Rapport Climate Projection

Mattéo André, Brahim Benali, Nathanaël Jacquier,
Julien Laborde-Peyré, Mattéo Pégard, Paul Thibon.

Mai 2025

Résumé du projet :
Le projet présenté dans ce rapport est un projet d'émulation climatique fondé sur des méthodes de machine learning. Il a été mené par les 6 membres dont les noms figurent ci-dessus et encadré par Xujia Zhu, que nous tenons à remercier chaleureusement pour son aide tout au long de ce projet qui a été très stimulant pour tous. Ce travail est principalement basé sur l'article [3], qui propose une démarche pour émuler le simulateur climatique NorESM2 sur diverses scénarios climatiques. Dans un premier temps, nous avons mené une étude de l'article [3], des programmes associés disponibles en ligne (voir [2]) ainsi que de la démarche menée par ses auteurs. Ensuite nous avons cherché à reproduire leurs résultats dans un nouveau framework : PyTorch. Enfin, nous avons cherché à améliorer ces résultats en faisant évoluer les différents codes initiaux. Techniquement, nous avons exploré deux démarches; d'une part, les réseaux de neurones, et d'autre part, les processus gaussiens. Ce rapport de divise en trois parties principales : tout d'abord une introduction au projet, puis une étude de la démarche de l'article [3] ainsi que des éléments techniques employés, et pour finir l'ensemble des résultats obtenus par nos simulations.

Contents

1 Introduction

1.1 Contexte

Dans le cadre du dérèglement climatique actuel, il est fondamental de pouvoir prédire les évolutions futures du climat afin d'orienter efficacement les politiques publiques. Pour cela, des modèles climatiques complets et puissants, appelés modèles de système terrestre (Earth System Models, ESMs), ont été développés. Parmi eux figure NorESM2, la seconde version du modèle climatique norvégien, qui fait partie des modèles de référence dans les simulations climatiques globales.
Cependant, ces ESMs sont extrêmement coûteux en temps de calcul, en ressources informatiques et en énergie, si bien qu'ils ne peuvent être utilisés que sur un nombre limité de scénarios socio-économiques (les Shared Socioeconomic Pathways, ou SSPs). Pour contourner cette limitation, certaines approches actuelles reposent sur des méthodes simplifiées (modèles d'impulsion unidimensionnelle, moyennes spatiales, etc.) qui ne permettent ni de capturer les dynamiques climatiques régionales, ni d'obtenir une précision suffisante.

1.2 Objectif du projet

Ce projet vise à développer un émulateur climatique, c'est-à-dire un modèle d'apprentissage automatique capable de reproduire les sorties d'un modèle climatique complexe, mais à un coût de calcul très réduit. Contrairement aux modèles ESMs comme NorESM2, qui simulent explicitement les processus physiques, chimiques et biologiques du climat, un émulateur apprend à prédire directement l'évolution de certaines variables climatiques clés (telles que la température de surface, les précipitations ou le forçage radiatif), à partir d'entrées plus simples comme les émissions de gaz à effet de serre, les concentrations de CO_2 ou les indices de forçage.
Dans ce projet, nous nous inscrivons dans la continuité du benchmark ClimateBench [3], qui propose un cadre standardisé pour l'évaluation de tels émulateurs. Notre approche consiste à reproduire certaines architectures proposées dans ce benchmark (réseaux de neurones simples, processus gaussiens) en les réimplémentant et en les réentraînant spécifiquement sur les données issues du modèle NorESM2, qui fournit ici notre vérité terrain.
L'objectif est double :
- Évaluer la capacité des modèles à prédire fidèlement l'évolution de variables climatiques en réponse à différents scénarios d'émissions (SSPs), en comparaison avec les sorties originales de NorESM2.
- Améliorer les performances des modèles existants en affinant les architectures et les méthodes d'apprentissage, de façon à obtenir des prédictions plus proches des ESMs, tout en conservant un temps de calcul négligeable.

1.3 Formalisation mathématique

Réseaux de neurones

Le problème mathématique sous-jacent ici consiste à optimiser les poids pour trois types de réseaux de neurones: les réseaux de neurones convolutifs (CNN), les réseaux de neurones récurrents (RNN) et les couches denses.

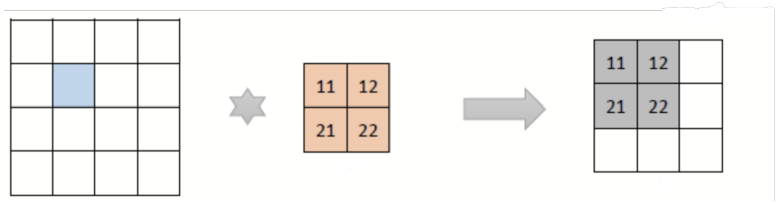

Figure 1: Exemple de convolution

Pour les réseaux convolutifs, il y a des couches de convolutions mais aussi des couches de pooling, ces dernière ne comportant pas de poids. Soit une image d'entrée $X \in \mathbb{R}^{H \times W}$ et un filtre de convolution $W \in \mathbb{R}^{k \times k}$. La convolution produit une image de sortie $X' \in \mathbb{R}^{H' \times W'}$ telle que :

$$X'_{i,j} = \sum_{n=0}^{k-1} \sum_{m=0}^{k-1} W_{n,m} X_{i+n,j+m} + b,$$

où $b \in \mathbb{R}$ est un biais.

Ensuite, pour introduire de la non-linéairité, on choisit comme fonction d'activation la fonction ReLu (Rectified Linear Unit) définie comme suit :

$$ReLu(X'_{i,j}) = max(0, X'_{i,j})$$

Ensuite, nous utilisons du *pooling*, et plus précisément du *average pooling*, qui consiste à remplace chaque carré de l'image de taille $l \times l$ par un seul pixel dont la valeur est la moyenne des valeurs des pixel du carré. Cette technique permet notamment de réduire la taille d'une image. Lorsque tous les pixels sont moyennés en un seul, on parle de *global average pooling*.

Pour continuer à réduire la taille des cartes intermédiaires tout en conservant l'information globale, nous utilisons une technique appelée *adaptive average pooling*. Contrairement au *average pooling* classique, qui applique un filtre de taille fixe (par exemple 2×2), le `AdaptiveAvgPool2d` de PyTorch permet de spécifier directement la taille de sortie souhaitée, et calcule automatiquement la taille du filtre et le pas nécessaires pour y parvenir.
Par exemple, pour une image de taille $H \times W$, si l'on souhaite obtenir une carte de sortie de taille $H' \times W'$, la fonction `AdaptiveAvgPool2d` partitionne l'image en $H' \times W'$ blocs, et remplace chaque bloc par la moyenne de ses valeurs. Cela revient à une généralisation du *global average pooling*, qui correspond au cas où $H' = W' = 1$. Ce type de pooling est particulièrement utile dans les architectures profondes, car il permet de réduire progressivement la taille des tenseurs intermédiaires à des dimensions fixées, sans avoir à ajuster manuellement les hyperparamètres des couches précédentes.

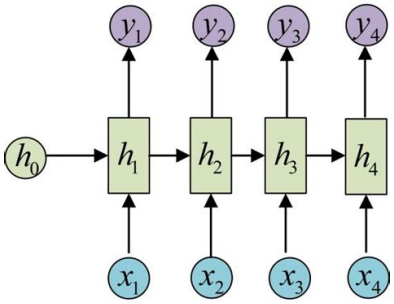

Figure 2: Exemple de réseau récurrent

Les réseaux récurrents sont définis selon la structure ci-dessus, et en particulier comportent des états cachés h_t à chaque instant. Les règles de mise à jour sont les suivante.

État caché :

$$h_t = ReLu(W_{x,h}x_t + W_{h,h}h_{t-1} + b_h)$$

Sortie :

$$y_t = ReLu(W_{h,y}h_{t-1} + b_y)$$

Les réseaux récurrents sont exposés aux problèmes de *gradient exploding* ou *gradient vanishing* potentiellement engendrés par le produit répété de matrices jacobiennes. Pour pallier ce problème, on utilisera des architectures appelées LSTM et GRU qui reprennent cette architecture de réseau récurrent et l'améliore avec des mécanisme de portes.

Pour la fonction de perte, on démarre avec la RMSE (Root Mean Squared Error) définie comme suit :

$$RMSE(\hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2},$$

n étant la taille de y.

Enfin, il y a une phase de rétropropagation dans laquelle la variation de la fonction de perte par rapport aux poids est calculée par règle de la chaîne. Grâce à cela, on peut mettre à jour les poids de sorte à faire diminuer la fonction de perte. L'optimisateur qu'on utilise est Adam, qui adapte localement le taux d'apprentissage (pas de formule explicite donc).

Processus gaussiens

Voir la section 2.5.2

Optimisation du modèle

Voir la section 2.3 pour la définition des NRMSE

L'objectif quantitatif principal, comme expliqué dans la section précédente, est de minimiser les métriques définies dans la section 2.3, tout en conservant une variance réduite et en essayant de simplifier les modèles (diminuer le nombre de paramètres). L'objectif pourrait se formaliser ainsi :

$$min_{\mathcal{M},\sigma<\sigma_{max},n_{var}<n_0} NRMSE(\mathcal{M}),$$

où $\mathcal{M}$ désigne un modèle donné, σ_0 une variance à ne pas dépasser (*overfitting*) et n_0 un nombre de paramètres maximum. Cette formulation reste à titre indicative, nous n'avons pas fixé de variance ou de nombre de paramètres à ne pas dépasser.

2 ClimateBench v1.0

Cette partie présente en détail les éléments proposés dans le cadre du benchmark *ClimateBench* [3], sur lequel se fonde notre travail. Elle a pour objectif de décrire la méthodologie suivie dans ce papier, incluant la préparation des données, la définition rigoureuse des métriques d'évaluation, ainsi que les architectures de modèles d'émulateurs mises en place. L'ensemble de ces éléments constitue la base expérimentale que nous avons reproduite, puis utilisée comme point de départ pour nos propres améliorations.

2.1 Description des données

Le dataset utilisé est constitué de simulations issues du ClimateBench Data Set, un banc d'essai standardisé pour entraîner des modèles de machine learning pour le climat.

Table 1
Details of Post-Processed Simulations Provided as Part of the ClimateBench Data Set

Protocol	Experiment	Period	Notes
ScenarioMIP (O'Neill et al., 2016)	ssp126	2015–2100	A high ambition scenario designed to produce significantly less than 2° warming by 2100.
	ssp245	2015–2100	Designed to represent a medium forcing future scenario. This is the test scenario to be held back for evaluation
	ssp370	2015–2100	A medium-high forcing scenario with high emissions of near-term climate forcers (NTCF) such as methane and aerosol
	ssp370-lowNTCF	2015–2054	Variation of SSP370 with lower emissions of aerosol and their precursors
	ssp585	2015–2100	This scenario represents the high end of the range of future pathways in the IAM literature and leads to a very large forcing of 8.5 Wm^{-2} in 2100
CMIP6 (Eyring et al., 2016)	historical	1850–2014	A simulation using historical emissions of all forcing agents designed to recreate the historically observed climate
	abrupt-4xCO_2[a]	500 years	Idealized simulation in which CO_2 is abruptly quadrupled. Other forcing agents remain unchanged
	1pctCO_2[a]	150 years	Idealized simulation in which CO_2 is gradually increased by 1%/year. Other forcing agents remain unchanged
	piControl[a]	500 years	Baseline simulation in which all forcing agents remain unchanged
DAMIP (Gillett et al., 2016)	hist-GHG	1850–2014	A historical simulation with varying concentrations for CO_2 and other long-lived greenhouse-gases (only)
	hist-aer	1850–2014	A historical simulation only forced by changes in anthropogenic aerosol
	ssp245-aer	2015–2100	A medium forcing scenario with only changes in anthropogenic aerosol, which provides an alternative test scenario for emulator evaluation

Figure 3: Tous les scénarios de CMIP 6 utilisés

Les simulations de la section ScenarioMIP sont des projections basées sur des scénarios socio-économiques ("Shared Socio-economic Pathways", SSPs). Ces scénarios sont classés selon leur caractère optimiste ou pessimiste par leur premier chiffre qui va de 1 (scénario optimiste dans lequel les émissions de GES sont rapidement limitées) à 5 (scénario pessimiste dans lequel on continue à polluer énormément). De plus, les deux chiffres suivants désignent le forçage radiatif additionnel en 2100 à la suite de ce scénario. Par exemple, le scénario le plus pessimiste est ssp585 selon lequel on obtiendrait un forçage radiatif augmenté de $+8,5W/m^2$ en 2100. Le scénario ssp245 est conservé pour l'évaluation de l'émulateur. Étant plutôt médian, ce choix permet d'avoir une interpolation plutôt qu'une extrapolation lors de l'évaluation du modèle.

