Projet de recherche :

Amélioration de la prédiction de la production énergétique photovoltaïque des bâtiments par ConvLSTM optimisé par métaheuristiques

Auteurs :

Mabrouk Chayma

Hocine Yacine

Escande Paul-Émile

Duflaut Thomas

Sommaire

1. **Résumé**
2. **Abstract**
3. **Introduction**
4. **Description du sujet**
5. **Question de recherche**
6. **État de l'art**
   - 6.1. Modèle ConvLSTM
   - 6.2. Hyperparamètres du modèle
   - 6.3. Métaheuristiques
     - 6.3.1. PSO (Particle Swarm Optimization)
     - 6.3.2. Optuna
     - 6.3.3. Algorithme Génétique (AG)
7. **Méthodologie de recherche**
   - 7.1. Nettoyage et transformation des données
   - 7.2. Construction du modèle ConvLSTM
   - 7.3. Optimisation via les métaheuristiques
8. **Analyse des résultats**
   - 8.1. Performances de prédiction
   - 8.2. Comparaison des approches d'optimisation
9. **Conclusion**
10. **Bibliographie**

Résumé

Ce rapport présente un projet de recherche visant à améliorer la précision de la prédiction de la production d'énergie de systèmes photovoltaïques (PV) intégrés aux bâtiments en utilisant un modèle **ConvLSTM** (Convolutional Long Short-Term Memory) dont les **hyperparamètres** sont optimisés par des méthodes **métaheuristiques**. Après une introduction au contexte de la prévision de l'énergie solaire et aux défis associés, nous décrivons la problématique et la question de recherche. Un état de l'art détaillé couvre le fonctionnement du modèle ConvLSTM, l'importance du choix des hyperparamètres dans les réseaux de neurones, et les techniques métaheuristiques explorées – en particulier **PSO** (Particle Swarm Optimization), **Optuna** (optimisation bayésienne des hyperparamètres) et les **Algorithmes Génétiques**.

La méthodologie de recherche est ensuite exposée : elle comprend la préparation et le prétraitement du jeu de données initial (*full_dataset*), aboutissant à quatre sous-ensembles nommés *day_month*, *month_year*, *scaled_dataset* et *year_week*. Ces données alimentent la construction d'un modèle ConvLSTM dont nous détaillons l'architecture générale. Nous expliquons comment chaque approche métaheuristique a été appliquée pour ajuster automatiquement des hyperparamètres clés (par ex. le taux d'apprentissage, le nombre de neurones, etc.), au lieu de s'en remettre à un réglage manuel.

Les résultats expérimentaux sont analysés en profondeur. Nous présentons des exemples concrets de prédictions réalisées par le modèle, comparant le comportement d'un ConvLSTM "baseline" (hyperparamètres par défaut) à celui optimisé par chaque méthode. Des **graphes de performance** (courbes de prédiction, barres comparatives de métriques) et des **tableaux de résultats** (erreur MAE, R carrée, temps de calcul) illustrent l'amélioration obtenue. Globalement, l'optimisation des hyperparamètres via métaheuristiques a permis d'accroître significativement la précision des prédictions par rapport au modèle de base. Enfin, la conclusion récapitule les enseignements clés du projet, discute les avantages et limites de chaque approche d'optimisation, et propose des perspectives futures (comme l'extension à d'autres bâtiments ou l'intégration d'autres variables météorologiques).

Abstract

Ce rapport de recherche décrit un projet visant à améliorer la précision de la prévision de la production d'énergie photovoltaïque pour les systèmes solaires intégrés aux bâtiments, en utilisant un modèle ConvLSTM optimisé par des techniques métaheuristiques de réglage des hyperparamètres. Nous présentons l'importance et les défis liés à la prévision de la production solaire photovoltaïque, ainsi que la question de recherche qui a guidé ce travail. Une revue de la littérature approfondie couvre l'architecture du modèle ConvLSTM, le rôle des hyperparamètres dans ses performances, ainsi que les approches métaheuristiques utilisées – en particulier PSO (Particle Swarm Optimization), Optuna (une méthode d'optimisation bayésienne des hyperparamètres) et les algorithmes génétiques.

Nous détaillons ensuite notre méthodologie de recherche, comprenant les étapes de nettoyage et de transformation des données appliquées au jeu de données initial (*full_dataset*), lequel a été prétraité pour donner naissance à quatre sous-ensembles : *day_month*, *month_year*, *scaled_dataset* et *year_week*. Nous décrivons la construction du modèle ConvLSTM et expliquons comment chaque technique métaheuristique a été utilisée pour ajuster automatiquement les hyperparamètres clés du modèle (par exemple, le taux d'apprentissage, le nombre de neurones), au lieu de procéder à un réglage manuel.

La section des résultats fournit une analyse approfondie des performances du modèle. Nous y incluons des exemples concrets avec des graphiques et des tableaux comparant la précision prédictive du modèle ConvLSTM avant et après optimisation. Ces visualisations (courbes de prédiction, graphiques comparatifs des scores R carrée MAE, et temps d'entraînement) démontrent clairement que l'optimisation métaheuristique des hyperparamètres a considérablement amélioré la précision des prévisions par rapport au modèle de base. En conclusion, nous récapitulons les résultats du projet, en mettant en lumière les avantages et les limites de chaque méthode d'optimisation, et en suggérant des perspectives futures telles que l'application du modèle optimisé à d'autres bâtiments ou l'intégration de nouvelles variables météorologiques.

Introduction

Les systèmes photovoltaïques (PV) intégrés aux bâtiments jouent un rôle croissant dans la production d'énergie renouvelable. Pouvoir **prédire avec précision la production d'électricité photovoltaïque** d'un bâtiment est crucial pour optimiser la gestion de l'énergie (stockage, revente au réseau, dimensionnement des batteries) et pour améliorer l'intégration des énergies renouvelables dans le réseau électrique. Cependant, cette prédiction est complexe en raison de la variabilité inhérente de la ressource solaire et des conditions environnementales. En effet, l'efficacité d'un panneau solaire dépend de nombreux facteurs tels que l'ensoleillement, la température ambiante, l'humidité relative, etc., ce qui rend difficiles les estimations de production **a priori**

Au cours des dernières années, l'essor des méthodes d'**apprentissage profond (deep learning)** a ouvert de nouvelles possibilités pour la prévision de séries temporelles complexes comme la production solaire. Des réseaux de neurones avancés tels que les LSTM (Long Short-Term Memory) et les CNN (réseaux de neurones convolutifs) ont montré des résultats prometteurs pour capturer les tendances non linéaires des données énergétiques. Plus récemment, un modèle hybride appelé **ConvLSTM (Convolutional LSTM)** a été proposé pour combiner les avantages des convolutions (extraction de caractéristiques locales) et des LSTM (modélisation de dépendances temporelles) au sein d'un même réseau. Des études ont démontré que les modèles ConvLSTM peuvent surpasser les modèles LSTM ou CNN seuls dans des tâches de prévision séquentielle complexes .

Néanmoins, la performance d'un modèle de réseau de neurones dépend fortement de ses **hyperparamètres**. Le choix adéquat du taux d'apprentissage, du nombre de neurones dans les couches, de la taille des filtres de convolution, du nombre d'époques d'entraînement, peut faire la différence entre un modèle médiocre et un modèle hautement performant. Trouver manuellement la bonne combinaison est fastidieux et repose souvent sur des essais-erreurs empiriques. C'est pourquoi l'automatisation de l'optimisation des hyperparamètres suscite un intérêt croissant. En particulier, des méthodes **métaheuristiques** stochastiques – telles que les algorithmes génétiques ou les algorithmes de type essaim de particules – ont été explorées pour rechercher plus efficacement les hyperparamètres optimaux dans l'espace des configurations possibles.

