 0.950 |
| 3 | b7f7e83a | fearless-shad-563 | 38 | 0.822 | 0.800 | 0.835 | 0.893 | 0.853 | 0.890 | 0.976 | 0.921 |
| 4 | d5599a49 | dapper-hound-665 | 45 | 0.784 | 0.790 | 0.798 | 0.842 | 0.851 | 0.854 | 0.878 | 0.875 |
| **Moy ± Std** | | | | **0.805 ± 0.017** | **0.799 ± 0.018** | **0.804 ± 0.028** | **0.847 ± 0.035** | **0.840 ± 0.026** | **0.843 ± 0.040** | **0.906 ± 0.056** | **0.894 ± 0.049** |

TABLE 2 – Validation croisée (val) — métriques par fold (CT, raw8, manualcw, seed=600).

| Fold | Run ID | Run name | epoch | BalAcc (window) | Macro-F1 (window) | Acc (window) | BalAcc (record) | Macro-F1 (record) | Acc (record) | BalAcc (subject) | Macro-F1 (subject) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1ebc9c06 | dazzling-pig-862 | 55 | 0.792 | 0.790 | 0.805 | 0.871 | 0.848 | 0.871 | 0.847 | 0.763 |
| 1 | 16521b4c | bold-fawn-723 | 67 | 0.784 | 0.791 | 0.804 | 0.769 | 0.785 | 0.833 | 0.833 | 0.871 |
| 2 | a8f5bd3a | masked-foal-764 | 60 | 0.784 | 0.786 | 0.795 | 0.818 | 0.826 | 0.843 | 1.000 | 1.000 |
| 3 | b7f7e83a | fearless-shad-563 | 38 | 0.755 | 0.764 | 0.789 | 0.800 | 0.806 | 0.829 | 0.822 | 0.806 |
| 4 | d5599a49 | dapper-hound-665 | 45 | 0.842 | 0.830 | 0.870 | 0.885 | 0.858 | 0.902 | 0.970 | 0.873 |
| **Moy ± Std** | | | | **0.791 ± 0.032** | **0.792 ± 0.024** | **0.813 ± 0.033** | **0.829 ± 0.049** | **0.824 ± 0.030** | **0.856 ± 0.031** | **0.894 ± 0.084** | **0.863 ± 0.090** |

**Baselines (détail).** Pour isoler les modèles de référence, la Table 3 présente uniquement les baselines EEG (moyenne ± écart-type sur 5 folds).

**Baselines.** La comparaison interne CT vs baselines (EEGNet, ShallowConvNet, SimpleCNN, CSP+LDA) est fournie en Table 4. Les scores baselines proviennent de l'export MLflow des baselines (fichier `runs_summary.csv` ; sélection déterministe du run le plus récent par {modèle, fold}).

TABLE 3 – Baselines EEG (test) — moyenne ± std sur 5 folds.

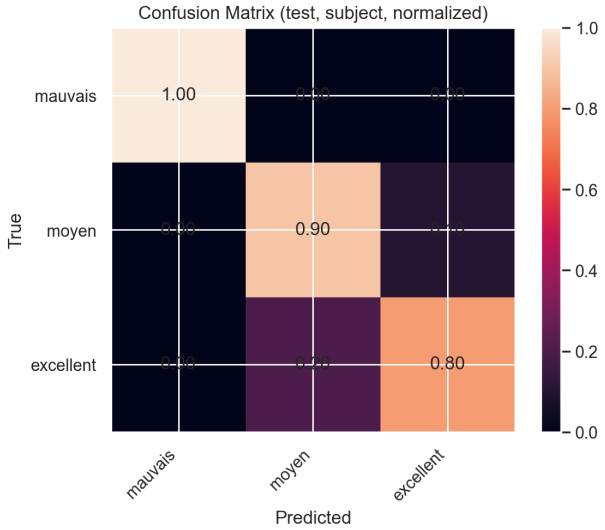
| Modèle | Balanced Acc (subject) | Macro-F1 (record) |
|---|---|---|
| EEGNet | 0.706 ± 0.070 | 0.697 ± 0.043 |
| ShallowConvNet | 0.557 ± 0.098 | 0.542 ± 0.068 |
| SimpleCNN | 0.757 ± 0.089 | 0.735 ± 0.054 |
| CSP+LDA | 0.503 ± 0.072 | 0.370 ± 0.070 |

TABLE 4 – Benchmark interne — CT vs baselines (test).

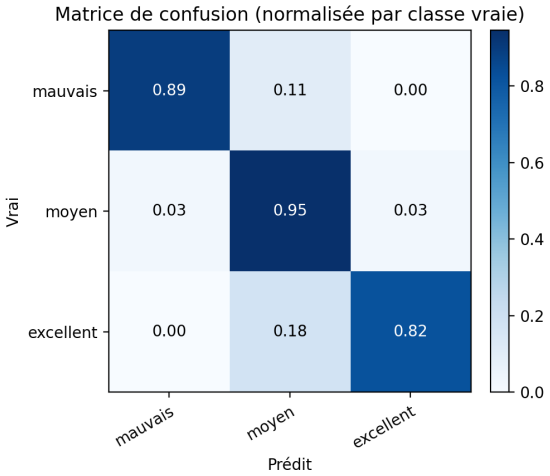
| Modèle | BalAcc (subject) | Macro-F1 (record) | Observations |
|---|---|---|---|
| **CT (Raw8, manualcw)** | **0.906 ± 0.056** | **0.840 ± 0.026** | Config candidate (seed=600, 5 folds, train-only). |
| EEGNet | 0.706 ± 0.070 | 0.697 ± 0.043 | |
| ShallowConvNet | 0.557 ± 0.098 | 0.542 ± 0.068 | |
| SimpleCNN | 0.757 ± 0.089 | 0.735 ± 0.054 | |
| CSP+LDA | 0.503 ± 0.072 | 0.370 ± 0.070 | |

## 4.2 Matrices de confusion (exemple vs agrégé)

**Niveau *subject*.** La Figure 6 compare la matrice de confusion normalisée du fold 0 (seed=600) à la matrice agrégée sur 5 folds (seed=600). Des figures supplémentaires (niveaux record/window et courbes d'entraînement) sont fournies en annexe (Section 9.4).