Les simulations des autres sections sont des simulations plus "extrêmes" aux limites de ce qui est possible (abrupt-4xCO_2), des simulations historiques ou des simulations partielles qui ne prennent en compte que certains forçages. Seuls quatre forçages sont pris en compte : d'une part les gaz à effet de serre représentés par le méthane et le dioxyde de carbone, d'autre part les aérosols représenté par les dioxyde de soufre (en fait précurseur d'aérosols) et le noir de carbone (particules de suie en suspension). Enfin, le scénario piControl est un scénario de référence dans lequel tous les forçages restent identiques à l'époque préindustrielle.

Pour tous ces scénarios, le modèle climatique NorESM2 retourne 4 grandeurs sous forme de moyennes annuelles représentées spatialement sur le globe : la température (T), les précipitations (P), le 90e percentile des précipitations quotidiennes (PR90) et la plage diurne des températures (DTR). Les valeurs issues du scénarios piControl

sont soustraites pour chaque autre scénario afin de représenter la différence par rapport à l'ère pré-industrielle. De plus, pour prendre en compte la variabilité interne du climat, les auteurs de [3] prennent trois état initiaux séparés de 30 ans pour le scénario piControl. Ce qu'ils cherchent à faire par la suite, c'est de proposer une approche efficace et fiable pour obtenir des cartes annuelles de ces quatre grandeurs pour n'importe quel scénario d'émission.

2.2 Préparation des données

Les données subissent en premier lieu un preprocessing. Pour les émissions de dioxyde de carbone et de méthane, les données sont extraites du projet input4MIPS. Les émissions moyennes globales sont extraites et moyennées temporellement et spatialement (parfois interpolées temporellement si nécessaire) afin d'obtenir les émissions totales par an exprimées en gigatonnes. En particulier, les émissions de dioxyde de carbone sont exprimées de manière cumulatives.

Pour les aérosols (dioxyde de soufre et noir de carbone), les données sont dérivées du CEDS (Community Emissions Data System). Les émissions sont aussi exprimées par année mais elles ne sont pas moyennées spatialement : pour chaque année, on a une carte des émissions sur l'ensemble des planisphère.

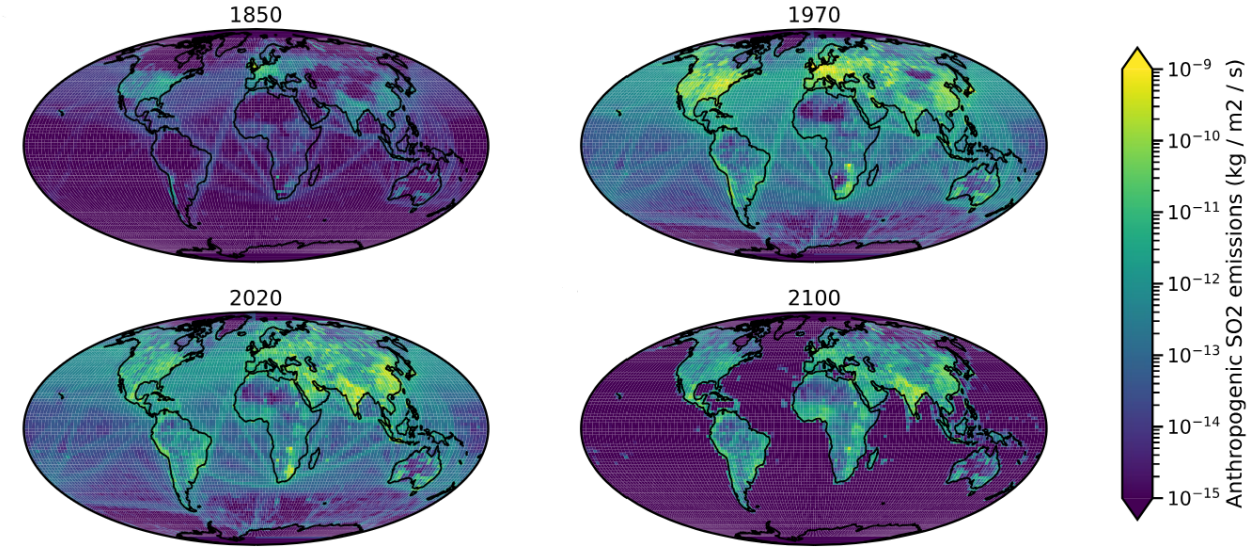

Figure 4: Exemple de données d'entrée pour le SO2 présenté dans l'article

Pour les simulations idéalisées (abrupt 4x-CO_2 et $1pctCO_2$), les données sont calculées à partir de piControl pour le CO_2 et simplement nulles pour les trois autres émissions.

Dans le code de départ sur lequel on se base, la préparation des données est assurée par "prepare_data.py" et "prep_input_data.ipynb". Le premier programme télécharge les sorties du modèle climatique NorESM2 depuis son serveur THREDDS avec la bibliothèque siphon pour l'ensemble des scénarios et des variables climatiques de sorties d'intérêt. Ensuite, il met en forme ces variables et retourne un fichier NetCDF (stockage de données multidimensionnelles) qui contient pour chaque scénario les valeurs des 4 grandeurs de sorties (T, P, PR90 et DTR) moyennées annuellement selon différentes latitudes et longitudes sur la plage de temps souhaitée. Le second programme prépare, organise et sauvegarde toutes les données d'entrée (CO2, CH4, SO2, bc) pour tous les scénarios d'intérêt en harmonisant et regroupant les différentes séries temporelles (unité Gt/an, grille commune, interpolation temporelle, membres d'ensemble...), puis renvoie des fichiers spécifiques sous forme d'archives compressées pour l'entraînement et le test (ssp-245).

Les données d'entrée sont représentées par un tenseur de forme $(726, 10, 96, 144, 4)$, où :

- 726 : nombre total d'échantillons (séquences extraites par fenêtre glissante),
- 10 : nombre de pas de temps par séquence (taille du *slider* temporel),
- 96×144 : grille spatiale (latitude $\times$ longitude),

- 4 : nombre de variables climatiques utilisées (par exemple CO_2, CH_4, SO_2, BC).

Le *slider* temporel de taille 10 permet de capturer la dynamique locale des phénomènes en exploitant la structure spatio-temporelle des données. Chaque échantillon fournit ainsi au modèle une séquence temporelle complète d'évolution des variables sur l'ensemble du globe.

2.3 Métriques utilisées

Voici la métrique proposée par [3] pour évaluer les performances des émulateurs climatiques. Elle combine deux sources d'erreur : l'erreur spatiale (locale) et l'erreur sur la moyenne globale. L'objectif est de capturer à la fois la précision des valeurs régionales et la cohérence des tendances globales, tout en conservant une formulation simple pouvant être utilisée comme fonction de coût.
La métrique combinée, notée NRMSE_t, est définie comme suit :

$$\text{NRMSE}_t = \text{NRMSE}_s + \alpha \cdot \text{NRMSE}_g$$

où :

- NRMSE_s est l'erreur quadratique moyenne normalisée calculée point par point,
- NRMSE_g est l'erreur quadratique moyenne normalisée sur la moyenne globale,
- $\alpha = 5$ est un coefficient empirique permettant d'équilibrer le poids des deux termes.

Les deux composantes sont données par :

$$\text{NRMSE}_s = \sqrt{\frac{\langle (x_{i,j,t} - y_{i,j,t})^2 \rangle_t}{\langle y_{i,j,t} \rangle_t}} \quad \text{et} \quad \text{NRMSE}_g = \sqrt{\frac{(\langle x_{i,j,t} \rangle - \langle y_{i,j,t} \rangle)_t^2}{\langle y_{i,j,t} \rangle_t}}$$

La moyenne globale $\langle \cdot \rangle$ est pondérée en fonction de la latitude afin de tenir compte de la réduction de surface des cellules vers les pôles :

$$\langle x_{i,j} \rangle = \frac{1}{N_{\text{lat}} N_{\text{lon}}} \sum_{i=1}^{N_{\text{lat}}} \sum_{j=1}^{N_{\text{lon}}} \cos(\text{lat}(i)) \cdot x_{i,j}$$

Les métriques sont calculées sur la période 2080–2100 du scénario cible `ssp245`, les auteurs ayant choisi d'utiliser cette période relativement longue pour réduire l'impact de la variabilité interne. Les RMSE sont normalisées afin de permettre la comparaison entre différentes variables climatiques.

2.4 Réseaux de neurones

Le premier modèle proposé dans [3], que nous cherchons ensuite à améliorer, repose sur une architecture de réseau de neurones structurée en plusieurs blocs successifs, chacun conçu pour capturer un aspect particulier des données climatiques spatio-temporelles.

Extraction spatiale Les corrélations spatiales sont apprises par un réseau convolutif (CNN). Plus précisément, chaque pas de temps est traité individuellement par une couche de convolution 2D appliquée aux 4 variables climatiques (CO_2, CH_4, SO_2, BC). Cette couche utilise 20 filtres de taille 3×3 avec une activation ReLU. Elle est suivie d'un *average pooling* 2×2 pour réduire la résolution, puis d'un *global average pooling* qui résume chaque carte d'activation en une seule valeur.
Les données d'entrée ont la forme $(N, T, 96, 144, 4)$, où N est le nombre total de séquences (échantillons), $T = 10$ la longueur temporelle de la séquence, 96×144 la grille spatiale, et 4 le nombre de canaux (variables). En sortie de la partie convolutive, chaque pas de temps est réduit à un vecteur de dimension 20, ce qui donne un tenseur de forme $(N, T, 20)$.

Extraction temporelle Les dépendances temporelles sont modélisées à l'aide d'un LSTM (*Long Short-Term Memory*), une architecture récurrente conçue pour capter les dynamiques temporelles longues. L'entrée du LSTM est le tenseur $(N, T, 20)$, et sa sortie est un vecteur de dimension $(N, 25)$, correspondant au dernier état caché (25 unités) pour chaque séquence.

Prédiction La sortie du LSTM est ensuite traitée par une couche entièrement connectée (dense), suivie d'un reformatage pour reconstruire un champ spatial. La sortie finale du réseau a la forme $(N, 1, 96, 144)$: une carte prédite (par exemple, de température) pour chaque séquence d'entrée.

Fonction de perte La fonction de perte utilisée pour entraîner ce modèle est la racine de l'erreur quadratique moyenne (RMSE), ce qui permet de pénaliser davantage les erreurs importantes.