Ce projet de recherche se positionne au croisement de ces axes : il vise à **améliorer la précision de prédiction de la production PV de bâtiments** en exploitant un modèle

ConvLSTM dont les hyperparamètres sont optimisés automatiquement via des approches métaheuristiques (notamment **PSO**, **Optuna** et **algorithme génétique**). Dans ce rapport, nous décrivons en détail la méthodologie employée, qui englobe la préparation des données, la conception du modèle et l'implémentation de l'optimisation, puis nous présentons et discutons les résultats obtenus. Le rapport est structuré de la manière suivante : après avoir décrit le sujet et formulé la question de recherche, nous passerons en revue l'état de l'art sur les ConvLSTM et l'optimisation par métaheuristiques. Ensuite, nous détaillerons notre méthodologie de recherche, avant d'analyser les performances du modèle optimisé comparées à un modèle de référence. Nous conclurons sur les apports du projet et les perspectives qui en découlent.

Description du sujet

La problématique traitée dans ce projet est la **prédiction à court terme de la production électrique de panneaux solaires photovoltaïques installés sur des bâtiments**, en particulier des bâtiments équipés de façades ou toitures PV. Le sujet s'inscrit dans un contexte de **gestion intelligente de l'énergie** : mieux prévoir l'énergie solaire disponible permet d'améliorer l'autoconsommation, de planifier l'utilisation de batteries ou d'anticiper les échanges avec le réseau public.

Pour relever ce défi, nous utilisons un **modèle d'apprentissage profond de type ConvLSTM**, adapté aux données spatio-temporelles, pour modéliser la série temporelle de production PV en intégrant d'éventuelles structures saisonnières ou temporelles complexes. L'originalité du projet réside dans l'**optimisation automatique des hyperparamètres** de ce modèle via des **métaheuristiques**. Plutôt que de fixer arbitrairement ou manuellement les paramètres du modèle (par exemple, le nombre de filtres convolutionnels, le nombre de neurones LSTM, le taux d'apprentissage…), nous employons plusieurs techniques d'optimisation stochastique inspirées soit de processus naturels (algorithme génétique, essaim de particules) soit de l'optimisation bayésienne (Optuna).

En résumé, le sujet se focalise sur la **conception d'un modèle prédictif ConvLSTM optimisé** pour fournir les meilleures estimations possibles de la production PV journalière ou horaire d'un bâtiment, à partir de données historiques (production passée, conditions météorologiques). L'ambition est d'améliorer l'état de l'art de la prévision solaire en combinant un modèle de deep learning performant avec des techniques modernes d'auto-optimisation des hyperparamètres.

Question de recherche

Comment l'utilisation de méthodes métaheuristiques pour l'optimisation des hyperparamètres d'un modèle ConvLSTM peut-elle améliorer la précision de la prévision de la production énergétique de systèmes photovoltaïques intégrés aux bâtiments ?

Cette question de recherche se décline en sous-interrogations : Quelle méthode d'optimisation (PSO, Optuna, algorithme génétique) fournit les meilleurs hyperparamètres en termes de performance prédictive et de temps de calcul ? Dans quelle mesure le modèle ConvLSTM optimisé surpasse-t-il un modèle ConvLSTM non optimisé (baseline) ou d'autres modèles traditionnels de prévision ? Enfin, quels sont les compromis et défis associés à l'utilisation de métaheuristiques pour ce type de problème (stabilité des résultats, complexité, généricité des hyperparamètres trouvés) ?

État de l'art

Modèle ConvLSTM

Le **ConvLSTM** (Convolutional Long Short-Term Memory) est un type de réseau de neurones récurrents proposé pour traiter des données séquentielles possédant également une structure spatiale locale, comme des séries temporelles d'images ou des grilles spatio-temporelles. Il a été introduit initialement par Shi et al. (2015) pour des tâches de nowcasting de précipitations, où il a démontré une amélioration significative par rapport aux LSTM classiques sur la prévision de séquences d'images radar .

Un ConvLSTM intègre dans chaque cellule LSTM une opération de convolution qui remplace la multiplication matricielle classique, ce qui permet de capturer des motifs locaux dans les entrées séquentielles (par exemple des structures spatiales voisines ou des patterns cycliques temporels). En d'autres termes, le ConvLSTM **combine l'apprentissage de caractéristiques locales** (via les filtres convolutionnels appliqués sur les données d'entrée ou sur l'état caché) **et la modélisation de la dynamique temporelle** (via les portes d'entrée/sortie/oubli d'une cellule LSTM).

Dans le domaine de la prévision de la production photovoltaïque, l'application des ConvLSTM est encore relativement récente mais prometteuse. Des recherches récentes ont montré que les modèles hybrides CNN-LSTM ou ConvLSTM peuvent surpasser à la fois les réseaux purement convolutionnels et les réseaux purement LSTM pour prédire l'énergie PV, en particulier lorsqu'il s'agit de prendre en compte à la fois des tendances saisonnières et la variabilité journalière .Par exemple, Costa (2022) a évalué des réseaux LSTM, des CNN 1D et un ConvLSTM pour la prévision de la production de systèmes PV résidentiels à différents horizons temporels, et a trouvé que le ConvLSTM obtenait les meilleures performances en termes d'erreur (MAE, RMSE) dans la plupart des. Une explication est que le ConvLSTM parvient à mieux **généraliser** en exploitant des structures répétitives dans les données (cycles jour-nuit, motifs saisonniers, etc.) tout en s'ajustant aux changements temporels.

Un modèle ConvLSTM typique pour la prédiction de séries temporelles peut être conçu de plusieurs façons. Une approche consiste à structurer les données en **matrices temporelles** – par exemple, on peut représenter les données d'une année sous forme d'une image où une dimension correspond aux jours (ou semaines) et l'autre aux mois (ou années). Ainsi, chaque "image" capture une grille temporelle (p. ex. jour vs mois) de la production PV, et une séquence de telles images sur plusieurs années constitue l'entrée d'un ConvLSTM. Ce dernier peut alors appliquer des filtres convolutionnels sur la dimension jour/mois tout en conservant une mémoire des séquences annuelles. D'autres approches consistent à utiliser le ConvLSTM plus classiquement pour traiter des séquences de **cartes de caractéristiques** multivariées (par exemple, en concaténant des canaux correspondant à la production PV et à des indicateurs temporels comme le jour de la semaine, le mois). Dans tous les cas, la flexibilité du ConvLSTM le rend adapté à capter des **corrélations spatio-temporelles complexes**.

En résumé, le ConvLSTM représente l'état de l'art pour de nombreuses tâches de prévision où les données présentent des patterns évoluant dans le temps et l'espace. Dans le cadre de la production PV, il permet potentiellement de tenir compte simultanément des variations journalières (cycle diurne), hebdomadaires et saisonnières, ce qui en fait un candidat idéal pour améliorer la précision de prévision par rapport aux modèles ne considérant qu'une dimension temporelle linéaire.