(a) Fold 0, seed=600




(b) Agrégé 5 folds, seed=600

FIGURE 6 – Matrices de confusion normalisées au niveau *subject* (test).

## 4.3 Analyse de Stabilité (Robustesse Multi-Seed)

La robustesse multi-seed (pipeline **train-only**) est résumée en Table 5 pour deux campagnes complètes (seeds 600 et 700, 5 folds chacune).

TABLE 5 – Robustesse multi-seed (test) — pipeline train-only (CT, raw8, manualcw, 5 folds).

| Seed | Balanced Acc (subject) | Macro-F1 (record) |
|---|---|---|
| 600 | 0.906 ± 0.056 | 0.840 ± 0.026 |
| 700 | 0.884 ± 0.066 | 0.830 ± 0.046 |

### Analyse par classe (agrégé sur 5 folds, seed=600)

Pour comprendre les classes difficiles, les prédictions test ont été concaténées sur les 5 folds (niveau fenêtre, record, sujet) afin de produire des rapports par classe.

TABLE 6 – Rapport par classe (test, agrégé sur 5 folds) — niveau window (seed=600).

| Classe | Support | Precision | Recall | F1 |
|---|---|---|---|---|
| mauvais | 32425 | 0.910 | 0.865 | 0.887 |
| moyen | 29851 | 0.673 | 0.804 | 0.733 |
| excellent | 29255 | 0.860 | 0.736 | 0.793 |

TABLE 7 – Rapport par classe (test, agrégé sur 5 folds) — niveau record (seed=600).

| Classe | Support | Precision | Recall | F1 |
|---|---|---|---|---|
| mauvais | 532 | 0.944 | 0.891 | 0.917 |
| moyen | 534 | 0.740 | 0.865 | 0.798 |
| excellent | 460 | 0.877 | 0.763 | 0.816 |

# 5 Validation de la Causalité et du Modèle

**Résumé de la section** — Cette section regroupe les tests visant à vérifier que le pipeline apprend une relation entrée→label non triviale et qu'il n'exploite pas de fuite évidente. Elle discute (i) les shuffles d'entrée, (ii) les contrôles de validité (permutation des labels, découplage inter-batch), et (iii) l'analyse d'erreurs via matrices de confusion.

## 5.1 Preuve de Dépendance Temporelle (Input Shuffle)

Le dépôt implémente deux familles de shuffles d'entrées :

TABLE 8 – Rapport par classe (test, agrégé sur 5 folds) — niveau subject (seed=600).

| Classe | Support | Precision | Recall | F1 |
|---|---|---|---|---|
| mauvais | 38 | 0.971 | 0.895 | 0.932 |
| moyen | 37 | 0.778 | 0.946 | 0.854 |
| excellent | 34 | 0.966 | 0.824 | 0.889 |

— **Shuffle intra-séquence** : permutation de l'ordre des fenêtres **au sein** d'une séquence (train-only), en conservant l'alignement entre covariables et labels. Ce test évalue surtout la sensibilité du modèle à l'*ordre* des fenêtres.

— **Shuffle inter-batch (découplage)** : permutation des entrées **entre** éléments du batch sans permuter les labels. Ce test casse explicitement la correspondance entrée↔label et doit faire chuter les métriques vers le hasard si le pipeline est sain.

**Pourquoi le shuffle intra-séquence peut peu dégrader au niveau *subject*.** Dans le code EEGMMIDB (`src/data/physionet_eegmmidb/dataset.py`), le flag `CT_SHUFFLE_TRAIN_INPUTS=1` permute l'axe temporel **dans chaque séquence du train** en *lock-step* sur covariables/traits/qualité : l'alignement entrée↔label n'est donc pas cassé. Sur cette tâche, (i) la supervision est largement *fenêtre-dépendante* (qualité calculée fenêtre par fenêtre) et (ii) les agrégations record/subject (moyennes/votes) sont **invariantes** à la permutation de l'ordre des fenêtres. Conséquence : ce test est informatif sur la sensibilité à l'ordre, mais ne constitue pas un contrôle « fort » anti-autopilote.

## 5.2 Contrôles de validité (contrôle qualité)

Les contrôles de validité suivants sont considérés comme les plus informatifs avec la contrainte « max safe » :

— **Permutation de labels** : les performances doivent chuter vers le hasard (3 classes).

— **Découplage inter-batch** : permutation des entrées uniquement, labels fixes, ce qui doit également rapprocher les scores du hasard.

**Lecture.** La Table 9 synthétise les runs de contrôle de validité exportés. Pour une tâche à 3 classes, une Balanced Accuracy proche de 1/3 est cohérente avec un comportement "au hasard".

**Tests additionnels.** Le dépôt contient aussi un script de tests (`runnables/quality_tests.py`) incluant une prédiction d'ID sujet et des ablations de canaux. Aucun run exporté correspondant n'est inclus dans l'export MLflow utilisé pour ce rapport ; ces tests sont donc listés comme actions futures minimales (Section 7).