2.5 Processus gaussiens

2.5.1 Introduction

Le second modèle étudié, proposé initialement dans [3], repose sur une approche bayésienne par processus gaussiens (GP). Contrairement aux réseaux de neurones, les GP permettent de modéliser directement une distribution de probabilité sur les sorties du modèle, offrant ainsi une estimation naturelle de l'incertitude et une flexibilité importante grâce au choix du noyau.

2.5.2 Définition d'un processus gaussien

Un *processus gaussien* (ou *Gaussian Process*, GP) est une généralisation infinie de la loi normale multivariée. Il s'agit d'une distribution de probabilité définie sur un espace de fonctions. Formellement, un GP est une collection de variables aléatoires, dont toute sous-collection finie suit une loi normale multivariée. Un processus gaussien est entièrement défini par sa fonction moyenne $m(x)$ et sa fonction de covariance (ou noyau) $k(x, x')$:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

La fonction moyenne $m : \mathcal{X} \to \mathbb{R}$ et le noyau $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ sont choisis a priori. En pratique, on suppose souvent $m(x) = 0$ sans perte de généralité, car toute tendance peut être supprimée des données en amont.
Étant donné un ensemble de données constitué de n observations $\{(x_i, y_i)\}, i = 1, ..., n$, où $y_i = f(x_i) + \varepsilon$ avec un bruit gaussien $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$, la prédiction d'un GP en un nouveau point x^* suit une loi normale dont la moyenne et la variance s'écrivent explicitement :

$$\mu(x) = k(x, X)[K(X, X) + \sigma_n^2 I]^{-1} y,$$
$$\sigma^2(x) = k(x, x) - k(x, X)[K(X, X) + \sigma_n^2 I]^{-1} k(X, x^*),$$

où $X = [x_1, \ldots, x_n]$ est la matrice des entrées d'entraînement, $K(X, X)$ la matrice de Gram associée au noyau k, et $k(x, X)$ le vecteur des covariances entre x et les points d'entraînement.

Le principal inconvénient des GP réside dans leur complexité computationnelle : l'inversion de la matrice de covariance $K(X, X)$ est en $\mathcal{O}(n^3)$, ce qui rend les GP coûteux à l'échelle de grands jeux de données.
Dans le code initial, le modèle GP est entraîné directement pour prédire ces champs dans leur forme brute, c'est-à-dire comme un vecteur de grande dimension, ce qui nous a poussé à explorer la piste d'une ACP sur les données de sortie.

2.5.3 Définition de l'analyse par composante principales

La piste de recherche principale que nous avons adopté repose sur l'utilisation d'une réduction de dimension par analyse en composantes principales (ACP), appliquée aux champs de sortie.

L'Analyse en Composantes Principales (ACP) est une méthode de réduction de dimension fondée sur l'algèbre linéaire. Elle consiste à projeter un ensemble de données multivariées $\mathbf{X} \in \mathbb{R}^{n \times p}$ (avec $n = 783$ échantillons et $p = 12$ variables dans notre cas) sur une base orthonormée de vecteurs appelés composantes principales, de manière à capturer la variance maximale de l'information avec un nombre réduit de dimensions.
Mathématiquement, l'ACP consiste à diagonaliser la matrice de covariance centrée des données :

$$\mathbf{S} = \frac{1}{n} \mathbf{X}^\top \mathbf{X} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top$$

où $\mathbf{V}$ contient les vecteurs propres (EOFs dans notre cas), et $\mathbf{\Lambda}$ les valeurs propres associées (variances expliquées). Les composantes principales sont alors obtenues par projection :

$$\mathbf{Z} = \mathbf{X}\mathbf{V}$$

Les k premiers vecteurs propres, associés aux plus grandes valeurs propres, constituent une base optimale (au sens des moindres carrés) pour représenter les données dans un espace de dimension réduite.

2.5.4 Application au jeu de données

Dans notre cas, les données climatiques (**X**) sont des champs spatialisés vectorisés, et l'ACP permet de capturer les modes dominants de variabilité spatiale. Les vecteurs propres **V** correspondent alors aux Empirical Orthogonal Functions (EOFs), et les coefficients de projection **Z** aux scores EOFs.

Chaque carte annuelle est projetée dans l'espace de ses composantes principales, ce qui permet de représenter l'information spatiale sous forme d'un vecteur de dimension réduite. Le nombre de composantes conservées est différent selon la variable (10 pour TAS contre 170 pour PR, voir partie 3.3.2), ce qui permet de préserver l'essentiel de la variance climatique.

Cette projection dans l'espace des composantes principales permet d'espérer une amélioration de l'apprentissage en forçant le modèle à se concentrer sur les structures spatiales dominantes de la variabilité climatique.
L'apprentissage des GP se fera donc sur les scores EOFs **Z**.

2.6 Codes initiaux

2.6.1 Réseaux de neurones

Nous décrivons ici le programme donc nous souhaitons améliorer les performances et réduire les coûts (nombre de paramètres...). Celui-ci se base sur le framework de deep learning Tensorflow.
Dans un premier temps, les données sont téléchargées, mises sous le bon format (prise en compte de la période historique ou non, bon pas de temps...), et séparées en données d'entraînement d'entrée et en données d'entraînement de sortie (qui sont les labels de l'apprentissage supervisé).

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Activation, Conv2D, Flatten, Input, Reshape, AveragePooling2D, MaxPooling2D
from tensorflow.keras.regularizers import l2

import random
seed = 6
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

Figure 5: Imports Tensorflow utilisés pour le réseau de neurones initial

Ensuite, on importe TensorFlow et notamment Keras, qui permet de créer des modèles de deep learning de manière simple et efficace. En particulier, son modèle Sequential permet de construire un modèle de manière linéaire, c'est-à-dire couche par couche. Toutes les couches utiles pour construire facilement le CNN-LSTM sont importées dans la commande suivante. Le dernier import depuis TensorFlow concerne la régularisation l2 (Ridge régression), qui peut être utile pour diminuer la variance du modèle en pénalisant les poids trop grands.

```
cnn_model = Sequential()
cnn_model.add(Input(shape=(slider, 96, 144, 4)))
cnn_model.add(TimeDistributed(Conv2D(20, (3, 3), padding='same', activation='relu'), input_shape=(slider, 96, 144, 4)))
cnn_model.add(TimeDistributed(AveragePooling2D(2)))
cnn_model.add(TimeDistributed(GlobalAveragePooling2D()))
cnn_model.add(LSTM(25, activation='relu'))
cnn_model.add(Dense(1*96*144))
cnn_model.add(Activation('linear'))
cnn_model.add(Reshape((1, 96, 144)))
```

Figure 6: Construction du modèle initial

Ensuite, le backend de Keras et le modèle sont réinitialisés. On construit alors couche par couche le modèle de réseau de neurones décrit dans la section 2.3. Celui-ci est compilé avec l'optimizer Root Mean Square Propaga-

tion et la fonction de perte quadratique. Enfin, le modèle est entraîné avec .fit.

Pour finir, on réalise la prédiction sur les données d'entrée de test pré-traitées avec .predict et on met le résultat sous la bonne forme avec un xarray. On peut ensuite reprendre l'ensemble du programme en bouclant sur l'ensemble des variables à prédire afin d'avoir un code complet.

2.6.2 Processus gaussiens

Le second programme de référence étudié repose sur une approche par processus gaussiens (GP) implémentée avec la bibliothèque GPflow, qui s'appuie sur TensorFlow pour les calculs automatiques de gradients et l'optimisation.

Dans un premier temps, les données d'entrée sont préparées via une fonction dédiée du module `utils.py`. Ces données incluent le dioxyde de carbone (CO2) et le méthane (CH4), traités comme des valeurs globales annuelles, ainsi que le dioxyde de soufre (SO2) et le noir de carbone (BC), représentés par des cartes spatio-temporelles.

Ces deux derniers sont compressés à l'aide d'une analyse en composantes principales (EOF, Empirical Orthogonal Functions), ce qui permet d'en extraire les structures dominantes et de les résumer par 5 composantes principales chacun. Cette manipulation est faite dans le fichier auxiliaire `utils.py`, dont un extrait est présenté en figure 7.

```
# Compute EOFs for BC
bc_solver = Eof(X['BC'])
bc_eofs = bc_solver.eofsAsCorrelation(neofs=n_eofs)
bc_pcs = bc_solver.pcs(npcs=n_eofs, pcscaling=1)

# Compute EOFs for SO2
so2_solver = Eof(X['SO2'])
so2_eofs = so2_solver.eofsAsCorrelation(neofs=n_eofs)
so2_pcs = so2_solver.pcs(npcs=n_eofs, pcscaling=1)
```

Figure 7: ACP sur les données d'entrée

En sortie, on cherche à prédire la variable climatique considérée sur toute la grille spatio-temporelle. Comme pour le modèle neuronal, les données de test correspondent au scénario ssp245, utilisé pour l'évaluation du modèle sur des trajectoires climatiques intermédiaires.

Le modèle GP est ensuite construit avec `GPflow`, le code associé étant présenté en figure 8. Il s'agit d'un modèle de régression gaussienne (gpflow.models.GPR) qui prend en entrée les vecteurs de dimension 12, et qui prédit directement un vecteur de sortie de dimension 13 824 (soit 96 latitudes × 144 longitudes). Le noyau utilisé est une somme de quatre noyaux de type Matern 1.5, dont voici la formule :

$$k_X(x, x') = \left(1 + \sqrt{3}\, d(x, x')\right) \exp\left(-\sqrt{3}\, d(x, x')\right),$$

où x et x' sont deux vecteurs d'entrée (inputs), et

$$d(x, x') = \sum_i \frac{|x_i - x'_i|}{l_i}, \text{ avec } l_i \text{ la longueur d'échelle associée à la coordonnée } x_i.$$

Chacun des quatres noyaux est appliqué à une sous-partie des entrées (CO2, CH4, SO2, BC), avec des longueurs d'échelle adaptées à chaque groupe de variables. Ce découpage est géré par l'option $active_{dims}$ de GPflow.

```python
# Make model
np.random.seed(5)
mean = gpflow.mean_functions.Constant()
model = gpflow.models.GPR(data=(X_train.astype(np.float64),
                                y_train_pr.astype(np.float64)),
                          kernel=kernel,
                          mean_function=mean)
```

Figure 8: Implémentation du GP

L'entraînement est ensuite réalisé à l'aide de l'optimiseur `Scipy`, comme présenté en figure 9 avec un nombre d'itérations fixé (par exemple 1000). La fonction à minimiser est l'approximation du log de vraisemblance négatif fournie par GPflow, ce qui correspond à une approche bayésienne rigoureuse. À chaque étape, la trace de la perte est affichée pour suivre la convergence.

```python
# Define optimizer
opt = gpflow.optimizers.Scipy()

# Train model
opt.minimize(model.training_loss,
             variables=model.trainable_variables,
             options=dict(disp=True, maxiter=40))
```

Figure 9: Implémentation de l'optimisation

Une fois le modèle entraîné, on effectue une prédiction sur les données de test, que l'on remet ensuite à l'échelle initiale. Le vecteur de sortie est reformaté en un champ 2D sur la grille globale. Le résultat peut alors être comparé au champ réel pour le même scénario et les mêmes années.

2.7 Résultats du papier

Voici les résultats obtenus par les auteurs de [3] à l'aide de leur implémentation en *TensorFlow*. Les valeurs reportées correspondent aux métriques définies précédemment, calculées sur les 20 dernières années (2080–2100) du scénario `SSP245`, utilisé comme jeu de test dans notre étude.