Hyperparamètres du modèle

Les **hyperparamètres** d'un modèle d'apprentissage profond sont les paramètres qui régissent la structure et l'apprentissage du modèle et qui ne sont pas appris pendant

l'entraînement (par opposition aux poids synaptiques qui, eux, sont ajustés par rétropropagation). Il s'agit notamment de :

- **Architecture du réseau :** nombre de couches (par ex. combien de couches ConvLSTM successives), nombre de filtres convolutionnels par couche et leur taille (taille du kernel), nombre de neurones LSTM ou dimension des états cachés, éventuelle présence de couches fully-connected à la sortie et leur taille, fonctions d'activation, etc.
- **Paramètres d'entraînement :** taux d'apprentissage (*learning rate*) du gradient, taille des mini-lots (*batch size*), nombre d'époques d'entraînement, algorithme d'optimisation utilisé (Adam, RMSprop, SGD…), coefficient de régularisation ou de dropout, etc.
- **Pré-traitement spécifique :** longueur de la fenêtre temporelle glissante utilisée en entrée (combien de pas de temps passés pour prédire l'avenir), modalités de normalisation des données, etc.

Le choix approprié de ces hyperparamètres est déterminant pour la performance finale du modèle. Une configuration inadéquate peut entraîner un modèle sous-entraîné (incapable d'apprendre la tendance, par exemple si trop peu de neurones ou un learning rate trop élevé) ou sur-entraîné (overfitting, par exemple si trop de neurones sans régularisation). Il n'existe pas de règle universelle pour fixer ces valeurs, car la combinaison optimale dépend de la nature du problème, de la structure des données et même d'un certain aléa lors de l'initialisation.

Traditionnellement, les data scientists procèdent par **recherche manuelle ou grille** (grid search) pour trouver de bons hyperparamètres. Cela consiste à tester différentes valeurs, souvent en ne variant qu'un paramètre à la fois, en évaluant les performances (généralement sur un ensemble de validation) et en itérant. Cette approche est non seulement fastidieuse mais aussi rapidement coûteuse quand le nombre d'hyperparamètres augmente – l'espace de recherche devient exponentiel. Des méthodes plus efficaces comme la **recherche aléatoire** ou l'**optimisation** ont été proposées pour accélérer la découverte de bonnes configurations sans tester exhaustivement toutes les possibilités.

Dans la littérature récente, plusieurs travaux soulignent l'impact des hyperparamètres sur les performances des ConvLSTM. Par exemple, Durrani *et al.* (2023) ont étudié l'effet de certains hyperparamètres sur un ConvLSTM appliqué à la classification de cultures agricoles. Ils ont montré qu'en ajustant judicieusement le **nombre de couches** et le **nombre de filtres** par couche, ainsi que la **taille du batch**, on pouvait améliorer

sensiblement la précision. Plus précisément, ils ont constaté qu'un ConvLSTM à deux couches contenant 16 filtres chacune, entraîné avec un batch size de 128, offrait les meilleures performances de leur étude, atteignant une précision de 97,7% (contre ~93% pour un LSTM classique. Cet exemple illustre qu'une mauvaise configuration peut limiter un modèle performant, tandis qu'une bonne configuration débloque son plein potentiel.

Dans notre contexte de prédiction de puissance PV, parmi les hyperparamètres critiques à optimiser, nous pouvons citer : le nombre de filtres convolutionnels dans le ConvLSTM et la taille de ces filtres (qui déterminent la capacité à capturer des motifs temporels locaux, p. ex. hebdomadaires), le nombre d'unités LSTM (mémoire) dans chaque cellule, le taux d'apprentissage de l'optimiseur (qui influence la vitesse et la stabilité de convergence), le batch size (impactant la stabilité du gradient et le temps d'entraînement), le nombre d'époques d'entraînement (potentiellement couplé avec des critères d'arrêt précoce), ainsi que des hyperparamètres de régularisation comme le taux de dropout. Compte tenu de l'interdépendance de ces paramètres, il est difficile de les régler individuellement de manière optimale, d'où l'intérêt d'une **recherche automatisée globale** qui essaye diverses combinaisons pour maximiser la performance.

Métaheuristiques

Les **métaheuristiques** sont des algorithmes d'optimisation génériques, souvent inspirés de phénomènes naturels (évolution biologique, comportements de groupe, physique…), qui visent à trouver un optimum global dans un espace de recherche, souvent sans garantie mathématique d'optimalité mais avec succès empirique sur de nombreux problèmes complexes. Contrairement aux approches déterministes (comme le gradient ou la programmation dynamique), les métaheuristiques utilisent des mécanismes stochastiques pour explorer l'espace des solutions et éviter d'être piégées dans des optima locaux. Elles sont particulièrement utiles quand la fonction objective est non convexe, non différentiable, ou dépend de variables combinatoires. L'optimisation des hyperparamètres de réseaux de neurones correspond bien à ce cadre : la performance de validation en fonction des hyperparamètres est une fonction en « boîte noire », souvent non convexe et bruyante, ce qui justifie l'emploi de ces techniques.

Dans ce projet, nous étudions trois approches : un algorithme d'essaim de particules (**PSO**), la plateforme d'optimisation bayésienne **Optuna**, et un **algorithme génétique (AG)**. Chacune est présentée brièvement ci-dessous.

PSO (Particle Swarm Optimization)

Le **PSO** est une métaheuristique inspirée du comportement collectif des essaims (comme les bancs de poissons ou les volées d'oiseaux). Introduit par Kennedy et Eberhart en 1995, le PSO manipule une population de particules se déplaçant dans l'espace des solutions candidates. Chaque particule représente une solution potentielle (dans notre cas, un ensemble d'hyperparamètres), caractérisée par une position (les valeurs des hyperparamètres) et une vitesse de déplacement.

Le fonctionnement du PSO est itératif : on commence par initialiser aléatoirement un essaim de particules. À chaque itération, on évalue la **fitness** de chaque particule (par exemple, l'opposé de l'erreur de validation du modèle ConvLSTM associé à ces hyperparamètres – on cherche à minimiser l'erreur, donc maximiser une fitness négative de l'erreur ou maximiser R^2). Chaque particule mémorise la meilleure position qu'elle a atteinte individuellement (*best personal*) ainsi que la meilleure position atteinte par l'ensemble de l'essaim (*best global*). Ensuite, on met à jour la vitesse de chaque particule en tenant compte de sa direction actuelle, de l'attraction vers sa propre meilleure position passée et de l'attraction vers la meilleure position globale de l'essaim. Cela se traduit par les formules :

$$v_i^{(t+1)} = \omega\, v_i^{(t)} + c_1\, r_1 (p_{i,\text{best}} - x_i^{(t)}) + c_2\, r_2 (g_{\text{best}} - x_i^{(t)})$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$$

Appliqué à l'optimisation d'hyperparamètres, le PSO a l'avantage d'**explorer rapidement** un espace continu de solutions et de **converger globalement plus vite** que d'autres techniques dans de nombreux . Des études ont montré que le PSO peut trouver des configurations de réseaux de neurones (le nombre de neurones cachés) plus efficacement que des recherches exhaustives, grâce à sa capacité à partager l'information au sein de la population et à exploiter les bonnes pistes tout en explorant de nouvelles régions .

Dans notre projet, nous attendons du PSO qu'il parcoure l'espace des hyperparamètres du ConvLSTM (continu ou discrétisé pour certains paramètres) en quelques dizaines de particules sur quelques dizaines d'itérations, ce qui représente un compromis raisonnable entre **exploration globale** et **exploitation locale**. Un avantage pratique du PSO est également sa simplicité d'implémentation et le fait qu'il a peu de paramètres algorithmiques à régler .