TABLE 9 – Contrôles de validité — résultats test issus des exports MLflow.

| Test | Statut | BalAcc (test) | Macro-F1 (test) | Run ID(s) |
|---|---|---|---|---|
| Label permutation (folds 0–4) | PASS | BalAcc subject (moyenne ± std) = 0.3333 ± 0 (identique sur 5 folds) | Macro-F1 subject (moyenne ± std) = 0.1633 ± 0.0564 | `691ce7bf`, `ae0afe45`, `2bd040d2`, `c6c94f0d`, `9cea104f` |
| Input-shuffle inter-batch-decouple (fold 0) | PASS | Δ BalAcc (window/record/subject) = -0.3778 / -0.4527 / -0.5524 | Δ Macro-F1 (window/record/subject) = -0.5386 / -0.5885 / -0.6611 | baseline : `d45bb37f` shuffle : `47eda505` |

## 5.3 Analyse des Erreurs (Matrice de Confusion)

**Matrice de confusion.** Une comparaison *fold* vs agrégé au niveau *subject* est présentée en Section 4.2 (Figure 6). Des matrices complémentaires aux niveaux *record* et *window* sont fournies en annexe (Section 9.4).

**Confiance des prédictions.** À titre illustratif, la Figure 7 montre l'histogramme de confiance (max des probabilités) au niveau fenêtre pour fold 0 (seed=600), tel qu'exporté par le post-traitement.

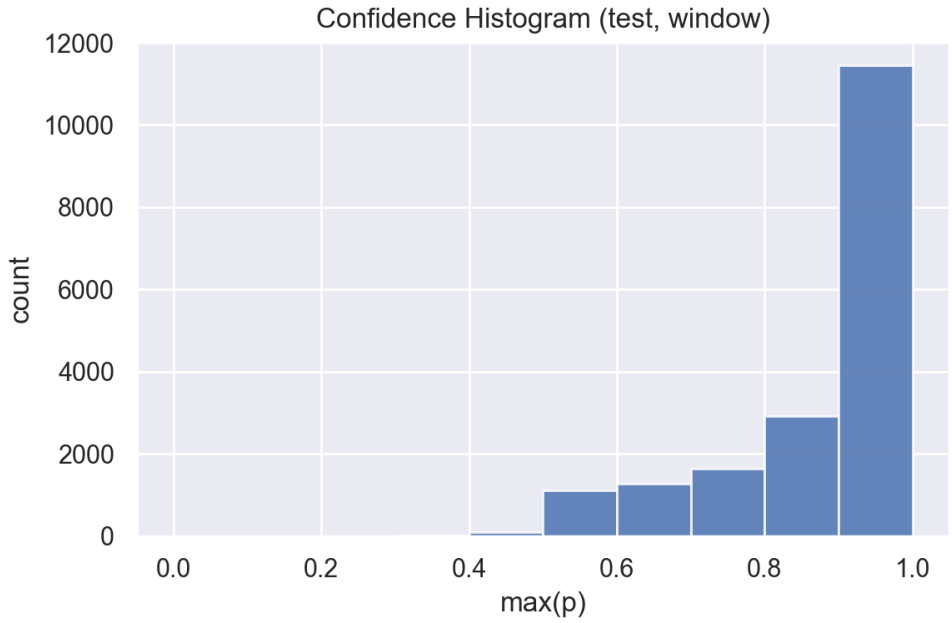



FIGURE 7 – Histogramme de confiance (niveau *window*, test, fold 0, seed=600) issu des artefacts exportés.

# 6 Défis rencontrés et résolutions

**Résumé de la section** — Cette section synthétise les difficultés techniques rencontrées et les correctifs appliqués pendant le projet. Chaque point est relié à une preuve exportée (configuration `params.json`, métriques/artefacts MLflow, et/ou inspection du code). Si un élément est *rapporté* mais non directement re-vérifiable depuis les exports, il est explicitement indiqué.

## 6.1 Alignement labels ↔ features

- **Problème (rapporté).** Les labels QC reposent sur des statistiques d'amplitude (RMS/PTP/*line ratio*). Des premières itérations basées uniquement sur des features de type bandpower normalisé pouvaient supprimer une partie de l'information d'amplitude, créant un décalage supervision→entrées.
- **Résolution.** Deux voies complémentaires : (i) utiliser un `feature_set` incluant des informations d'amplitude (`raw8`, utilisé dans les runs finaux ; Section 3.3) ; (ii) pour les presets bandpower, concaténer explicitement les statistiques d'amplitude via `dataset.bandpower_include_quality=True`. Toute modification de dimension nécessite de régénérer le cache (`.npz`).
- **Preuves (exports).** La configuration finale utilise `dataset.feature_set=raw8` et `dataset.bandpower_include_quality=False` (Annexe : Listing 1). Les performances obtenues sur la campagne seed=600sont reportées en Table 1.

## 6.2 Agrégation record/subject plus robuste

- **Problème (rapporté).** L'agrégation au niveau record par moyenne de probabilités (`mean_prob`) peut être instable face à des fenêtres aberrantes.
- **Résolution.** Passage à `record_level_agg=mean_logit` (moyenne des logits) et ajout d'options d'agrégation robustes (p. ex. `trimmed_mean_prob` avec `record_level_trim`).
- **Preuves (exports + code).** Les runs finaux sont exportés avec `exp.record_level_agg=mean_logit` (Listing 1) et les fichiers `predictions_test_record.csv` contiennent la colonne `agg_used_record` (artefacts exportés par `src/models/time_varying_model.py`). Les scores record-level associés sont reportés en Table 1 (Macro-F1 record = $0.840 \pm 0.026$).