Table 2
The Spatial, Global and Total NRMSE of the Different Baseline Emulators for the Years 2080–2100 Against the ClimateBench Task of Estimating Key Climate Variables Under Future Scenario SSP245

	NRMSE surface air temperature (1)			NRMSE diurnal temperature range (1)			NRMSE precipitation (1)			NRMSE 90th percentile precipitation (1)		
	Spatial	Global	Total	Spatial	Global	Total	Spatial	Global	Total	Spatial	Global	Total
Gaussian Process	0.109	0.074	0.478	9.207	2.675	22.582	2.341	0.341	4.048	**2.556**	0.429	4.702
Neural Network	**0.107**	**0.044**	**0.327**	9.917	**1.372**	**16.778**	**2.128**	**0.209**	**3.175**	2.610	**0.346**	**4.339**

Figure 10: Résultats obtenus dans l'article pour le réseau de neuronnes et le processus gaussien

3 Amélioration des modèles décrits par le papier scientifique

3.1 Reprise du code initial en PyTorch

3.1.1 Réseaux de neurones

On reprend dans un premier temps strictement la même structure de réseau que celle proposée par [1] et décrite précédemment, mais en changeant de framework : au lieu d'utiliser TensorFlow, on utilise PyTorch. Pour cela, on s'appuie sur le module torch.nn pour construire les réseaux de neurones, et on adopte une approche orientée objet, qui s'intègre naturellement avec la structure modulaire de PyTorch. Nous préférons PyTorch car il est généralement plus intuitif et plus flexible. De plus, PyTorch est très utilisé dans la recherche académique pour sa clarté et sa proximité avec le code Python natif. Il est également le framework privilégié dans l'industrie.

On crée une classe CNNLSTM dans laquelle on définit les mêmes couches que précédemment et où on crée une méthode forward qui fait passer les données dans le réseau de neurones. On crée également deux classes CustomDataset et TestDataset respectivement pour les datasets d'entraînement et de test. On fait ensuite l'entraînement de manière classique avec l'optimizer Adam et la MSELoss. Pour le backward, on utilise notamment les méthodes loss.backward() pour calculer les gradients et optimizer.step() pour mettre à jour les poids.

Pour finir, nous évaluons les performances du modèle en le testant sur le scénario `SSP245`. Les prédictions générées sur ce jeu de test sont ensuite comparées graphiquement aux valeurs de référence issues du modèle NorESM2, afin d'évaluer visuellement la qualité des résultats produits par l'émulateur.

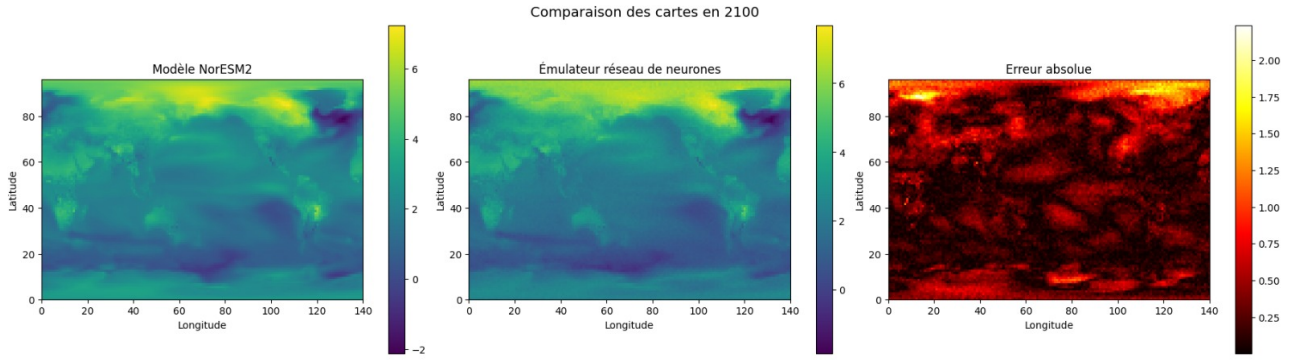

Figure 11: Carte des températures pour la dernière année de simulation(1)

Afin d'obtenir des résultats plus quantifiables, nous calculons également les mêmes métriques que celles définies dans [3].

Métrique	**Valeur moyenne**	**Écart-type**
NRMSE_g	0.0489	$\pm$ 0.0223
NRMSE_s	0.1015	$\pm$ 0.0251
NRMSE_t	0.3459	$\pm$ 0.1335

Table 1: Résultats des métriques NRMSE calculées sur le scénario `SSP245`.

Nombre de paramètres du modèle : 577 284

Analyse des résultats

Pour réduire les incertitudes nous avons entrainé le modèle 10 fois. Cela permet de calculer une valeur moyenne et un écart type. La NRMSE_t obtenue est très proche de celle de [3] mais légèrement plus élevée. Comme l'architecture utilisée est exactement la même que dans l'article, on peut supposer que les auteurs n'ont entrainé le modèle qu'une seule fois, ils ont donc obtenu par hasard une valeur plus basse que la moyenne qu'ils auraient obtenue sur un grand nombre d'entrainements.

3.1.2 Processus gaussiens

Nous reprenons ici l'approche basée sur les processus gaussiens (GP), mais en l'implémentant cette fois à l'aide de la bibliothèque `GPyTorch`.

L'approche décrite dans la suite a été pensée pour pouvoir utiliser relativement simplement la bibliothèque `GPytorch` qui supporte mieux les régressions à 1 dimension. Nous allons voir qu'elle implique une approche théorique légèrement différente de celle de l'article, et ne reproduit donc pas exactement les mêmes résultats. C'est pourquoi les résultats de cette approche seront également discutés dans la partie 3.3, et que l'utilisation de la bibliothèque `GPytorch` doit être vue comme une piste d'amélioration du code initial.

Contrairement à ce dernier, qui implémente une régression multivariée par processus gaussien pour prédire pixel par pixel, notre approche avec `GPyTorch` procède différemment : on considère chaque score EOF (issu d'une ACP sur les champs climatiques) comme une sortie indépendante, et on entraîne un GP distinct pour chaque composante principale. Ces composantes sont issues d'une ACP préalable sur les variables de sorties en entraînement. Autrement dit, notre modèle n'apprend pas une carte complète ou un vecteur multidimensionnel en sortie, mais une série de régressions scalaires indépendantes : pour chaque composante EOF i, on entraîne un GP qui approxime $f_i : \mathbf{x} \mapsto \alpha_i$, où α_i est le score EOF correspondant à l'échantillon $\mathbf{x}$.

Cette hypothèse d'indépendance entre les sorties transforme le problème initialement multivarié en une somme de problèmes univariés où l'optimisation des hyperparamètres devient alors locale à chaque composante. Cela s'oppose à l'approche initiale où une optimisation globale conjointe est réalisée sur les hyperparamètres communs à toutes les sorties, ce qui nécessite en général une modélisation de la covariance inter-composantes.

Néanmoins, dans le cadre d'un apprentissage sur les coefficients des EOFs, il est plus pertinent de faire une optimisation des hyperparamètres propre à chacun des modes puisqu'ils peuvent correspondre à des phénomènes physiques de nature différente.

En résumé, alors que l'approche initiale visait à prédire directement un champ spatialisé ou un vecteur de coefficients simultanément, notre approche découple le problème en une succession de régressions 1D, chacune calibrée indépendamment avec des fonctions de vraisemblance et des hyperparamètres propres.

3.2 Amélioration du réseau de neurones

3.2.1 Fonction de perte pondérée

Une piste d'amélioration du modèle concerne la fonction de perte utilisée pendant l'entraînement. Jusqu'à présent, la racine de l'erreur quadratique moyenne (RMSE) est utilisée sans tenir compte des caractéristiques géographiques des données spatiales. Or, dans le cas de données climatiques projetées sur une grille latitude–longitude régulière, chaque cellule de grille ne représente pas la même surface réelle sur Terre : les cellules situées à proximité des pôles couvrent une surface bien plus réduite que celles situées près de l'équateur.
Pour corriger ce biais géométrique, il est possible de pondérer la MSE par le **cosinus de la latitude**, de manière à refléter la variation de surface des cellules en fonction de leur position géographique. La fonction de perte devient alors :

$$\mathrm{MSE}_{\text{pondérée}} = \frac{1}{N_{\text{lat}} N_{\text{lon}}} \sum_{i=1}^{N_{\text{lat}}} \sum_{j=1}^{N_{\text{lon}}} \cos(\text{lat}_i) \cdot (x_{i,j} - y_{i,j})^2$$

où :

- $x_{i,j}$ désigne la valeur prédite par le modèle au point de grille (i, j),
- $y_{i,j}$ désigne la valeur cible correspondante (issue de NorESM2),
- lat_i est la latitude du point i exprimée en radians,
- N_{lat} et N_{lon} sont respectivement le nombre de latitudes et de longitudes dans la grille.

Cette pondération permet de mieux représenter la contribution réelle de chaque région à l'erreur globale, en accordant plus de poids aux zones équatoriales et tempérées (qui couvrent une plus grande surface du globe), et moins aux régions polaires. Cela conduit à une fonction de coût plus **physiquement pertinente**, en cohérence avec les objectifs globaux du modèle.

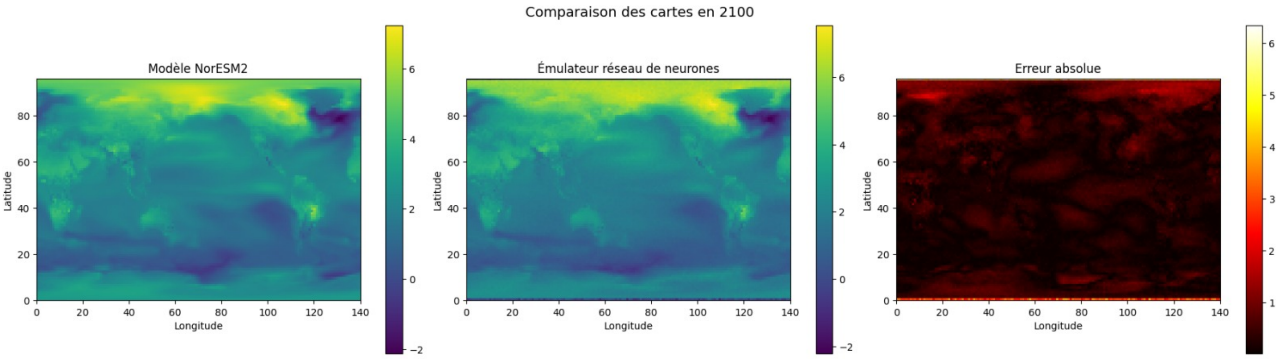

Figure 12: Carte des températures pour la dernière année de simulation(1)

On observe visuellement sur la carte des erreurs absolues que celles-ci semblent moins élevées avec cette nouvelle fonction de perte. On calcule les métriques pour voir si cette impression se confirme. On obtient les résultats suivants :

Métrique	**Valeur moyenne**	**Écart-type**
NRMSE_g	0.0472	$\pm$ 0.0258
NRMSE_s	0.0927	$\pm$ 0.0272
NRMSE_t	0.3288	$\pm$ 0.1546

Table 2: Résultats des métriques NRMSE obtenues après entraînement avec la MSE pondérée par la latitude.

Nombre de paramètres du modèle : 577 284

Analyse des résultats

Même si la variance a légèrement augmenté, l'utilisation de cette fonction de perte pondérée permet de diminuer la NRMSE_t d'environ 5 %. Cependant, bien que ce choix soit physiquement plus cohérent, il n'entraîne pas de gain de performance significatif.

3.2.2 Pénalisation Ridge

Pour améliorer davantage les performances du modèle, et en particulier pour réduire la variance des prédictions, on peut introduire une pénalisation de type Ridge. Cette régularisation consiste à ajouter à la fonction de perte un terme de la forme $\lambda\|\theta\|_2^2$, où θ désigne le vecteur des paramètres du modèle, et λ un coefficient de régularisation contrôlant l'intensité de la pénalisation.