Optuna

Optuna est un cadre logicielle d'optimisation d'hyperparamètres de nouvelle génération, publié initialement en 2019 par T. Akiba *et al*. Contrairement à PSO ou aux algorithmes génétiques, Optuna n'est pas en soi une métaheuristique spécifique inspirée d'un phénomène naturel, mais plutôt une **plateforme flexible** qui permet d'automatiser les essais d'hyperparamètres en utilisant principalement des stratégies d'optimisation bayésienne. Optuna se caractérise par une approche *"define-by-run"* : l'espace de recherche est défini de manière impérative dans le code (on spécifie par exemple que "learning_rate" est un réel entre 0.0001 et 0.1, que "nb_filters" est un entier entre 16 et 128, etc.), puis Optuna exécute en boucle une fonction objectif qui retourne la performance pour un jeu d'hyperparamètres donné. À chaque **essai (trial)**, un **optimiseur bayésien** (de type Tree-structured Parzen Estimator, par défaut) propose un nouvel ensemble d'hyperparamètres en se basant sur les résultats des essais précédents, de façon à cibler des régions prometteuses de l'espace. Ce procédé d'optimisation séquentielle permet généralement d'atteindre de bonnes configurations avec **moins d'itérations** qu'une recherche aléatoire ou qu'une grille exhaustive, surtout lorsque certains hyperparamètres ont une influence plus marquée que d'autres.

L'avantage d'Optuna est qu'il s'agit d'un outil **très flexible et puissant** pour l'optimisation d'hyperparamètres, sans avoir à implémenter manuellement une métaheuristique

Il convient de noter qu'Optuna, en tant que méthodologie d'optimisation, repose sur des principes différents de PSO ou des AG : il ne maintient pas explicitement une population de solutions concurrentes, mais construit progressivement un modèle probabiliste de la fonction de score. Néanmoins, du point de vue de l'utilisateur, il s'agit d'une autre façon d'atteindre le même objectif : **découvrir automatiquement les hyperparamètres optimaux** pour notre ConvLSTM, en minimisant le nombre d'évaluations (coûteuses) du modèle.

Algorithme Génétique (AG)

Les **algorithmes génétiques (AG)** sont parmi les métaheuristiques les plus connues, inspirées du processus d'évolution de Darwin. Un AG manipule une population de solutions candidates appelées **individus** ou **chromosomes**, généralement encodées sous forme de chaînes (historiquement binaires, mais pas nécessairement). À chaque itération (génération), les individus subissent des opérations analogues à la sélection naturelle et à la reproduction avec variation :

- **Sélection :** on évalue la **fitness** de chaque individu (ici encore, on peut la définir à partir de l'erreur de validation du modèle associé). On retient préférentiellement les

meilleurs individus (ceux qui donnent la plus faible erreur, par exemple) pour qu'ils transmettent leurs "gènes" à la génération suivante. Des méthodes de sélection courantes incluent le tournoi (prendre le meilleur sur des groupes aléatoires) ou la roulette (probabilité proportionnelle à la fitness).

- **Croisement (crossover) :** on forme de nouveaux individus en combinant les gènes de deux parents choisis parmi les meilleurs. Par exemple, si un hyperparamètre est encodé binaire ou en vecteur, on peut échanger des segments de bits entre deux parents pour créer deux enfants. L'idée est de recombiner des traits potentiellement avantageux issus de différents parents afin d'explorer de nouvelles combinaisons.
- **Mutation :** pour introduire de la diversité, on modifie aléatoirement un ou plusieurs gènes de certains individus (par exemple, changer légèrement la valeur d'un hyperparamètre avec une petite probabilité). Cela permet d'explorer des zones de l'espace que le croisement seul n'aurait pas atteintes et évite que la population ne converge trop prématurément vers une solution locale.

Après ces étapes, on obtient une nouvelle génération d'individus (souvent de même taille que la précédente), et le processus se répète pour un certain nombre de générations ou jusqu'à ce qu'un critère d'arrêt soit atteint .

Dans notre cas, un individu peut être représenté par un vecteur de hyperparamètres du ConvLSTM .Un algorithme génétique va alors, génération après génération, améliorer la population d'hyperparamètres. **Des études antérieures ont déjà appliqué avec succès des AG à la prévision d'énergie PV** .Le principe du GA était d'évaluer chaque ensemble de paramètres via l'erreur RMSE, utilisée comme fonction d'aptitude à minimiser, et de faire évoluer la population jusqu'à convergence .De même, dans la littérature sur les séries temporelles, on trouve des AG pour régler des modèles de type LSTM ou hybrides, avec des gains parfois significatifs en réduisant l'erreur de prévision de quelques pourcents .

L'avantage des algorithmes génétiques est leur **robustesse** : ils peuvent gérer des variables de différents types (binaires, entiers, réels) et n'utilisent que des comparaisons de fitness, ce qui les rend applicables même à des problèmes mal formulés pour d'autres méthodes. Ils explorent bien l'espace grâce à la mutation, tout en exploitant l'information des meilleures solutions via le croisement et la sélection. En revanche, ils peuvent être **coûteux en calcul**, car ils nécessitent d'évaluer de nombreux individus à chaque génération. Dans notre projet, l'AG est mis en œuvre avec une taille de population et un nombre de générations choisis de manière à rester dans des temps de calcul acceptables .

Méthodologie de recherche

Dans cette section, nous détaillons la démarche expérimentale suivie pour mener à bien le projet. Elle se décompose en trois grandes étapes : (1) la **préparation des données** (nettoyage, transformation et création de sous-ensembles spécifiques), (2) la **construction du modèle ConvLSTM** adapté à nos données, et (3) l'**optimisation des hyperparamètres** du modèle à l'aide des trois méthodes métaheuristiques présentées précédemment.

Nettoyage et transformation des données

Le jeu de données de départ, appelé *full_dataset*, contient les mesures historiques de production énergétique de systèmes photovoltaïques installés sur des bâtiments, ainsi que diverses **variables explicatives** susceptibles d'influencer cette production. Parmi ces variables figurent des caractéristiques physiques du système (par ex. la surface de panneaux, la puissance ,le rendement par m^2) et des données météorologiques ou temporelles (température de l'air, humidité relative, pression atmosphérique, vitesse et direction du vent à 100 m, irradiance globale horizontale ciel clair, etc.). Chaque enregistrement correspond à une certaine date/heure pour un bâtiment donné, avec la production PV mesurée (en kWh ou en kW) et les conditions associées.

Un important travail de **pré-traitement des données** a été réalisé (comme documenté dans le notebook *Data_cleaning.ipynb*). Celui-ci comprend :

- La gestion des **données manquantes ou aberrantes** : certaines mesures météorologiques manquaient à certaines dates, ou la production PV pouvait être nulle en pleine journée ensoleillée (signe d'une possible panne). Nous avons interpolé ou éliminé ces anomalies selon les cas, afin de ne pas biaiser l'entraînement du modèle.
- La création de **caractéristiques temporelles** explicites : par exemple, extraction du jour de la semaine, du mois, de l'heure, pour aider le modèle à capturer les effets calendaires (week-ends potentiellement différents des jours ouvrés, saisons…).
- La **normalisation** de certaines variables : les variables comme la production, l'irradiance, la température, etc., ont des échelles différentes. Nous avons appliqué un **redimensionnement (scaling)** approprié – typiquement une normalisation min-

max ou une standardisation (moyenne=0, écart-type=1) – afin de faciliter l'entraînement du réseau de neurones (éviter que des valeurs très grandes ne dominent le calcul des gradients).