## 6.3 Masquage d'attention (padding, causalité) et stabilité fp16

- **Problème (rapporté).** Risque de fuite via les tokens de padding si le masque d'attention n'est pas appliqué symétriquement (requêtes et clés). En fp16, l'utilisation de grandes constantes négatives (p. ex. $-10^9$) peut provoquer des overflow.
- **Résolution.** Masquage cohérent des requêtes *et* des clés (padding) et valeur de remplissage *dtype-safe*. En pratique, les runs EEG sont exécutés en `precision=32` sur Windows.

— **Preuves (exports + code).** Implémentation dans `src/models/utils_transformer.py` (fonctions `_build_attn_mask` et `_get_mask_fill_value`). La configuration exportée utilise `exp.precision=32` (Listing 1).

## 6.4 Optimisation / régularisation (stabilité et variance inter-fold)

— **Problème (rapporté).** Premières exécutions sensibles au sur-apprentissage (LR fixe, peu de régularisation, et instabilités en fp16).

— **Résolution.** Régularisation via `weight_decay=1e-4`, scheduler `cosine_warmup` (warmup `warmup_pct=0.05`) et `label_smoothing=0.05`. Ces choix sont intégrés dans les presets CT EEG.

— **Preuves (exports).** Ces hyperparamètres apparaissent dans la configuration condensée (Listing 1). Les métriques CV associées sont reportées en Tables 2 et 1.

## 6.5 Distribution des splits sujets

— **Problème (rapporté).** Variations de distribution des classes entre train/val/test au niveau sujet (shift inter-split) susceptibles d'augmenter la variance inter-fold.

— **Résolution (rapporté).** Introduction/activation de `dataset.stratify_subject_split=True` pour stabiliser les proportions de classes lors de la construction des folds sujets.

— **État actuel (vérifié dans le code).** Le paramètre est bien présent et exporté (`params.json`), mais l'implémentation CV actuellement utilisée (`src/data/physionet_eegmmidb/dataset.py::_split_subjects`) ne met pas en œuvre de stratification explicite. Une base technique existe néanmoins (`_make_subject_folds(...,strata=...)`, `_compute_subject_strata`) et constitue une amélioration minimale (voir Section 7).

## 6.6 Contrôles de validité et anti-"autopilote"

— **Problème.** Risque de modèles "autopilotes" (apprentissage d'artefacts, fuites implicites) donnant des scores artificiellement élevés.

— **Résolution.** Deux contrôles minimaux archivés dans les exports : (i) permutation de labels, (ii) découplage entrées/labels via permutation inter-batch (`CT_SHUFFLE_TRAIN_INPUTS_MODE=inter_batch_decouple`). Un point clé est que le shuffle intra-séquence ne casse pas l'alignement entrée↔label et peut donc peu impacter le niveau *subject* (discussion en Section 7 et Section 4.2).

— **Preuves (exports).** La Table 9 liste les runs utilisés (IDs + résumé). Pour une tâche à 3 classes, une Balanced Accuracy proche de 1/3 est cohérente avec un comportement "au hasard".

— **Vérification des splits seule (rapporté).** Un mode `CT_ONLY_SPLIT_CHECK=1` est utilisé pour vérifier la disjonction des splits sans entraînement. Les exports disponibles contiennent le flag, mais pas l'intégralité des logs texte de disjonction ; la vérification textuelle détaillée est donc *partielle* depuis les exports seuls.

## 6.7 Obstacles pratiques (Windows, Hydra, cache)

— **Windows / DataLoader (rapporté).** L'utilisation de workers multiples dans les DataLoader PyTorch peut provoquer des conflits OpenMP (`libiomp5md.dll` already initialized). Pour stabilité, les presets EEG fixent `dataset.num_workers=0`, `exp.num_workers=0`, `exp.persistent_workers=False` et `exp.pin_memory=False` (valeurs visibles dans les `params.json` exportés).

— **Inspection Hydra (rapporté).** Pour afficher la configuration résolue sans exécuter : `--cfgjob`. Cette option évite des erreurs d'interpolation (p. ex. `UnsupportedInterpolationType`) lors d'une inspection rapide.

— **Cache features.** Les changements de dimension de features nécessitent de reconstruire le cache (`.npz`) pour éviter des collisions de dimensions.

# 7 Discussion et Limitations

**Résumé de la section** — Cette section interprète les résultats en tenant compte des choix de labels et du protocole d'évaluation, identifie les limites vérifiables du pipeline (paramètres non utilisés, informations manquantes, risques méthodologiques) et propose des améliorations minimales et actionnables.

## 7.1 Forces du Modèle

— **Traçabilité.** Les résultats présentés proviennent d'exports MLflow structurés (`mlflow_exports`) incluant `metrics.json`, `params.json`, prédictions et matrices de confusion.

— **Évaluation réaliste par sujet.** L'évaluation est menée en mode sujet-disjoint (objectif : éviter la mémorisation des signatures individuelles), avec métriques reportées au niveau *subject*.

— **Rigueur "train-only".** Le code applique une normalisation fit sur le train et interdit `normalization_scope=global`. La labellisation par quantiles est également fit sur les scores du train du fold (`label_strategy=quantile_fold`).

— **Contrôles de validité.** Les tests *permutation des labels* et *découplage inter-batch* fournissent un garde-fou contre un apprentissage "autopilote" (Table 9).