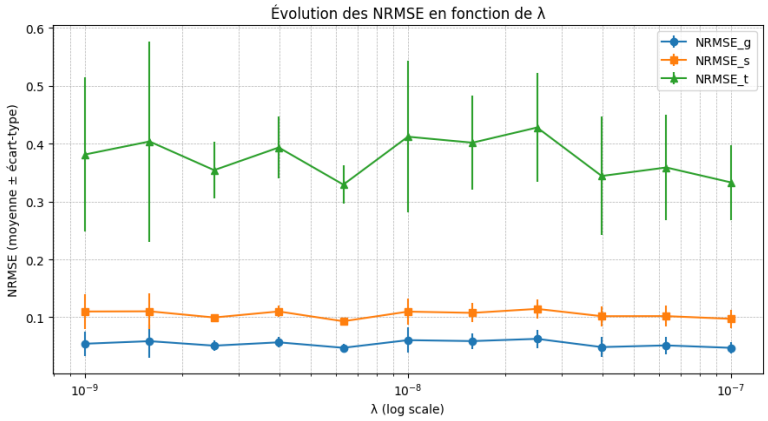

Figure 13: Évolution des métriques en fonction du coefficient de pénalisation

Analyse des résultats
Comme le montre la figure ci-dessus, l'ajout d'une régularisation de type Ridge ne permet pas d'améliorer significativement les performances du modèle, quelle que soit la valeur du coefficient λ. Les variations des métriques NRMSE_g, NRMSE_s et NRMSE_t restent faibles et non concluantes.

Cela peut s'expliquer par plusieurs facteurs : d'une part, le modèle initial ne semble pas souffrir de surapprentissage important, rendant l'effet de la régularisation limité. D'autre part, la taille du jeu d'entraînement est relativement restreinte, ce qui limite la capacité de la régularisation à stabiliser efficacement les poids. Enfin, il est possible que le choix du nombre de paramètres (notamment dans la couche dense finale) soit déjà raisonnablement adapté au volume de données disponibles.

3.2.3 Réduction de dimension pour CO2 et CH4

Contrairement au code de base, étant donné que les émissions de CO2 et de méthane sont moyennées annuellement sur l'ensemble du planisphère, on traite des grandeurs comme des scalaires au lieu de tableau uniforme. Cela permet d'appliquer l'étape de convolution uniquement pour les aérosols.

Cependant, pour quand même prendre en compte les gaz à effet de serre dans la partie CNN avec le passage dans le LSTM (ou le GRU, voir juste après), on rajoute une petite combinaison linéaire en les valeurs d'émission de ces gaz à la sortie du global average pooling.

Le test de cette amélioration est présenté dans la partie suivante, puisque réalisé de manière concomitante avec le GRU.

3.2.4 Remplacement du LSTM par GRU

Le GRU (Gated Recurrent Unit) est un autre type de réseau de neurones récurrents conçus pour bien gérer les dépendances longues au cours du temps tout en étant plus simples que le LSTM (un peu moins de paramètres, dans notre cas 7440 au lieu de 9920, avec une entrée de taille 20 et un état caché de taille 40). Le GRU fusionne en effet l'état caché et l'état mémoire du LSTM. Il est donc plus léger et plus rapide à entraîner, bien que la différence ne soit pas toujours significative devant le reste de paramètres du modèle. Pour un dataset pas trop gros, comme celui sur lequel on travaille, on peut s'attendre à ce que les performances soient équivalentes à celles d'un LSTM.

Voici les résultats de modèles qui, à partir du modèle de base de l'article, incluent la réduction de dimension pour CO2 et CH4 ainsi qu'un GRU à la place du LSTM. Le premier conserve la fonction de perte initiale et une pénalisation nulle, le deuxième utilise la fonction de perte améliorée et le troisième utilise en plus une pénalisation Ridge. Ces résultats sont obtenus pour la variable de sortie "T" correspondant la température.

1er modèle. Nombre de paramètres : 574 644

Métrique	**Valeur moyenne**	**Écart-type**
NRMSE_t	0.312	$\pm$ 0.0369

Table 3: Résultats des métriques NRMSE obtenues en ajoutant la réduction de dimensions et le GRU

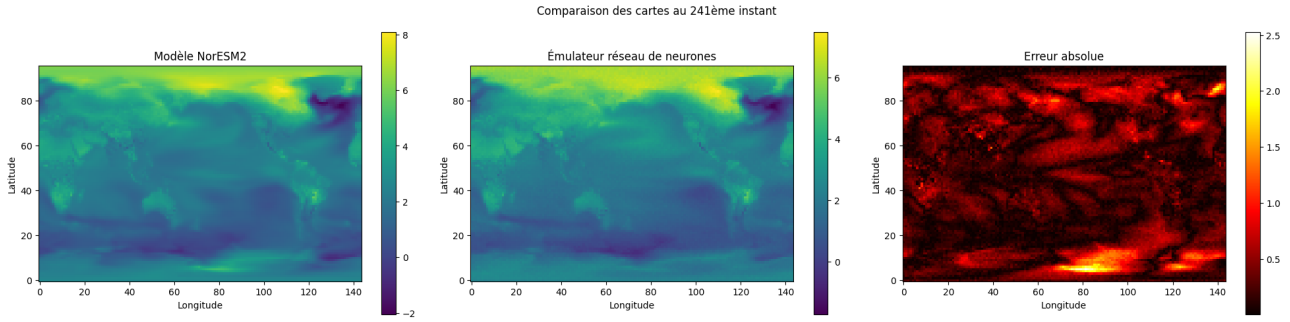

Figure 14: Carte des températures pour la dernière année de simulation(1)

2e modèle. Nombre de paramètres : 574 644

Ici, on reprend le modèle précédent mais avec la fonction de perte pondérée par les cosinus.

Métrique	Valeur moyenne	Écart-type
NRMSE_t	0.328	$\pm$ 0.0355

Table 4: Résultats du NRMSE total obtenus en ajoutant la fonction de perte pondérée au modèle précédent

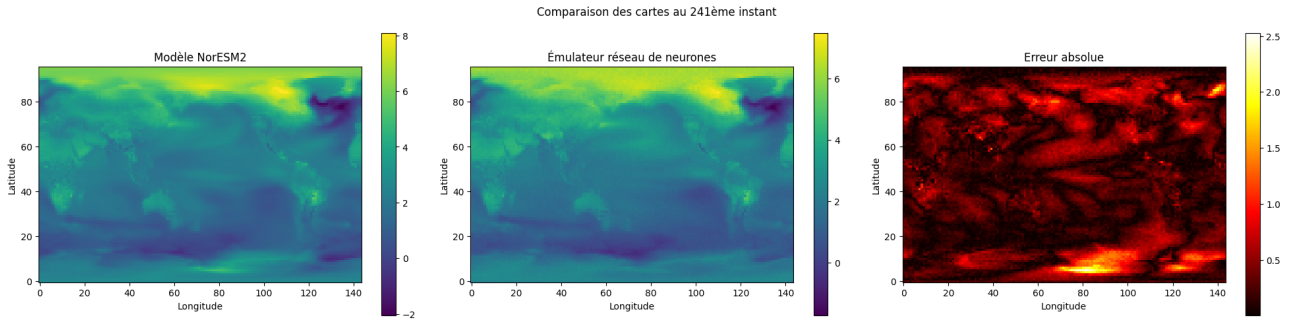

Figure 15: Carte des températures pour la dernière année de simulation(2)

3e modèle. Nombre de paramètres : 574 644
Ici, on reprend le modèle précédent en ajoutant une pénalisation Ridge. En testant des valeurs du facteur de pénalisation sur le modèle de base (voir figure 6), on a remarqué que les petites valeurs fonctionnaient le mieux mais qu'aucune tendance claire ne se dégageait entre 10^{-9} et 10^{-7}. Les résultats suivants ont été obtenu avec 10^{-7} mais comme on le verra ensuite, une optimisation des hyperparamètres moins grossière devrait permettre de cibler plus précisément la bonne valeur à choisir.

Métrique	Valeur moyenne	Écart-type
NRMSE_t	0.292	$\pm$ 0.0170

Table 5: Résultats du NRMSE total obtenus en ajoutant une pénalisation

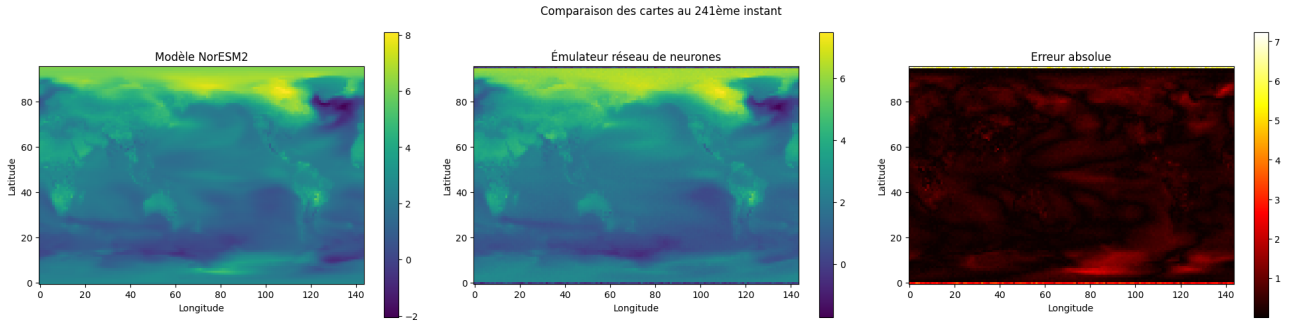

Figure 16: Carte des températures pour la dernière année de simulation(3)

Analyse des résultats.

On remarque qu'avec l'architecture proposée dans cette section, on obtient des résultats meilleurs que ceux de l'article (d'après la figure 10, 0.327 pour le $NRMSE_{total}$ pour la température) mais surtout plus consistants. En effet, les résultats sont moyennés sur 10 entraînements et la variance est explicitée (et elle est très faible à chaque fois). L'article quant à lui est plus vague et a pu potentiellement proposer un chiffre légèrement enjolivé.

Pour ce qui est des ajouts apportés à l'architecture de base (1er modèle), on voit que l'apport le plus intéressant est celui de la pénalisation Ridge, qui a permis de diviser la variance par deux tout en diminuant la valeur moyenne du $NRMSE_{total}$ de plus de 6%.

3.2.5 Convolution/Déconvolution

Dans le but d'explorer une reconstruction spatiale plus explicite des cartes climatiques, nous avons implémenté un modèle de type encodeur-décodeur, basé sur des couches convolutives (`Conv2D`) pour l'encodage spatial et des convolutions transposées (`ConvTranspose2D`) pour la reconstruction finale. Cette approche permet de capter les motifs locaux présents dans les champs climatiques, en réduisant fortement la dimensionnalité avant de reconstruire l'image finale.
Concrètement, chaque pas de temps de l'entrée est compressé par trois blocs convolutifs suivis de *average pooling*, jusqu'à obtenir un tenseur de taille $(N, T, 20)$, transmis ensuite à un LSTM chargé de modéliser les dépendances temporelles. Le vecteur issu du dernier pas de temps est alors utilisé comme vecteur latent pour le décodeur, composé de couches de convolution transposée successives permettant de générer une image complète de température annuelle.
Cette structure permet une réduction drastique du nombre de paramètres du modèle. Alors que le modèle de base basé sur CNN+LSTM comptait environ **574 644 paramètres**, le modèle convolution-déconvolution présenté ici n'en utilise que **19 203**.
Cependant, cette approche présente des limites en termes de performance prédictive. Les résultats obtenus sont significativement moins bons que ceux des autres modèles testés, comme le montre la valeur du NRMSE total :

Modèle convolution-déconvolution. Nombre de paramètres : 19 203

Métrique	Valeur moyenne	Écart-type
NRMSE_t	0.581	$\pm$ 0.191

Analyse des résultats

Plusieurs éléments peuvent expliquer cette contre-performance :

- Le nombre très réduit de paramètres dans la partie décodeur ne permet pas au modèle de reconstruire correctement les cartes de température à partir du vecteur latent.
- La déconvolution utilisée pour la reconstruction est moins expressive que les couches linéaires employées dans d'autres modèles, ce qui limite la qualité des sorties générées.