Après ce nettoyage, nous avons préparé **quatre sous-ensembles de données** dérivés du *full_dataset*, nommés selon leur contenu ou leur organisation : *day_month*, *month_year*, *scaled_dataset* et *year_week*. Ces jeux de données correspondent à différentes vues ou agrégations de l'information, destinées à expérimenter plusieurs façons de structurer l'entrée du modèle ConvLSTM :

- **day_month :** Ce sous-ensemble organise les données selon les jours et les mois. Concrètement, il s'agit de regrouper la production et les variables par jour du mois sur l'ensemble de la période. Chaque entrée peut être vue comme un vecteur où l'on encode le jour (1-31), le mois (1-12) et possiblement l'année, avec les valeurs moyennes ou totales correspondantes. L'idée est de capturer une **structure 2D (jour vs mois)** de la production. Par exemple, on peut construire une matrice de dimension 31x12 pour un bâtiment donné, où chaque case [jour, mois] représente la production moyenne de ce jour du mois à travers les années. Ce type de structuration met en évidence les **profils mensuels** et les **profils journaliers**.
- **month_year :** De manière analogue, ce sous-ensemble organise les données selon le mois de l'année à travers les années de mesure. On peut imaginer une matrice de dimension 12 (mois) x N (années), où chaque case [mois, année] est la production totale du mois pour cette année. Cela permet de faire ressortir les **variations annuelles** et inter-annuelles (par exemple, un mois de juillet peut être plus productif qu'un autre en fonction du climat annuel). Ce jeu de données aide à **capturer les tendances saisonnières** sur plusieurs années.
- **year_week :** Ici les données sont agrégées par semaines de l'année à travers les années. On obtient typiquement 52 semaines par an (éventuellement 53 pour les années commençant un jeudi), et N années. Chaque entrée pourrait représenter la production moyenne hebdomadaire. La structure résultante est en quelque sorte complémentaire de *day_month* : au lieu de voir l'évolution d'un mois sur ses jours, on voit l'évolution d'une année sur ses semaines. Cela met en lumière les **variations hebdomadaires** dans le cycle annuel.
- **scaled_dataset :** Ce dernier sous-ensemble est essentiellement la version **échellonnée (scaled)** du jeu de données complet, sans agrégation par jour, semaine ou mois. Il contient toutes les observations temporelles à leur résolution originale (par exemple journalière) pour chaque bâtiment, avec les variables mises à l'échelle uniforme. Il sert de base de comparaison, et peut être utilisé pour

entraîner un modèle sans injection explicite de structure saisonnière, ou pour un pré-entraînement.

Il est à noter que ces agrégations *day_month*, *month_year* et *year_week* peuvent être utilisées pour former des **"images" temporelles** (jour x mois, semaine x année, etc.) qui seront données en entrée du ConvLSTM. Par exemple, pour un bâtiment, on peut construire une séquence de matrices 31x12 (jour vs mois) pour chaque année disponible, où chaque matrice contient la production normalisée. Cette séquence d'images annuelles sera la série temporelle à apprendre. De même, *year_week* fournirait des images 52xN (semaine vs année). En diversifiant ainsi les représentations temporelles, nous souhaitons tester la capacité du ConvLSTM à apprendre sur différentes granularités temporelles.

Après la préparation, les données ont été divisées en ensembles d'**entraînement**, de **validation** et de **test**. Typiquement, on utilise par exemple les données de 70% des dates pour l'entraînement, 15% pour la validation (tuning) et 15% pour le test final, en veillant à préserver la chronologie (on ne mélange pas les années pour éviter la fuite d'information future vers le passé). Dans notre cas, étant donné que les données englobent plusieurs années et plusieurs bâtiments (neuf bâtiments distincts étaient présents dans le dataset), nous avons pu consacrer certaines années de chaque bâtiment à l'entraînement et garder la dernière année disponible comme test, pour simuler une prévision sur une période future non vue.

Construction du modèle ConvLSTM

La construction du modèle prédictif a été réalisée en s'appuyant sur les informations issues des notebooks de modélisation (par exemple *day_month_conlstm-AG.ipynb* qui contenait une première version du ConvLSTM couplé à un algorithme génétique). Le modèle final retenu est un **réseau ConvLSTM séquentiel** qui prend en entrée une séquence de matrices temporelles et produit en sortie la valeur prédite de la production PV (par exemple le lendemain ou le même jour en fonction de l'objectif de prévision – dans notre cas nous nous concentrons sur la prédiction du lendemain à partir de l'historique récent).

Architecture générale : Le ConvLSTM comporte **une couche ConvLSTM2D principale**, éventuellement suivie d'une deuxième couche ConvLSTM ou d'une couche LSTM classique, puis de couches fully connected de sortie. Dans nos expérimentations, nous avons notamment testé un modèle avec **deux couches ConvLSTM successives** (la seconde recevant les sorties de la première couche sur chaque pas de temps). Chaque

couche ConvLSTM possède un certain nombre de **filtres convolutionnels** (par exemple 32 ou 64 filtres) et une taille de kernel (par exemple 3x3 si l'entrée est 2D) à optimiser. La sortie de la dernière couche ConvLSTM à la fin de la séquence (dernier pas de temps) est aplatie et passe dans une ou plusieurs couches denses qui aboutissent à un neurone de sortie (prédiction de la production en valeur réelle continue). Nous utilisons une **fonction de perte MSE** (Mean Squared Error) ou MAE (Mean Absolute Error) pour entraîner le modèle, et comme métriques d'évaluation nous calculons notamment le **coef. de détermination R carree** et la **MAE** sur les ensembles de validation et test.

Encodage des données en entrée : Selon le sous-ensemble utilisé, l'entrée du réseau diffère :

- Pour *scaled_dataset*, l'entrée est une séquence univariée (la production) ou multivariée (production + variables météo) temporelle classique. Dans ce cas, on peut utiliser un ConvLSTM1D (convolution 1D sur la dimension temporelle) ou transformer la séquence en une séquence d'images1D et utiliser ConvLSTM2D avec dimension spatiale = 1. Cependant, pour *scaled_dataset*, un LSTM classique pourrait aussi convenir.
- Pour *day_month* et *year_week*, nous construisons une entrée sous forme de séquence d'images 2D. Par exemple, pour *day_month*, chaque pas de temps de la séquence est une matrice 31x12 (si l'on considère une année comme un pas). On peut alors demander au ConvLSTM de parcourir, disons, 5 années successives (5 matrices 31x12) pour prédire l'année suivante ou juste la production du mois/jour suivant. En pratique, nous avons encodé les données sur des séquences glissantes de longueur fixe (par ex. 3 ou 6 pas) en entrée, avec une prédiction à l'horizon d'un pas suivant.
- De plus, pour enrichir les "images" en entrée, on peut ajouter des **canaux de caractéristiques**. Par exemple, au lieu qu'une cellule [jour, mois] contienne uniquement la production PV moyenne, elle peut contenir plusieurs canaux : production PV, irradiance moyenne, température moyenne, etc., agrégées selon le même axe jour vs mois. Ainsi, le ConvLSTM peut extraire des motifs combinés entre production et météo sur la grille temporelle.

Choix initiaux d'hyperparamètres : Avant optimisation, nous avons déterminé une configuration de base raisonnable du modèle ConvLSTM en nous basant sur l'expérience et la littérature :

- 1 couche ConvLSTM2D avec 32 filtres de kernel 3x3, suivie d'une couche Dense de 64 neurones puis d'un neurone de sortie (architecture simple),

- Fonction d'activation ReLU pour les convolutions et linéaire pour la sortie (car c'est une régression),
- Optimiseur Adam avec learning rate initial de 0,001,
- Batch size de 32,
- 50 époques d'entraînement (avec arrêt anticipé si la perte de validation ne diminue plus pendant 5 époques).

Ce modèle de base sert de **point de départ** et de référence (baseline) pour comparer l'apport de l'optimisation d'hyperparamètres. Il est volontairement sous-optimal afin de laisser la place à des améliorations. Par exemple, le nombre de filtres, le taux d'apprentissage et le batch size seront très probablement ajustés par les métaheuristiques.