## 7.2 Limitations Connues

— **Labels non cliniques.** Les classes {mauvais, moyen, excellent} sont dérivées d'un score de qualité calculé sur le signal (Section 3.3). Elles ne constituent pas une annotation médicale ; les résultats ne doivent donc pas être interprétés comme une performance clinique.

— **Objectif QC vs dataset de tâches.** EEGMMIDB est un dataset de tâches motrices (exécution/imagerie). Les marqueurs `T0/T1/T2` décrivent des événements de *repos* et de

*début de mouvement* (mains/pieds) selon le protocole ; toutefois, ils ne sont pas utilisés pour la définition des labels QC, qui restent dérivés d'un score de qualité du signal.

— **Paramètre "stratify_subject_split".** Le paramètre `dataset.stratify_subject_split` est présent et exporté, mais l'implémentation actuelle du split CV (`_split_subjects`) ne met pas en œuvre de stratification explicite. Une amélioration minimale consisterait à exploiter la fonction `_make_subject_folds(..., strata=...)` déjà présente dans le dataset.

— **Risque de « tautologie » des labels.** Dans la configuration `raw8`, certaines features (RMS/PTP/line ratio) sont proches des métriques utilisées pour construire le score de qualité. Ce n'est pas une fuite train→test en soi, mais cela peut rendre la tâche proche d'un classement direct de la même famille de statistiques, limitant la portée « sémantique » du label.

— **Coût et déploiement.** Aucun benchmark de latence/mémoire n'a été exporté dans `mlflow_exports`. Toute comparaison de coût d'inférence nécessiterait un micro-benchmark dédié (mesure directe sur la machine cible).

# 8 Conclusion et Recommandation de Déploiement

**Résumé de la section** — Cette section conclut sur la faisabilité de la transformation du dépôt *CausalTransformer* en pipeline de classification EEG de qualité, récapitule les résultats obtenus sur EEGMMIDB et propose des étapes futures minimales pour renforcer la validité et la reproductibilité.

**Conclusion.** Le dépôt *CausalTransformer* a été converti en un pipeline complet de classification EEG à 3 classes, incluant un dataset dédié (EEGMMIDB), une labellisation heuristique train-only, une évaluation sujet-disjoint et des exports MLflow auditables. Sur la campagne finale **seed=600** (Table 1), le modèle CT (`raw8` + `manualcw`) surpasse les baselines internes sur la métrique critique *niveau sujet* (Table 4) et montre une stabilité multi-seed cohérente (Table 5).

**Recommandation (au stade preuve de concept).**

— Pour une utilisation académique/démonstrateur : conserver CT comme modèle principal afin d'exploiter l'infrastructure d'évaluation multi-niveau et de contrôles de validité déjà en place.

— Pour un scénario de déploiement réel : comparer explicitement CT à une baseline plus légère (p. ex. EEGNet) sur la machine cible, avec des mesures de latence/mémoire (non disponibles dans les exports actuels).

**Pistes futures (minimales et vérifiables).**

— **Stratification CV par sujet (implémentation).** Exploiter `_make_subject_folds(...,strata=...)` et `_compute_subject_strata` afin que `dataset.stratify_subject_split=True` ait un effet mesurable (puis exporter les distributions par split via MLflow).

— **Calibration.** Ajouter des métriques et figures de calibration (p. ex. ECE/Brier + reliability diagram) aux niveaux window/record/subject dans le post-analyze, et les archiver dans `mlflow_exports`.
— **Ablations features/labels.** Quantifier l'impact de l'alignement label↔features via des runs contrôlés (`raw8` vs `quality3` vs bandpower, avec/ sans `bandpower_include_quality`) exportés et comparés au *niveau sujet*.
— **Validations QC additionnelles.** Ajouter/archiver des runs MLflow pour *prédiction d'ID sujet* et *ablation de canaux* (`runnables/quality_tests.py`) afin de rendre ces tests auditables (artefacts non inclus dans l'export utilisé ici).

# 9 Annexes

## 9.1 Configuration Finale (`hydra`)

**Objectif.** Fournir un extrait *minimal* des hyperparamètres clés du run de référence (fold 0, seed=600), tel qu'exporté dans `params.json`, afin de pouvoir reproduire une exécution sans recopie non traçable. **Source.** Export MLflow (run `dazzling-pig-862`, fichier `params.json`).

Listing 1 – Configuration condensée (extrait).

```yaml
dataset:
  name: eegmmidb
  feature_set: raw8
  window_seconds: 2.0
  stride_seconds: 1.0
  max_seq_length: 60
  split_by_subject: True
  n_folds: 5
  fold_index: 0
  normalization_mode: fold_zscore
  normalization_scope: train
  label_strategy: quantile_fold
  quantile_range: [0.33, 0.66]
  label_feature: composite

model:
  name: CT
  multi:
    max_seq_length: 65
    optimizer:
      optimizer_cls: adam
      learning_rate: 0.0003
      weight_decay: 0.0001
      lr_scheduler:
        type: cosine_warmup
        warmup_pct: 0.05
        t_max: 100

exp:
  seed: 600
  max_epochs: 100
  precision: 32
  class_weights_mode: manual
  class_weights: [1.0, 1.5, 1.0]
  label_smoothing: 0.05
  record_level_agg: mean_logit
  subject_level_agg: mean_prob
```

Le dataset tronque/packe les séquences à 60 pas de temps ; le modèle est paramétré avec une capacité maximale de 65 (borne supérieure), ce qui laisse une marge sans impacter les séquences observées.

## 9.2 Index des Expériences MLflow

**Résumé de la section** — Cette annexe fournit un index des expériences et runs utilisés dans le rapport (IDs stables) ainsi que les emplacements locaux des exports (réutilisables sans MLflow UI).