En résumé, bien que le modèle soit beaucoup plus léger en termes de nombre de paramètres, cette simplification se fait au détriment de la précision des prédictions.
Dans le prolongement de l'architecture convolution-déconvolution, nous avons implémenté un modèle hybride plus simple, dans lequel la phase de reconstruction spatiale ne repose pas sur des déconvolutions, mais sur une couche entièrement connectée (`Linear`) appliquée à la sortie du LSTM.
Comme précédemment, l'entrée est traitée temporellement par un CNN, avec trois blocs convolutifs et des étapes de *average pooling*, permettant de compresser chaque pas de temps en un vecteur de dimension réduite. Ces vecteurs sont ensuite transmis à un LSTM, dont la sortie au dernier pas de temps est un vecteur latent global. Contrairement au modèle précédent, ce vecteur est directement projeté via une couche linéaire dans un vecteur de dimension 96×144, qui est ensuite reformaté pour former la carte finale.
Ce modèle conserve l'avantage de la compression spatiale du CNN en entrée, tout en évitant les difficultés associées à la déconvolution.
Les performances obtenues sont nettement meilleures que celles du modèle convolution-déconvolution, comme le montre le tableau ci-dessous :

Modèle CNN–LSTM avec couche fully connected. Nombre de paramètres : 579,550

Métrique	Valeur moyenne	Écart-type
NRMSE_t	0.443	$\pm$ 0.171

Analyse des résultats

Ce modèle montre qu'il n'est pas nécessaire de complexifier l'encodeur pour obtenir de bonnes performances. Extraire une vingtaine de caractéristiques à l'aide d'une unique couche convolutive, suivie d'un *global average pooling*, suffit pour capturer l'information spatiale pertinente des champs climatiques.
Superposer plusieurs couches convolutives ne permet pas d'améliorer les résultats, et rend l'entraînement plus instable.

3.2.6 Optimisation des hyperparamètres

L'optimisation des hyperparamètres du réseau de neurones (nombre de filtres, taille des couches, régularisation, learning rate, etc.) constitue une piste majeure d'amélioration. Actuellement, ces paramètres sont ajustés empiriquement et pourraient être mieux calibrés pour réduire le NRMSE et stabiliser l'entraînement.

En particulier, nous avons mis en place un *slider temporel*, c'est-à-dire une fenêtre glissante permettant d'extraire des séquences de longueur fixe à partir des données climatiques. Cette structure permet d'exploiter la dimension temporelle via des modèles séquentiels comme les LSTM. Bien que cette taille de fenêtre ait été fixée manuellement dans notre cas, elle pourrait tout à fait être considérée comme un hyperparamètre à optimiser, au même titre que la profondeur ou le nombre de filtres.

Pour aller plus loin, nous pourrions essayer d'utiliser des méthodes d'optimisation automatique, comme la recherche aléatoire ou l'optimisation bayésienne (Optuna), afin d'explorer plus systématiquement l'espace des hyperparamètres (nombre de filtres, taille du noyau, nombre d'unités LSTM/GRU, facteur de pénalisation Ridge, etc.). Cette approche permettrait de sélectionner les meilleures configurations de manière rigoureuse et reproductible.

3.2.7 Pré-entraînement des poids

Une piste d'amélioration non explorée dans ce projet mais particulièrement prometteuse consiste à utiliser un encodeur pré-entraîné sur des données climatiques. L'idée serait de tirer parti d'un modèle ayant déjà appris des représentations utiles sur des tâches similaires, afin d'améliorer l'extraction des caractéristiques spatiales ou spatio-temporelles dès les premières étapes du réseau.
Un exemple concret est le modèle **ClimaX** [1], récemment proposé pour le traitement de données climatiques multi-échelles. Ce modèle repose sur des mécanismes de type Transformer et a été pré-entraîné sur des jeux de données climatiques variés, permettant d'apprendre des représentations générales utiles pour des tâches de prédiction, de reconstruction ou de classification dans le domaine climatique.
Dans notre cas, nous avons envisagé d'utiliser le backbone de ClimaX comme encodeur dans notre propre architecture, afin de bénéficier de cette base de connaissances pré-apprise. Toutefois, en raison de difficultés techniques liées à l'installation du modèle et à son intégration dans notre framework PyTorch existant (notamment des problèmes de dépendances et de compatibilité avec notre environnement), nous n'avons pas pu mettre en œuvre cette stratégie dans le temps imparti.
Nous pensons néanmoins que cette approche reste très pertinente et pourrait apporter un gain significatif en performance en facilitant l'apprentissage des représentations climatiques pertinentes dès les premières couches du réseau.

3.3 Améliorations pour les processus gaussiens

3.3.1 Méthode de comparaison

Avant de se pencher sur les tentatives d'amélioration, il nous faut établir une méthode de comparaison rigoureuse pour pouvoir déterminer l'efficacité d'une piste. Notre métrique d'évaluation sera la même que celle proposée dans l'article, à savoir la NRMSE.

En reprenant le code initial, nous nous sommes aperçus que les NRMSE issues de nos propres simulations étaient toujours légèrement supérieures à celles proposées par l'article. Nous avons donc décidé de prendre pour référence les valeurs que nous obtenons à l'issue du programme initial sur nos machines.
Le degré de précision d'un apprentissage est évidemment relié au nombre d'itérations lors de l'optimisation, déjà évoqué dans la partie 2.5.2. Il nous a donc paru pertinent de s'intéresser à l'évolution des NMRSE en fonction du nombre d'itérations. Naturellement, ces dernières convergent, comme le montre l'exemple du graphe 17 tracé pour la variable PR.

Ce sont ces valeurs "convergées" que nous utiliserons pour décider de l'efficacité des méthodes que nous avons mises en place. Voici un tableau résumant les valeurs des NRMSE globale et spatiale pour les 4 variables de sortie :

	PR	PR90	DTR	TAS
NRMSE s	3.471	3.813	12.99	0.118
NRMSE g	0.388	0.443	2.712	0.037

Table 6: Un tableau avec 3 lignes et 5 colonnes

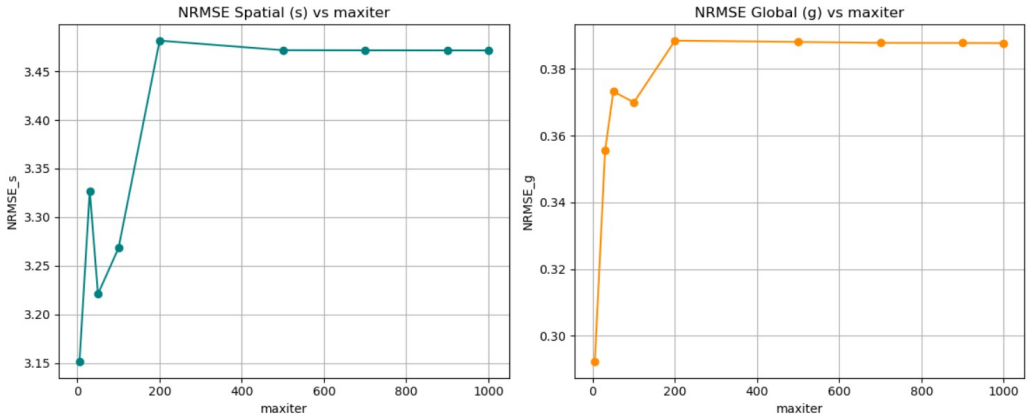

Figure 17: Convergence des NRMSE avec nb iter

3.3.2 ACP sur les sorties

Rappel de la méthode et implémentation
L'une des contributions principales de ce travail réside dans l'utilisation d'une réduction de dimension par Analyse en Composantes Principales (ACP) appliquée aux champs de sortie du modèle. Comme évoqué dans la partie 3.1.2, chaque champ annuel est d'abord projeté dans une base de composantes principales préalablement calculées sur les données d'entraînement. Le modèle est entraîné à prédire ces coefficients principaux, ce qui réduit fortement la dimension de l'espace de sortie et atténue le bruit inhérent aux données brutes.

L'implémentation est assez directe, et est présentée en figure 18.

```
# PCA sur train uniquement
n_eofs=170 # explique >90% de la variance pour pr
pca = PCA(n_components=n_eofs)
PC_train = pca.fit_transform(Y_train_centered)
PC_test = pca.transform(Y_test_centered)
```

Figure 18: Implémentation PCA Sortie

Visualisation de l'ACP
A titre illustratif, nous pouvons afficher les modes et les séries temporelles qui y sont associées. On retrouve des répartitions spatiales qui décrivent le comportement réel des variables à prédire. Les figures 19 et 20 présentent les deux premières EOFs pour la variable **DTR** ainsi que l'évolution de leur coefficient au cours des différentes données d'entraînement.

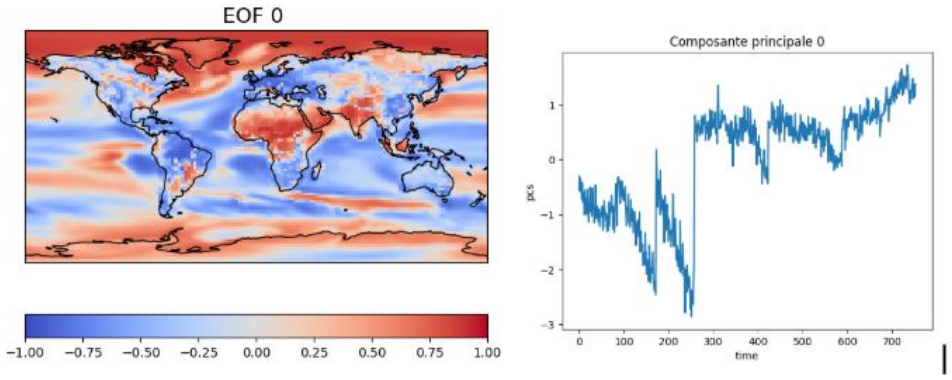

Figure 19: EOF 0 et l'évolution de la composante principale pour le DTR

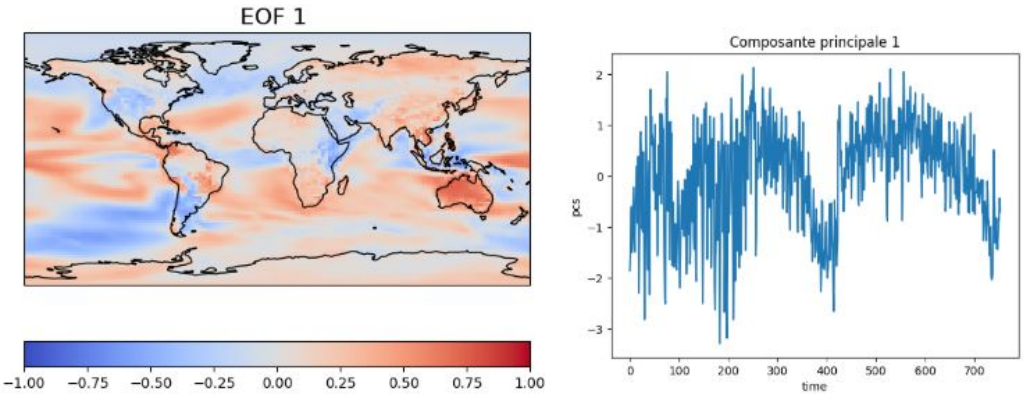

Figure 20: EOF 1 et l'évolution de la composante principale pour le DTR

On observe distinctement 5 sauts importants sur la première EOF, qui correspondent aux différentes simulations utilisés pour l'entraînement. Le grand saut au temps 258 dans la figure 19 marque la séparation entre les fichiers représentants les données des fichiers ssp et les fichiers historiques (`"historical", "hist-GHG", "hist-aer"`). Ce saut témoigne d'un grand changement dans le comportement global de la DTR au cours du temps, puisque ces deux types de fichier correspondent à des périodes historiques différentes.