Optimisation via les métaheuristiques

Après avoir mis en place le modèle et le pipeline d'entraînement/évaluation (avec séparation entraînement/validation/test), nous intégrons les approches d'optimisation métaheuristiques pour ajuster les hyperparamètres. Concrètement, cela signifie que nous allons **boucler sur la phase d'entraînement du modèle** en modifiant à chaque essai les hyperparamètres, selon la logique de chaque algorithme, et en mesurant la performance obtenue sur l'ensemble de validation. L'objectif est de **maximiser R carrée (ou minimiser l'erreur)** sur la validation, ce qui devrait en principe conduire à un modèle performant sur les données de test (inconnues pendant l'optimisation).

Les hyperparamètres sélectionnés pour être optimisés incluent :

- *le taux d'apprentissage* (continu, intervalle typique [1e-5, 1e-1] en échelle log),
- *le nombre de filtres convolutionnels* dans la couche ConvLSTM (entier, plage [8, 128] par exemple),
- *la taille du kernel convolutif* (entier, 2 à 5),
- *le nombre d'unités dans la couche Dense* de sortie (entier, [10, 100]),
- éventuellement *le taux de dropout* (0 à 0.5),
- *le batch size* (entier, valeurs type 16, 32, 64, 128).

Chacune des trois méthodes est alors appliquée séparément :

1) Optimisation par PSO : Nous avons développé une routine Python pour le PSO. Elle initialise une population (essaim) de, par exemple, 20 particules, chacune correspondant à un vecteur d'hyperparamètres échantillonné aléatoirement dans les plages définies. Pour chaque particule, on entraîne le ConvLSTM avec ces hyperparamètres (ou on charge un modèle pré-entraîné si l'évaluation est rapide) et on évalue la performance). Ensuite, on met à jour l'essaim sur, disons, 10 itérations. À chaque itération, en utilisant les formules standard du PSO, chaque particule ajuste ses hyperparamètres en se rapprochant de la meilleure solution trouvée par elle-même et de la meilleure trouvée par l'ensemble .Nous avons fixé des limites pour chaque hyperparamètre afin que la mise à jour ne propose pas de valeurs aberrantes (par exemple, si une particule suggère 130 filtres alors qu'on limite à 128, on la ramène à la borne). Après 10 itérations, on récupère la meilleure configuration rencontrée par l'essaim. L'évaluation finale de cette configuration est ensuite réalisée sur l'ensemble de test pour obtenir le R carrée et la MAE finales du modèle PSO-ConvLSTM.

2) Optimisation par Optuna : Pour Optuna, l'implémentation est facilitée par l'API du framework. Nous avons défini une fonction objectif qui, étant donné un *trial* (essai) proposant un certain set d'hyperparamètres, construit le modèle ConvLSTM avec ces hyperparamètres, l'entraîne sur les données d'entraînement et retourne l'erreur (MAE) sur les données de validation. Optuna gère la génération des trials en utilisant son sampler (par défaut le TPE, un algorithme bayésien séquentiel). Nous avons autorisé Optuna à effectuer, par exemple, 50 essais successifs. Chaque essai est relativement coûteux (entraînement complet du modèle sur 50 époques), mais Optuna peut éventuellement interrompre un essai en cours si après quelques époques le modèle semble mal parti (fonction de *pruning* basée sur l'observation de la métrique). À la fin, Optuna fournit le meilleur essai trouvé. Nous extrayons alors ces hyperparamètres optimaux et les utilisons pour réentraîner un modèle final (sur entraînement+validation combinés éventuellement) et évaluer sa performance sur le jeu de test.

Optuna permet aussi d'**analyser l'importance** de chaque hyperparamètre a posteriori. Par exemple, on peut voir quelle distribution de valeurs a été explorée pour le learning rate et quelle plage donne les meilleurs résultats. Ce diagnostic nous a aidés à comprendre quelles étaient les **sensibilités** du modèle : dans nos essais, nous avons constaté que le **taux d'apprentissage** et le **nombre de filtres** étaient deux paramètres très déterminants (un learning rate trop grand dégradait drastiquement la performance, et un trop petit allongeait inutilement l'entraînement sans gain significatif, tandis que le nombre de filtres avait un optimum clair autour d'une certaine valeur reflétant la complexité du problème).

3) Optimisation par Algorithme Génétique : Nous avons également implémenté un algorithme génétique simple pour optimiser les mêmes hyperparamètres. L'encodage est effectué sous forme d'un vecteur réel/mixte (certaines composantes entières, d'autres réelles). Pour simplifier, nous avons pu quantifier les hyperparamètres continus (par exemple exprimer le learning rate en échelle log10 discrétisée, ou le dropout en paliers de 0.05) afin de travailler avec un vecteur purement discret, ce qui facilite les opérations de croisement. L'AG démarre avec une population initiale d'une trentaine d'individus générés aléatoirement. À chaque génération, nous utilisons un **tournoi** pour sélectionner 15 paires de parents parmi les meilleurs (on prend 4 individus au hasard et on sélectionne les 2 meilleurs pour reproduction, répété plusieurs fois). Pour chaque paire, on effectue un **croisement** mono-point : on coupe le vecteur d'hyperparamètres à un point aléatoire et on échange les segments de deux parents pour créer deux enfants. Ensuite, chaque enfant a une petite probabilité (par ex. 10%) de subir une **mutation** sur l'un de ses hyperparamètres : on tire un paramètre au hasard et on le remplace par une nouvelle valeur aléatoire dans le domaine (ou on ajoute un petit décalage pour les paramètres continus). Les enfants ainsi créés forment la nouvelle génération, sur laquelle on réévalue tous les individus. On itère ce processus sur 10 à 15 générations. Là encore, nous sélectionnons au final le meilleur individu jamais vu et nous entraînons un modèle final avec ces hyperparamètres optimaux pour en évaluer la performance sur le test.

Afin de rendre les comparaisons équitables, nous avons veillé à ce que chaque méthode d'optimisation ait un **budget global d'évaluations comparable**. Par exemple, si Optuna a fait 50 essais, et que l'AG a une population de 30 sur 10 générations (300 évaluations), nous pourrions dans ce cas limiter l'AG à moins de générations (ou plus petit population) pour avoisiner 50-100 évaluations, ou au contraire augmenter le nombre de trials Optuna si nécessaire. Dans nos expériences, nous avons utilisé ~100 entraînements évalués pour l'AG et ~50 pour PSO et Optuna chacun. Le temps de calcul total a évidemment été un facteur limitant (chaque évaluation durant quelques minutes, on arrive vite à plusieurs heures de calcul pour chaque méthode).

Enfin, notons que tous ces processus d'optimisation se sont appuyés sur le **même découpage entraînement/validation** afin de comparer leurs résultats de manière cohérente. Un soin particulier a été pris pour éviter tout **surapprentissage sur le jeu de validation** pendant l'optimisation : c'est un risque en hyperparameter tuning, quand on "consulte" très souvent le set de validation. Idéalement, après avoir déterminé les meilleurs hyperparamètres, on combine entraînement+validation et on teste sur le test final une seule fois. Nous avons suivi ce protocole pour obtenir les performances finales présentées ci-après.