— **CT EEGMMIDB (export)** : `665075068361836799`
Dossier : `mlflow_exports/experiment_665075068361836799_CT_eegmmidb/`

— **Baselines EEGMMIDB (export)** : `886282226387861497`
Dossier : `mlflow_exports/experiment_886282226387861497_baselines_eegmmidb/`

**Runs CT présentés (seed=600, 5 folds).**

— fold 0 : `1ebc9c06368b41c9831524d285701843` (`dazzling-pig-862`)

— fold 1 : `16521b4c34b140e9aade8717f099261d` (`bold-fawn-723`)
— fold 2 : `a8f5bd3a58a94786bf71d821026eacf4` (`masked-foal-764`)
— fold 3 : `b7f7e83ab61940f1b99e9213d6c3dc6e` (`fearless-shad-563`)
— fold 4 : `d5599a491aed4c30beb8dd6981b410d9` (`dapper-hound-665`)

**Option B (seed=700, 5 folds).**

— fold 0 : `54c985de34014f7daff3ab6378c61cb4` (`likeable-owl-318`)
— fold 1 : `b6694b46fa7345aa8157f6be47ae14aa` (`unleashed-bee-731`)
— fold 2 : `7da77fac36064dc0bbba4bd40640d70f` (`exultant-bird-298`)
— fold 3 : `a974048a273242279bee17753f4de6e4` (`fun-roo-177`)
— fold 4 : `07df9328143b46e8b00e8e3357fbc216` (`nebulous-wren-944`)

## 9.3 Reproductibilité (Windows, commandes, exports)

**Résumé de la section** — Cette annexe consigne les éléments minimaux de reproductibilité sur Windows : OS/Conda, variables d'environnement, commandes de cross-validation, et export des runs MLflow.

**OS / environnement.** Les exécutions référencées proviennent d'un environnement Conda Windows (`(causaltransformer)`) et de commandes CMD depuis `C:\Projects\CausalTransformer`.

**Variables d'environnement (état "run normal").**

Listing 2 – Variables (extrait).

```
set "PYTHONPATH=."
set HYDRA_FULL_ERROR=1
set CT_ONLY_SPLIT_CHECK=0
set CT_SHUFFLE_TRAIN_LABELS=0
set CT_SHUFFLE_TRAIN_INPUTS=0
set CT_SHUFFLE_TRAIN_INPUTS_MODE=
```

**Campagne CV (seed=600, folds 0–4) — commandes Windows CMD (extrait fidèle).**

Listing 3 – Commande CV (seed=600).

```
set "PYTHONPATH=."
set HYDRA_FULL_ERROR=1

set CT_ONLY_SPLIT_CHECK=0
set CT_SHUFFLE_TRAIN_LABELS=0
set CT_SHUFFLE_TRAIN_INPUTS=0
set CT_SHUFFLE_TRAIN_INPUTS_MODE=

for %i in (0 1 2 3 4) do python runnables\train_multi.py ^
  +experiment=eegmmidb_ct_raw8_seed200_manualcw ^
  dataset.data_dir="C:\Projects\CausalTransformer\data\eegmmidb" ^
  dataset.stratify_subject_split=true ^
  dataset.cache=false ^
  fold_index=%i ^
  exp.seed=600 ^
  dataset.num_workers=0 exp.num_workers=0 ^
  exp.persistent_workers=false
```

**Note sur les noms d'expérience.** Le nom `+experiment=...` est un alias vers un preset Hydra et peut contenir un numéro de seed qui ne correspond pas au seed réellement exécuté. Le seed effectif est toujours celui passé via la ligne de commande (p. ex. `exp.seed=600`).

**Export des résultats MLflow.**

Listing 4 – Commande (extrait).

```
set "PYTHONPATH=."
python scripts\export_results.py --experiment-name "CT/eegmmidb" --last-n 20
    --post-analyze True
```

## 9.4 Figures complémentaires

**Résumé de la section** — Cette annexe regroupe des figures utiles mais non essentielles au corps du rapport : courbes d'apprentissage et matrices de confusion additionnelles (niveaux record/window). Les versions normalisées sont incluses ci-dessous ; lorsque disponibles, des versions brutes (comptes) existent aussi dans `report_latex/figures/`. Toutes ces figures proviennent de `report_latex/figures/` et sont soit copiées depuis `mlflow_exports`, soit générées à partir des exports (script `scripts/make_report_assets.py`).