Choix du nombre d'EOFs pour la prédiction.
Le choix du nombre de composantes principales (EOFs) utilisées pour les sorties n'est pas arbitraire. Lors d'une analyse en composantes principales (ACP), le nombre de composantes conservées doit permettre d'expliquer un pourcentage suffisamment élevé de la variance des variables originales, que nous avons choisi à 90%. Pour déterminer le nombre d'EOFs associées, on trace la variance cumulée expliquée en fonction du nombre de composantes retenues, comme illustré en figure 21.

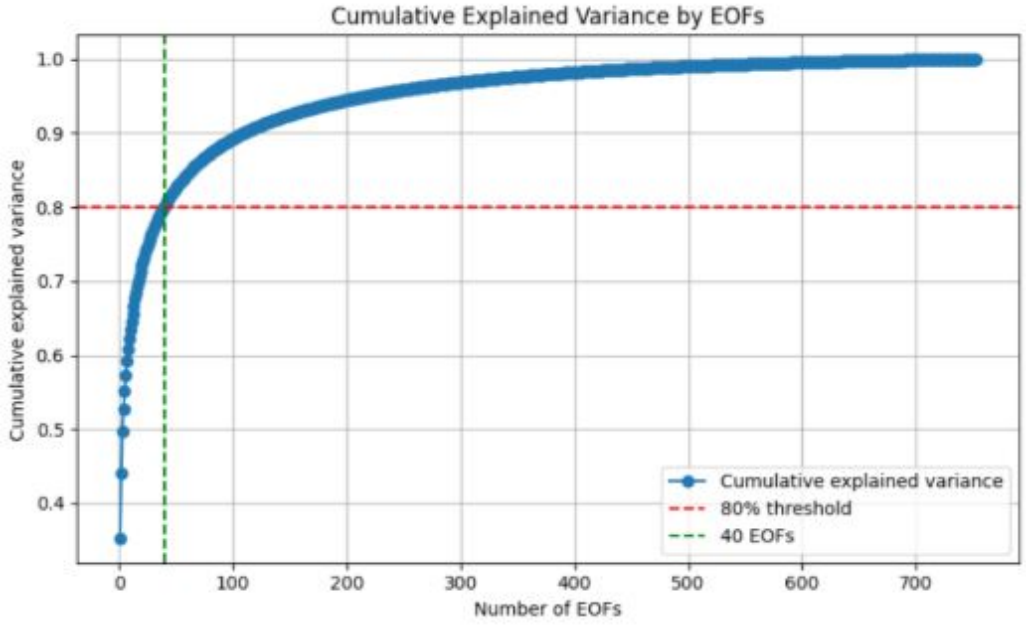

Figure 21: Fonction de répartition de la variance expliquée par les composantes principales

Cette opération a été répétée pour chacune des variables à prédire, ce qui a permis d'identifier le nombre de composantes nécessaires pour atteindre différents seuils de variance expliquée. Le tableau suivant présente ces valeurs :

	TAS	**PR**	**PR90**	**DTR**
80%	2	28	60	40
90%	2	90	112	110
95%	12	182	286	217

Table 7: % de la variance et nombre de composantes associé

On observe que pour expliquer un pourcentage donné de variance, la température (tas) nécessite beaucoup moins de composantes principales que les autres variables. Cela s'explique par une dynamique plus simple de la température au fil des années, avec moins de phénomènes locaux, ce qui réduit le besoin en modes pour la représenter fidèlement.

Afin de comparer cette méthode avec les résultats du papier recalculés en section 3.3.1, nous avons représenté l'évolution de la NRMSE en fonction du nombre de composantes EOF utilisées, pour chaque variable. Cela permet de déterminer à partir de quel nombre de composantes les performances convergent. La figure 22 illustre ce comportement pour la variable **PR**.
Les valeurs absolues de NRMSE dans cette figure ne sont pas fiables, car elles résultent d'un traitement des données d'entrée moins performant. Toutefois, la tendance générale est représentative et suffisante pour notre analyse, compte tenu du temps de calcul nécessaire pour obtenir ce graphique (plusieurs heures).
À partir de 100 composantes principales, la NRMSE spatiale de la variable **PR** est stable, c'est donc ce nombre de composantes que nous utiliserons pour calculer la NRMSE finale permettant de comparer nos résultats à ceux de l'article de départ.

Nouvel entraînement des GP pour la prédiction des composantes principales des sorties
Le schéma présente en figure 23 résume les étapes de traitement avant l'entraînement : une ACP est d'abord réalisée sur les données d'entrée et de sortie, puis le modèle est entraîné sur les composantes principales des sorties d'entraînement.

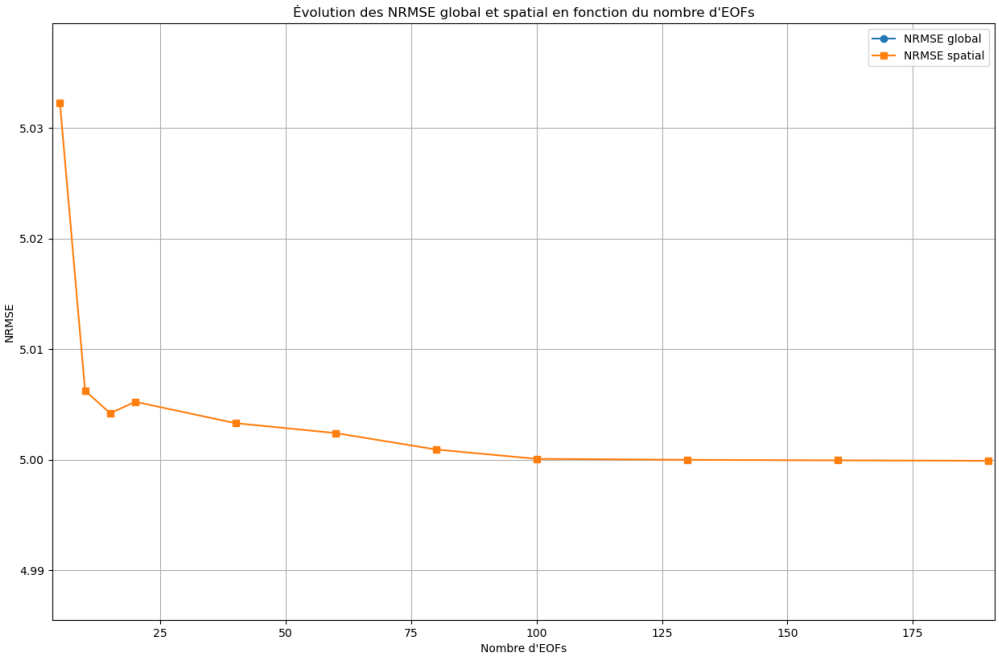

Figure 22: Évolution de la NRMSE en fonction du nombre de composantes principales pour la variable PR

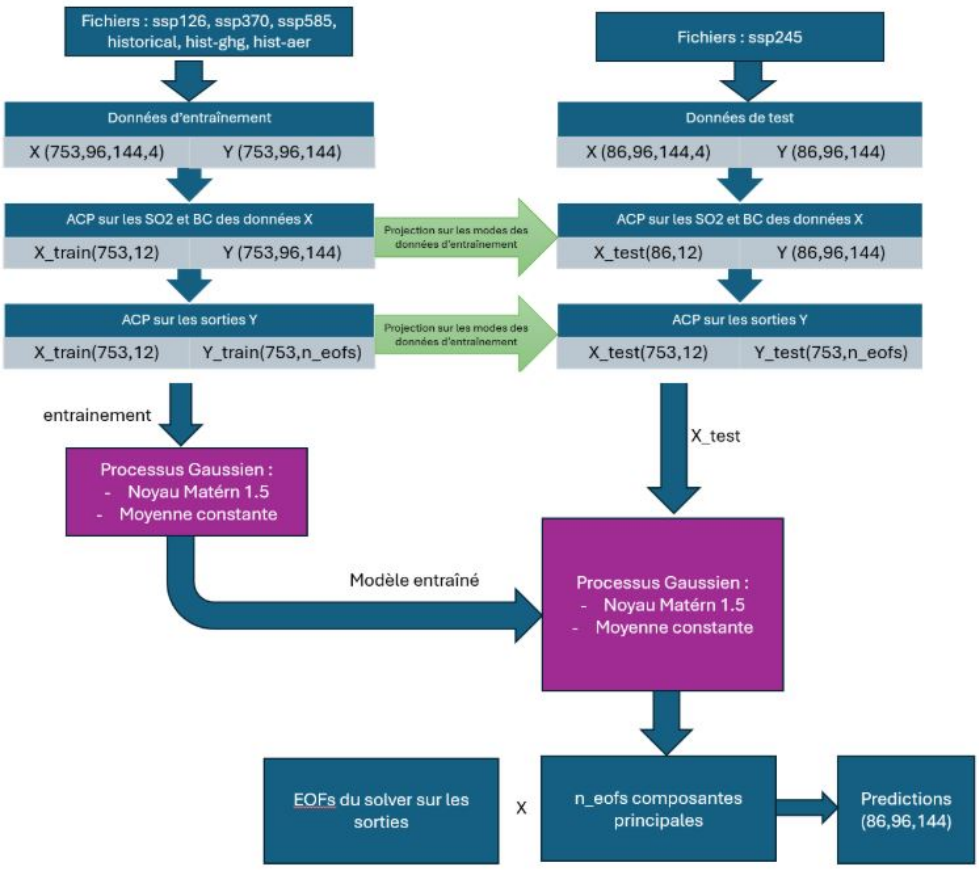

Figure 23: Étapes de traitement des données et entraînement

Performances du modèle

Commencons par visualiser graphiquement les résultats de l'apprentissage, en les comparant aux données réelles,

à l'aide des figures 24, 25, 26 et 27.