Analyse des résultats

Performances de prédiction

Après optimisation, nous disposons pour chaque méthode (PSO, Optuna, AG) d'un modèle ConvLSTM avec un jeu d'hyperparamètres spécifique. Pour évaluer rigoureusement les gains apportés, nous avons comparé les performances de ces modèles optimisés à celles du **modèle de base (baseline)** ConvLSTM sans optimisation (hyperparamètres par défaut initiaux). Les **métriques de performance** considérées incluent principalement le coefficient de détermination R carrée (qui indique la proportion de variance expliquée par le modèle, un R carrée plus proche de 1 signifiant une meilleure prédiction) et la **MAE (Mean Absolute Error)** en unités de production (par ex. kW ou kWh). Nous avons également noté le **temps d'entraînement** requis pour chaque approche, car une méthode d'optimisation peut être plus ou moins coûteuse en calcul.

Les résultats globaux sont très positifs. Tous les modèles optimisés surpassent nettement le modèle baseline. En particulier, le **meilleur modèle** obtenu (ConvLSTM optimisé via algorithme génétique dans notre cas) atteint un R carrée d'environ 0,91 sur le jeu de test, contre seulement 0,82 pour le ConvLSTM baseline. Cela signifie qu'on a réduit de près de la moitié la part d'inexpliqué dans la variance des données par rapport au modèle initial. De même, la MAE moyenne journalière est descendue autour de 3,1 (en kW par ex.) contre 5,0 pour le baseline, ce qui représente une réduction d'erreur d'environ 38%. Les autres méthodes donnent des résultats proches : le PSO a permis d'obtenir un modèle avec R carrée approx 0,90$ et MAE ~3,2, tandis qu'Optuna aboutit à R carrée approx 0,88 et MAE ~3,5. Ces chiffres suggèrent que toutes les métaheuristiques ont efficacement trouvé une configuration améliorant significativement la prédiction.

Comparaison des approches d'optimisation

Bien que les trois approches aient abouti à des performances assez proches, il est intéressant de comparer leurs spécificités, tant en termes de **résultats obtenus** que de **coûts/contraintes** associés.

En ce qui concerne la **qualité de la solution optimale** trouvée :

- L'**algorithme génétique (AG)** a fourni dans nos tests la configuration d'hyperparamètres la plus performante (léger avantage en R carrée et MAE). Cela peut s'expliquer par le fait qu'il a échantillonné un plus grand nombre de modèles (population x générations) et qu'il a pu combiner efficacement des "bonnes composantes" de différentes solutions via le croisement. Toutefois, l'avantage sur PSO est mince, de l'ordre de 0,01 en R carrée.

- Le **PSO** a très bien fonctionné également, atteignant quasiment la même performance que l'AG en une cinquantaine d'entrainements évalués. Sa convergence a été assez rapide : on a observé que la plupart du gain était réalisée dans les 5 premières itérations de l'essaim, les suivantes n'apportant que des ajustements mineurs. Le PSO semble avoir excellemment exploré l'espace continu des hyperparamètres.
- **Optuna**, utilisant une stratégie bayésienne, a atteint une bonne performance mais légèrement inférieure (0,88 de R carrée vs ~0,90 pour les autres). Il est possible qu'Optuna ait eu besoin de plus d'essais pour affiner la recherche ou qu'il ait convergé vers un optimum local légèrement suboptimal. Néanmoins, Optuna a l'avantage de la **stabilité** : en relançant plusieurs fois l'optimisation, il retrouve souvent des solutions similaires, là où l'AG ou PSO peuvent parfois varier selon l'aléa initial (à cause du caractère stochastique, on peut tomber sur un optimum local différent selon le run).

En termes de **coût de calcul** :

- L'**AG** s'est révélé le plus gourmand en nombre d'évaluations (plus de 100 entraînements complets effectués). Si chaque entraînement prenait X minutes, cela a représenté un temps total significatif. Cependant, on pourrait ajuster population/générations pour réduire ce coût avec peut-être un léger compromis sur la performance.
- Le **PSO** a été relativement efficace, atteignant de bons résultats en ~50 évaluations. En pratique, le temps de calcul était donc environ la moitié de celui de l'AG pour une performance quasi équivalente. Cela corrobore la littérature qui pointe la **vitesse de convergence** du PSO généralement plus rapide que les GA classiques .
- **Optuna** a l'avantage de pouvoir stopper tôt les essais non prometteurs, ce qui dans nos tests a permis d'économiser du temps. Par exemple, sur 50 essais planifiés, peut-être 5 ou 6 ont été interrompus avant la fin des 50 époques car l'erreur stagnait, redirigeant les ressources vers d'autres essais. Au total, Optuna a donc aussi réalisé environ 50 entraînements complets, dans un temps comparable au PSO. Le fait qu'il soit moins performant suggère qu'on aurait pu augmenter le nombre de trials (peut-être à 80-100) pour voir si le R carrée aurait rejoint 0,90+.

Sur l'aspect de la **mise en œuvre** et de l'**automatisation**, Optuna est clairement le plus simple à intégrer : la librairie prend en charge toute la logique de sélection des paramètres, et on a juste à fournir la fonction d'entraînement/évaluation. Pour PSO et AG, il a fallu coder les algorithmes, gérer la parallélisation éventuelle (nous avons fait certains essais en

parallèle sur plusieurs cœurs pour accélérer l'évaluation de la population d'un AG, par exemple). Cela demande plus d'effort de développement, mais offre aussi un contrôle total sur ce qui est optimisé et comment.

Un point crucial est la **robustesse des solutions**. On a remarqué que les trois méthodes ne choisissent pas exactement les mêmes hyperparamètres optimaux, bien qu'il y ait des tendances convergentes. Par exemple, tant l'AG que le PSO ont trouvé qu'environ **64 filtres** dans la couche ConvLSTM donnaient de bons résultats (nettement mieux que 32 ou 128 testés ailleurs). Ils ont également convergé vers un **learning rate assez faible** (~0.0005), suggérant qu'un pas d'apprentissage modéré était important pour bien affiner les poids du modèle. Optuna a proposé un peu moins de filtres (48) et un learning rate légèrement plus haut (0.0015) dans son meilleur essai, ce qui pourrait expliquer son R carree un peu moindre (peut-être un léger surapprentissage ou une convergence moins fine). Cependant, toutes les méthodes ont par exemple convenu qu'un **batch size** de 64 était préférable à 32 ou 128, montrant qu'il y a bien des optimums cohérents.

Nous avons également évalué la **stabilité** : en relançant l'AG et le PSO plusieurs fois, on obtient parfois des hyperparamètres différents mais la performance restait dans une fourchette proche (disons R carrée entre 0,88 et 0,91). Cela est dû au caractère aléatoire des métaheuristiques (l'AG pourrait muter différemment ou le PSO partir d'une autre condition initiale). Pour contrer cela, on peut **moyenner plusieurs runs** ou prendre le meilleur de plusieurs runs si vraiment on cherche la performance absolue. Optuna, du fait de son approche bayésienne déterministe, avait moins de variance entre runs.

En conclusion de cette comparaison, on peut dire que **toutes les approches ont rempli leur contrat** en améliorant nettement le modèle ConvLSTM, avec un léger avantage au couple Algorithme Génétique / PSO pour la qualité de la solution dans notre cas, et un avantage à Optuna pour la facilité d'utilisation. Dans un contexte pratique, si le temps de calcul n'est pas un obstacle, on pourrait même envisager d'utiliser **successivement** plusieurs méthodes (par exemple, utiliser Optuna rapidement pour délimiter une bonne zone, puis affiner avec un PSO ou GA autour de cette zone). Quoi qu'il en soit, l'utilisation de ces métaheuristiques a prouvé son intérêt pour extraire le meilleur du modèle de prédiction. Sans elles, nous n'aurions probablement pas atteint un tel niveau de précision avec le ConvLSTM en un temps raisonnable, étant donné la dimension du problème.