### 9.4.1 Courbes d'entraînement (exemple, fold 0, seed=600)

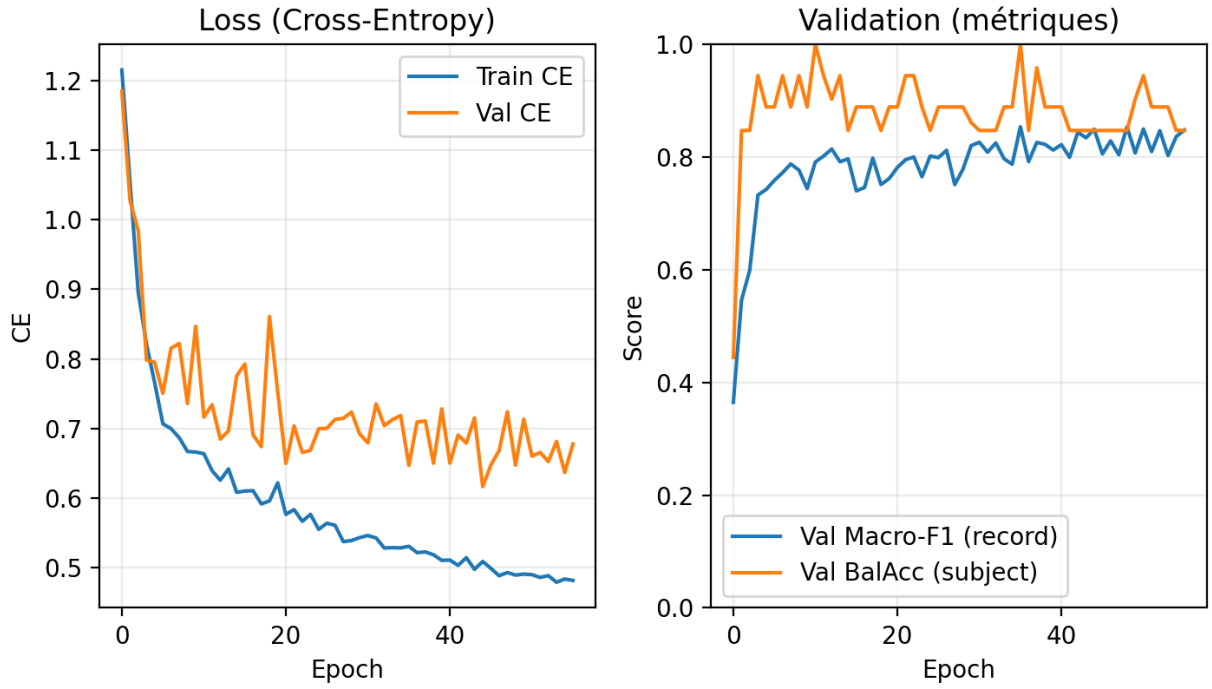
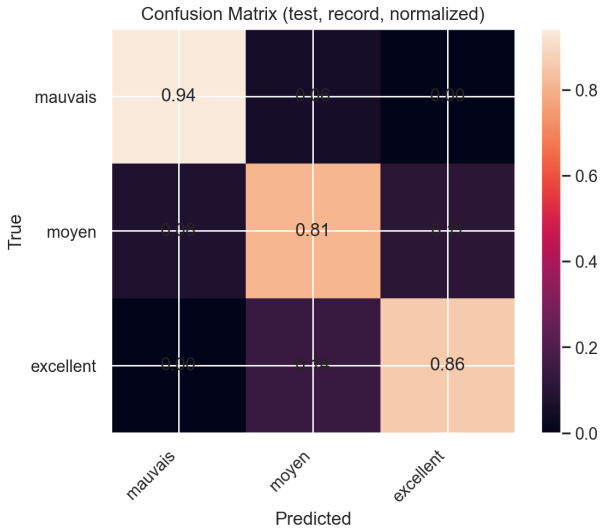
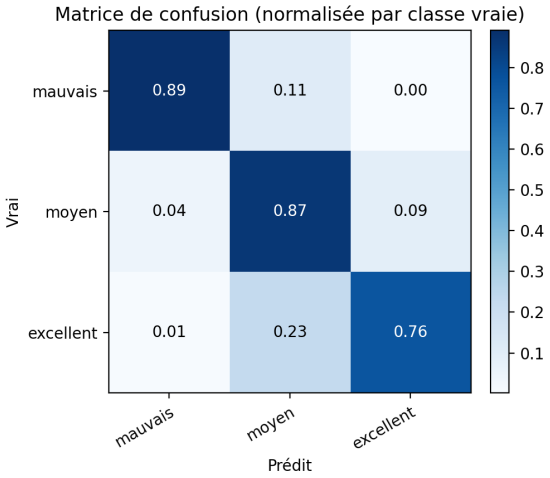



FIGURE 8 – Courbes d'entraînement/validation (fold 0, seed=600) générées à partir des logs MLflow locaux (`mlruns`).

### 9.4.2 Matrices de confusion additionnelles (seed=600)




(a) Record, fold 0, normalisée




(b) Record, agrégée 5 folds, normalisée

FIGURE 9 – Matrices de confusion (niveau *record*, test, seed=600).

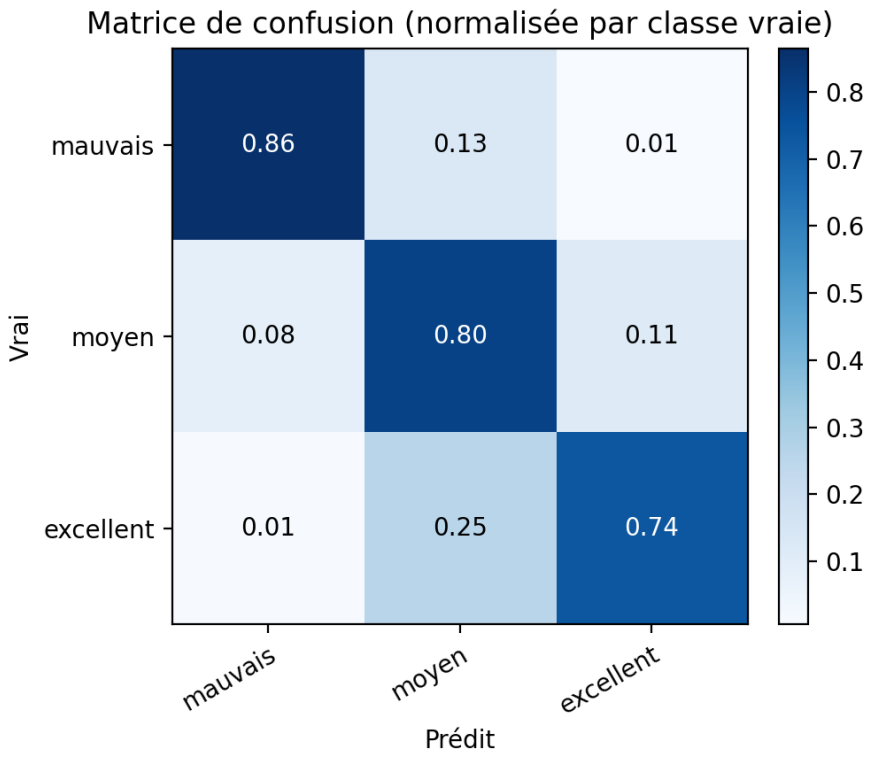



FIGURE 10 – Matrice de confusion normalisée (niveau *window*, test, agrégée 5 folds, seed=600).