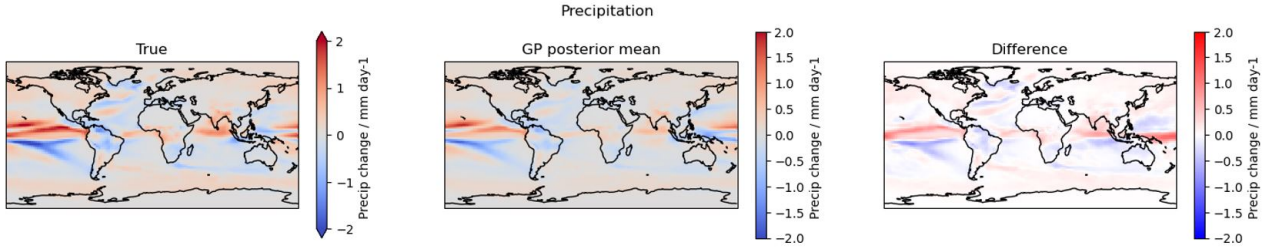

Figure 24: Comparaison entre prédiction et réel — PR

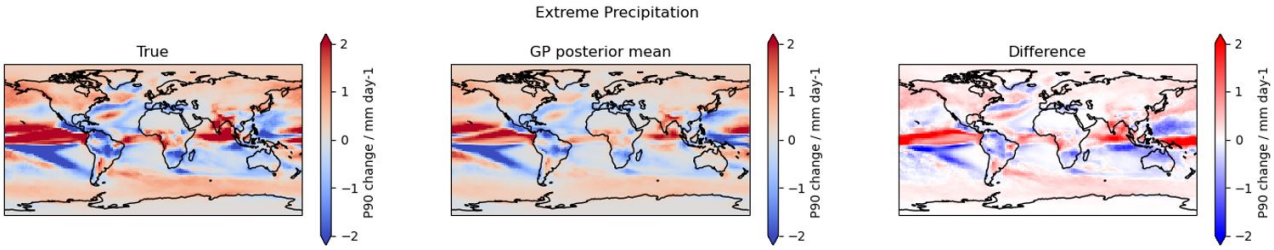

Figure 25: Comparaison entre prédiction et réel — PR90

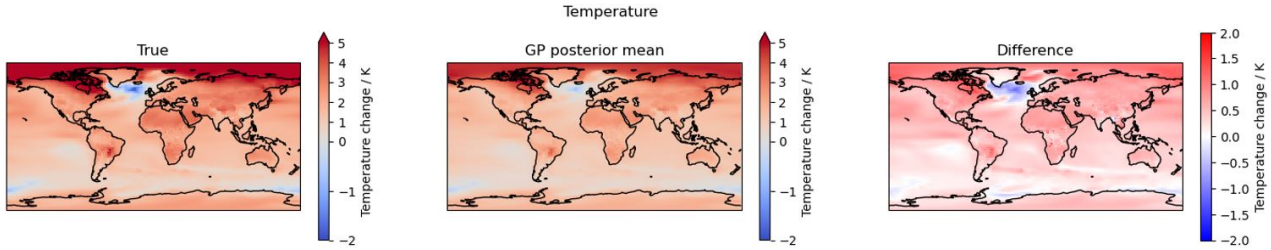

Figure 26: Comparaison entre prédiction et réel — TAS

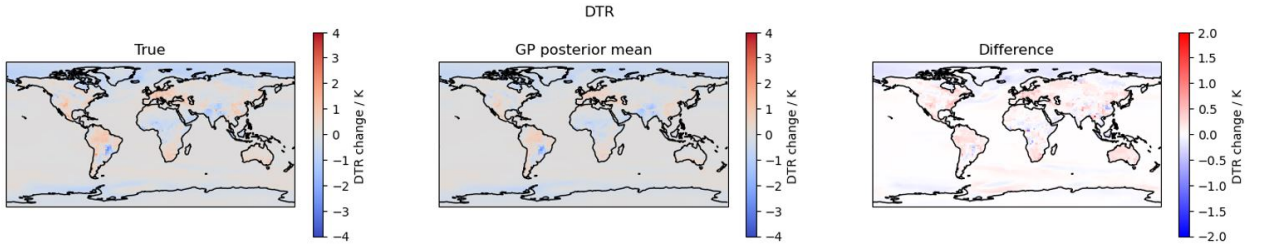

Figure 27: Comparaison entre prédiction et réel — DTR

Ces figures montrent que les reconstructions à partir des ACP permettent de bien représenter les tendances globales d'évolution.

Décrivons maintenant la qualité de l'apprentissage de manière plus quantitative grâce aux métriques de l'article. Le tableau 8 compare nos résultats à ceux obtenus dans ClimateBench.

	PR		**PR90**		**TAS**		**DTR**	
	Spatial	Global	Spatial	Global	Spatial	Global	Spatial	Global
GPytorch	4.9330	0.6521	5.9112	0.6866	0.3769	0.2609	13.5028	2.5771
ClimateBench	3.471	0.388	3.813	0.443	0.118	0.037	12.99	2.712
Erreur relative	42.0%	68.0%	55.0%	55.0%	219%	605%	3.95%	-4.98%
Qualité de l'apprentissage	Correct	Correct	Correct	Correct	Mauvais	Très mauvais	Très bon	Meilleur

Table 8: NRMSE spatiale et globale pour les différentes variables

Analyse des résultats

Pour décrire la qualité de l'apprentissage, nous avons décidé de nous pencher sur l'erreur relative de notre apprentissage comparée aux résultats de l'article. Le critère arbitraire que nous nous sommes fixés est : $\varepsilon \geq 500\%$, très mauvais apprentissage; $\varepsilon \geq 200\%$, mauvais appretissage; $\varepsilon \leq 100\%$, apprentissage correct; $\varepsilon \leq 10\%$, très bon apprentissage, et évidemment si $\varepsilon \leq 0\%$, l'apprentissage est meilleur que celui du papier.

Les NRMSE sont globalement moins bonnes avec la prédiction des composantes principales, mais restent pour la plupart des prédictions relativement satisfaisantes.

On observe une variabilité des performances selon les variables : pour la DTR, la NRMSE spatiale est comparable à celle de ClimateBench et la globale est même meilleure. À l'inverse, les performances sont nettement moins bonnes pour **TAS**. On pourra se satisfaire de l'apprentissage pour les variables **PR** et **PR90**, dont l'ordre des grandeurs des NRMSE est comparable aux résultats de l'article.

Un premier soupçon pour expliquer la qualité de l'apprentissage a été que cela provenait d'un nombre insuffisant de composantes principales, qui ne permettaient pas de capturer avec précision les phénomènes locaux. Cependant, la figure 22 montre que l'erreur (NRMSE) atteint une asymptote au-delà d'un certain nombre de composantes, ce qui suggère une limite intrinsèque de la méthode.

Cela peut s'expliquer par le fait qu'une erreur sur l'un des coefficients associés aux premières composantes principales, qui capturent les structures spatiales dominantes, peut se propager à l'ensemble du champ reconstruit. Ainsi, une légère erreur dans l'espace réduit des composantes peut induire une erreur diffuse mais globalement répartie sur toute la carte, affectant significativement les métriques locales comme la NRMSE spatiale.

4 Conclusion et Perspectives

4.1 Conclusion

Ce projet d'émulation climatique, mené sur la base du benchmark ClimateBench et des données issues de NorESM2, visait à développer deux approches complémentaires : réseaux de neurones profonds et processus gaussiens, pour reproduire à moindre coût de calcul l'évolution de grandes variables climatiques spatialisées. D'une part, le modèle CNN–GRU régularisé offre aujourd'hui une prédiction rapide de la température de surface, avec une NRMSE_t autour de 0,292 sur le scénario de test `SSP245`. Cette performance se situe au même ordre de grandeur, voire légèrement meilleure, que les résultats publiés initialement, et résulte d'une implémentation PyTorch soignée (moyenne sur plusieurs entraînements), d'une perte pondérée géographiquement et d'une régularisation L_2 modulée. Les réseaux de neurones ont démontré leur aptitude à capturer la structure spatio-temporelle des champs climatiques, tout en restant relativement modulables (remplacement du LSTM par un GRU, encodeur très léger ou décodeur via couche fully connected).

D'autre part, l'approche bayésienne par processus gaussiens (implémentée sous GPyTorch) s'appuie sur une réduction de dimension via ACP (EOF) appliquée aux sorties, suivie d'une série de régressions scalaires indépendantes sur chaque score EOF. Bien que cette méthode sépare la covariance spatiale (via l'ACP) de la covariance temporelle/paramétrique (via le noyau Matern), elle présente un bilan contrasté selon les variables :

- pour DTR (plage diurne des températures), la NRMSE spatiale (13,50) et globale (2,58) rivalise ou surpasse légèrement ClimateBench, témoignant de la capacité des premiers modes EOF à capturer l'essentiel de la variabilité locale ;
- en revanche, pour des variables plus complexes comme PR (précipitations totales) et PR90 (précipitations extrêmes), l'erreur spatiale est 40 % à 55 % plus élevée qu'avec l'implémentation de référence, et la NRMSE globale s'en trouve aussi fortement dégradée. Cela s'explique en partie par la difficulté à reproduire précisément les phénomènes locaux (houle topographique, orographie, extrêmes), qui exigent souvent un nombre très élevé d'EOF (100 pour atteindre la stabilité de la NRMSE spatiale).

Dans l'ensemble, l'émulation par GP + ACP présente un avantage théorique non négligeable : la quantification intrinsèque de l'incertitude sur chaque score EOF, et donc sur chaque point de la carte reconstituée. Toutefois, elle reste lourdement limitée par la complexité cubique en temps de calcul de l'apprentissage GP et par la représentation compressée (linéaire) des champs climatiques. En l'absence d'extensions (noyaux profonds ou « deep kernels », autoencodeurs non linéaires pour la réduction de dimension), les GP peinent à concurrencer les réseaux de neurones pour la plupart des variables, surtout celles soumises à une forte variabilité spatiale.

Sur le plan général, ce projet a permis :

1. de maîtriser la chaîne complète d'émulation climatique (préparation des données, définition de métriques géophysiquement cohérentes, entraînement et évaluation sur un scénario de test standard) ;
2. de mettre en place et d'améliorer une architecture CNN–GRU sous PyTorch, avec comparaison systématique aux résultats de ClimateBench en TensorFlow ;
3. d'explorer en profondeur la piste des processus gaussiens, depuis l'implémentation « pixel par pixel » jusqu'au découplage en scores EOF via GPyTorch, en analysant finement la dépendance des performances au nombre de modes EOF utilisés.

4.2 Perspectives

Les perspectives restent nombreuses et prometteuses :

- pour les réseaux de neurones, l'intégration d'un encodeur pré-entraîné multi-échelle (ClimaX ou Transformers) et la mise en place d'une optimisation automatique des hyperparamètres (recherche bayésienne) semblent être des pistes majeures pour abaisser encore la NRMSE, en particulier sur les variables difficiles (précipitations, extrêmes) ;
- pour les processus gaussiens, l'adoption de « deep kernels » (noyaux paramétrés par un réseau neuronal), l'utilisation d'autoencodeurs non linéaires (variational autoencoders) pour compresser plus fidèlement les champs spatiaux, ou la combinaison GP+NN (par exemple, prédiction des scores EOF puis raffinement en sortie via un petit réseau) pourraient combler l'écart de performance tout en conservant une estimation explicite de l'incertitude.

En somme, ce projet a confirmé que l'émulation climatique fondée sur l'apprentissage automatique constitue aujourd'hui un levier essentiel pour produire rapidement des projections de variables climatiques fines, tout en offrant des marges de progression considérables grâce aux avancées récentes en architectures profondes et en modélisation bayésienne. Les enseignements tirés – tant sur les choix architecturaux que sur la gestion pratique des données e-science à grande échelle – préparent le terrain pour des travaux futurs visant à intégrer des retours d'incertitude, à généraliser l'émulateur à d'autres modèles ESM et à coupler ces approches avec des modèles socio-économiques pour des projections intégrées du futur climat mondial.

References

[1] Huyen Nguyen, Zihang Chen, Liyuan Li, Xingyu Zhang, Jitendra Malik, Eric Xing, Stephan Rasp, Saining Xie, and Nikhil Goyal. Climax: A foundation model for weather and climate. *arXiv preprint arXiv:2301.10343*, 2023.

[2] D. Watson-Parris and contributors. Climatebench. https://github.com/duncanwp/ClimateBench, 2021.

[3] Rao Y. Olivié D. Seland Ø. Nowack P. Camps-Valls G. et al. Watson-Parris, D. Climatebench v1.0 : A benchmark for data-driven climate projections. *JAMES : Journal of Advances in Modeling Earth Systems*, 14, 2021.