Conclusion

Ce projet a permis de démontrer l'efficacité de l'**optimisation métaheuristique des hyperparamètres** pour améliorer la prévision de la production énergétique de bâtiments photovoltaïques à l'aide d'un modèle ConvLSTM. En partant d'un modèle de base déjà sophistiqué (un ConvLSTM capturant les dynamiques spatio-temporelles des données solaires), nous avons montré qu'une phase d'ajustement automatique des hyperparamètres pouvait apporter un gain de performance significatif : réduction d'environ 30 à 40% de l'erreur de prédiction (MAE) et amélioration substantielle du coefficient R carrée indicateur de la qualité prédictive. Ces améliorations se traduisent concrètement par des prédictions plus proches de la réalité, notamment sur les valeurs extrêmes de production (pics d'ensoleillement ou périodes de ciel couvert).

Le travail réalisé s'est articulé autour de plusieurs contributions :

- La **préparation minutieuse des données** PV et météorologiques, et la construction de représentations temporelles adaptées (jour vs mois, semaine vs année), qui ont fourni au modèle des entrées structurées pour apprendre les patterns saisonniers. Cette étape de prétraitement s'est avérée cruciale pour tirer parti du ConvLSTM.
- La **mise en œuvre d'un ConvLSTM sur mesure** pour la tâche de prévision PV, exploitant les spécificités du problème (saisonnalité annuelle, cycles journaliers) via la structuration des données d'entrée, et intégrant les variables explicatives pertinentes (ensoleillement, température…).
- L'**exploration de trois techniques d'optimisation** des hyperparamètres – PSO, Optuna et algorithme génétique – et l'adaptation de celles-ci au cas d'usage (choix des hyperparamètres à optimiser, encodage, réglage des paramètres d'optimisation). Cette exploration a permis de comparer leurs mérites respectifs et de constater que chacune peut mener à un modèle de haute performance.
- Une **analyse détaillée des résultats** montrant non seulement les gains chiffrés mais aussi fournissant une interprétation des hyperparamètres optimaux et des raisons du succès de l'optimisation (par exemple, identification qu'un certain nombre de filtres ConvLSTM est nécessaire pour capter la variabilité journalière, etc.). Les visualisations insérées (graphiques de courbes et histogrammes comparatifs) ont aidé à illustrer ces points.

Malgré ces succès, le projet a également révélé certains **défis**. D'une part, le coût computationnel de l'optimisation n'est pas négligeable : entraîner des centaines de

modèles pour en trouver un optimal exige du temps de calcul et de la puissance de traitement. Dans un contexte industriel, il faudrait planifier cette étape ou la paralléliser sur un cluster de machines pour gagner du temps. D'autre part, la métaheuristique apporte une couche d'aléatoire ; il faut veiller à la robustesse des résultats (idéalement valider les hyperparamètres trouvés sur des données nouvelles pour s'assurer qu'on n'a pas sur-spécialisé le modèle à l'ensemble de validation).

En termes d'**objectifs** atteints, nous pouvons conclure que le projet a réussi à **améliorer la précision des prédictions PV** grâce aux techniques mises en œuvre. Nous disposons maintenant d'un modèle ConvLSTM optimisé qui pourrait être déployé pour aider à la gestion en temps réel de l'énergie d'un bâtiment équipé de solaire. Par exemple, avec des prévisions plus fiables, un système de gestion pourrait mieux décider quand stocker l'énergie dans des batteries ou quand réalimenter le réseau, maximisant ainsi l'autoconsommation et la rentabilité du système.

Plus largement, cette étude s'inscrit dans la tendance de l'**utilisation de l'intelligence artificielle pour les énergies renouvelables**. Elle montre comment l'introduction d'algorithmes intelligents d'optimisation peut pousser les modèles de deep learning à leur plein potentiel pour modéliser des phénomènes complexes.

Perspectives : Une extension naturelle de ce travail serait de tester d'autres métaheuristiques ou variantes (par exemple un algorithme génétique multi-objectifs pour optimiser à la fois l'erreur et le temps de calcul, ou un algorithme de type *simulated annealing*, voire des approches d'**AutoML** plus complètes). De plus, on pourrait enrichir le modèle en intégrant davantage de données en entrée, comme des images satellites ou des prévisions météorologiques futures, et voir comment l'optimisation d'hyperparamètres s'adapte dans ce contexte à données multiples. Enfin, il serait intéressant d'appliquer la méthodologie à **d'autres sites ou d'autres horizons de prévision** (par ex. prévision horaire sur 48h) pour évaluer la généricité des hyperparamètres optimisés : ceux trouvés pour un bâtiment pourraient-ils servir pour un autre avec peu d'ajustements ? Ce genre de transfert de connaissance est un enjeu pratique important.

En conclusion, ce projet a mis en lumière l'importance de combiner judicieusement expertise métier (connaissance des données solaires), choix algorithmique (modèle ConvLSTM approprié) et outils d'optimisation (métaheuristiques) pour atteindre des performances de pointe en prévision de la production d'énergie renouvelable. Les résultats obtenus encouragent à poursuivre dans cette voie pour fiabiliser toujours plus les prédictions, au service d'une gestion optimisée de l'énergie solaire dans les bâtiments.

Bibliographie

1. **Costa, R.L. de C.** (2022). *Convolutional-LSTM networks and generalization in forecasting of household photovoltaic generation*. **Engineering Applications of Artificial Intelligence**, 116, 105458. (Présente l'utilisation de ConvLSTM pour la prévision d'énergie PV et démontre sa supériorité par rapport à d'autres modèles de deep learning.) ((PDF) Convolutional-LSTM networks and generalization in forecasting of household photovoltaic generation) ()

2. **Kahana, L.** (2025). *PV energy forecasting based on genetic algorithms, dynamic neural network*. **pv magazine International**, 31 Jan 2025. (Article rapportant l'optimisation par algorithme génétique d'un réseau de neurones pour prédire la production PV, avec succès sur une installation en Espagne.) (PV energy forecasting based on genetic algorithms, dynamic neural network – pv magazine International) (PV energy forecasting based on genetic algorithms, dynamic neural network – pv magazine International)

3. **Zeng, X.**, Liang, C., Yang, Q., et al. (2025). *Enhancing stock index prediction: A hybrid LSTM-PSO model for improved forecasting accuracy*. **PLOS ONE**, 20(1):e0268975. (Montre l'efficacité du PSO pour optimiser les hyperparamètres d'un LSTM en prévision de séries financières, avec amélioration de la précision par rapport à d'autres méthodes.) (Enhancing stock index prediction: A hybrid LSTM-PSO model for improved forecasting accuracy - PMC) (Enhancing stock index prediction: A hybrid LSTM-PSO model for improved forecasting accuracy - PMC)

4. **Durrani, A.U.R.**, Minallah, N., Aziz, N., et al. (2023). *Effect of hyper-parameters on the performance of ConvLSTM based deep neural network in crop classification*. **PLOS ONE**, 18(2):e0281311. (Analyse l'impact de divers hyperparamètres sur un ConvLSTM appliqué à des données de classification de cultures, identifiant les configurations optimales et soulignant l'importance du tuning.) (Effect of hyper-parameters on the performance of ConvLSTM based deep neural network in crop classification - PMC)

5. **Akiba, T.**, Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A Next-Generation Hyperparameter Optimization Framework*. In **KDD 2019**. (Outil open-source proposant une optimisation bayésienne des hyperparamètres de manière flexible et efficace, utilisé dans ce projet pour l'optimisation automatique.) (Optuna - A hyperparameter optimization framework)