## 9.5 Outils & environnement

**Résumé de la section** — Cette annexe liste l'environnement logiciel tel que documenté dans le dépôt (dépendances pinées) et tel qu'exporté depuis l'environnement Conda Windows utilisé pour entraîner les runs finaux. L'objectif est de rendre explicite tout écart potentiel entre versions «  attendues » et versions réellement utilisées.

— **Dépendances pinées (référence dépôt)** : p. ex. `torch==1.13.1`, `pytorch-lightning==1.4.5`, `hydra-core==1.3.2`, `mlflow==2.12.2`, `mne==1.4.2`.
— **Versions effectives (env Conda Windows)** : extrait ci-dessous.

Listing 5 – Versions logicielles exportées depuis l'environnement d'entraînement.

```
python 3.8.20 (default, Oct 3 2024, 15:19:54) [MSC v.1929 64 bit (AMD64)]
platform Windows-10-10.0.26100-SP0
numpy 1.24.3
pandas 2.0.3
mne 1.6.1
mlflow 2.17.2
hydra 1.3.2
torch 2.4.1
lightning 1.4.5
sklearn 1.3.2
```

## 9.6 Différences vs repo original

**Résumé de la section** — Cette annexe synthétise les différences majeures constatées via l'inspection du code, des configurations et des exports (sans diff git).

— **Ajout d'un dataset EEGMMIDB** : `src/data/physionet_eegmmidb/` (scan EDF, fenêtrage, features, labels heuristiques).
— **Passage à la classification** : configuration `dim_outcomes=3`, sorties logits + cross-entropy (`src/models/time_varying_model.py`).
— **Neutralisation de la logique "shock target" / traitements** : adaptation vers une tâche sans traitements (`treatment_mode=none`, `dim_treatments=0`).
— **Baselines EEG** : `src/models/baselines.py` et `runnables/train_baselines.py`.
— **Exports MLflow enrichis** : métriques, prédictions multi-niveau, matrices de confusion et rapports exportés (`mlflow_exports`).

## 9.7 Checklist de conformité (rapport final)

**Résumé de la section** — Cette checklist remplace les dépendances vers des documents externes et rend explicites les points couverts et les limites restantes, sans nécessiter l'ouverture de fichiers séparés.

— **Auto-suffisant (lecture).** Les commandes de reproduction (Windows), les hyperparamètres clés, les résultats et les contrôles de validité nécessaires à la compréhension sont inclus dans ce PDF.

— **Citations dataset.** EEGMMIDB/EEGBCI (PhysioNet) est cité avec DOI **10.13026/C28G6P** et licence **ODC-By 1.0** (Bibliographie).

— **Contrôles de validité (anti-fuite).** Permutation de labels et découplage inter-batch sont archivés et résumés en Table 9.

— **Bibliographie.** Les références clés (Causal Transformer, EEGMMIDB, BCI2000, PhysioNet, EEGNet) sont présentes sans stubs.

— **Limites restantes (optionnelles).** Les tests QC additionnels (prédiction d'ID sujet, ablation de canaux) ne sont pas inclus dans l'export MLflow utilisé dans ce rapport.

# Références


[1] Eeg motor movement/imagery dataset (eegmmidb/eegbci). PhysioNet, 2009. URL https://physionet.org/content/eegmmidb/1.0.0/. Version 1.0.0 (9 sept. 2009). Licence : Open Data Commons Attribution License v1.0 (ODC-By).

[2] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. C. Ivanov, R. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet : Components of a new research resource for complex physiologic signals. *Circulation*, 101 (23) :e215–e220, 2000. Citation standard PhysioNet telle que fournie dans la documentation EEGMMIDB locale ; RRID :SCR_007345.

[3] Vernon J. Lawhern, Amelia J. Solon, Nicholas R. Waytowich, Stephen M. Gordon, Chou P. Hung, and Brent J. Lance. Eegnet : a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 15(5) :056013, 2018. doi : 10.1088/1741-2552/aace8c. URL https://doi.org/10.1088/1741-2552/aace8c.

[4] Valentyn Melnychuk. Causaltransformer. Dépôt GitHub (code source), 2022. URL https://github.com/Valentyn1997/CausalTransformer. Consulté le 2026-01-29.

[5] Valentyn Melnychuk, Dennis Frauen, and Stefan Feuerriegel. Causal transformer for estimating counterfactual outcomes. arXiv, 2022. URL https://arxiv.org/abs/2204.07258.

[6] OpenAI. Codex cli. Software and documentation. URL https://developers.openai.com/codex/cli/. Consulté le 2026-01-29.

[7] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw. Bci2000 : A general-purpose brain-computer interface (bci) system. *IEEE Transactions on Biomedical Engineering*, 51(6) :1034–1043, 2004. doi : 10.1109/TBME.2004.827072.