Département : Génie Electrique

Filière : Master Génie Electrique et Systèmes Industriels Intelligents

Rapport de Projet d'innovation

DIAGNOSTIQUE INTELLIGENT DES SYSTEMES ELECTROMECANIQUES (DISEM)

Réalisé par :

LAAZIZ Youness

KEBE Mamadou

Encadré par :

M. RAFIK Mohamed

Présenté et soutenu le 13 Janvier 2026, devant le jury composé de :

M. RAFIK Mohamed

Mme TALBI Bouchra

Mme SFAR Wissal

Année Universitaire 2025/2026

Liste de figures

Table des matières

Introduction générale

L'industrialisation croissante et l'automatisation des systèmes électromécaniques ont conduit à une utilisation massive des moteurs électriques dans de nombreux domaines tels que l'industrie manufacturière, les systèmes de ventilation, le pompage ou encore les équipements domestiques. Parmi ces moteurs, les moteurs asynchrones et les moteurs à condensateur permanent (PSC) sont largement répandus en raison de leur robustesse, de leur simplicité et de leur faible coût. Toutefois, ces machines restent sujettes à divers défauts mécaniques et électriques pouvant entraîner des pertes de performance, des arrêts non planifiés ou des dommages matériels importants.

Dans ce contexte, la maintenance prédictive constitue une approche moderne visant à anticiper les défaillances avant qu'elles ne provoquent une panne critique. Contrairement aux méthodes de maintenance corrective ou préventive classiques, la maintenance prédictive repose sur l'analyse continue de signaux physiques tels que le courant électrique et les vibrations mécaniques afin de détecter précocement des anomalies de fonctionnement.

Parallèlement, les avancées récentes en intelligence artificielle (IA) et en systèmes embarqués ont permis le développement de solutions d'Edge AI, capables d'exécuter des algorithmes de diagnostic directement sur des microcontrôleurs à faible coût et faible consommation énergétique. Cette approche présente de nombreux avantages, notamment la réduction de la latence, l'autonomie du système et l'absence de dépendance à une infrastructure cloud.

L'objectif principal de ce projet est de concevoir et de valider un système intelligent de diagnostic de défauts moteur basé sur l'analyse conjointe des signaux de courant et de vibration. Le système proposé combine des techniques de traitement du signal, un modèle d'apprentissage automatique entraîné sous Python, et une implémentation embarquée sur un microcontrôleur ESP32-S3. L'approche adoptée vise à garantir un bon compromis entre précision de classification, simplicité de mise en œuvre et compatibilité avec les contraintes matérielles de l'embarquer.

Le travail présenté dans ce mémoire est structuré autour de plusieurs étapes clés : la génération et l'organisation des données, l'extraction de caractéristiques pertinentes, l'entraînement d'un modèle de réseau de neurones fusionné, puis la conversion et le déploiement de ce modèle sur une plateforme embarquée avec validation sur capteurs réels.

CHAPITRE 1 : ETAT DE L'ART

1. Introduction du chapitre

1.1 Objectif du chapitre

Ce chapitre a pour objectif de présenter l'état de l'art relatif au diagnostic intelligent des machines électromécaniques. Il expose les approches classiques et modernes de maintenance, les méthodes de traitement du signal utilisées pour la détection des défaillances, ainsi que les techniques récentes d'intelligence artificielle et d'IA embarquée appliquées à la maintenance prédictive.

Cette étude bibliographique constitue la base scientifique du projet, permettant de situer la contribution proposée — un système de diagnostic hybride combinant FFT, EMD et Deep Learning déployé sur ESP32 — dans le contexte des travaux existants.

1.2 Importance du diagnostic et de la maintenance dans les systèmes industriels

Les systèmes électromécaniques, tels que les moteurs asynchrones, les pompes ou les actionneurs, représentent des éléments essentiels de la chaîne de production industrielle. Toute défaillance de ces équipements peut engendrer des arrêts coûteux, des pertes de productivité et parfois des dommages matériels importants.

Historiquement, deux approches de maintenance dominaient :

- **La maintenance corrective**, qui intervient après une panne,
- **La maintenance préventive**, basée sur un calendrier fixe de révisions.

Cependant, ces stratégies présentent des limites : la première entraîne des arrêts imprévus, tandis que la seconde peut provoquer des interventions inutiles et des coûts excessifs. D'où l'émergence de la **maintenance prédictive**, qui repose sur l'analyse en temps réel des signaux issus des machines pour anticiper les défaillances avant qu'elles ne surviennent.

1.3 Problématique générale

Les moteurs à courant alternative sont exposés à divers types de défaillances — électriques, mécaniques ou combinées — se traduisant par des perturbations dans les signaux de courant, de vibration ou de vitesse. Ces signaux présentent souvent des comportements **non linéaires** et

non stationnaires, rendant difficile leur interprétation par des méthodes classiques. De plus, les solutions de diagnostic existantes exigent souvent des moyens de calcul importants, limitant leur intégration sur des systèmes embarqués à faible puissance.

Face à ces défis, il devient nécessaire de concevoir une solution :

- Capable d'analyser efficacement les signaux issus de capteurs (courant et vibration),
- D'extraire des caractéristiques pertinentes (FFT, EMD),
- Et de les classifier via une architecture d'intelligence artificielle embarquée (DNN sur ESP32).

L'objectif global est donc de proposer un **diagnostic intelligent, temps réel et embarqué**, capable de détecter précocement les défaillances des machines électromécaniques tout en respectant les contraintes de coût et de performance.

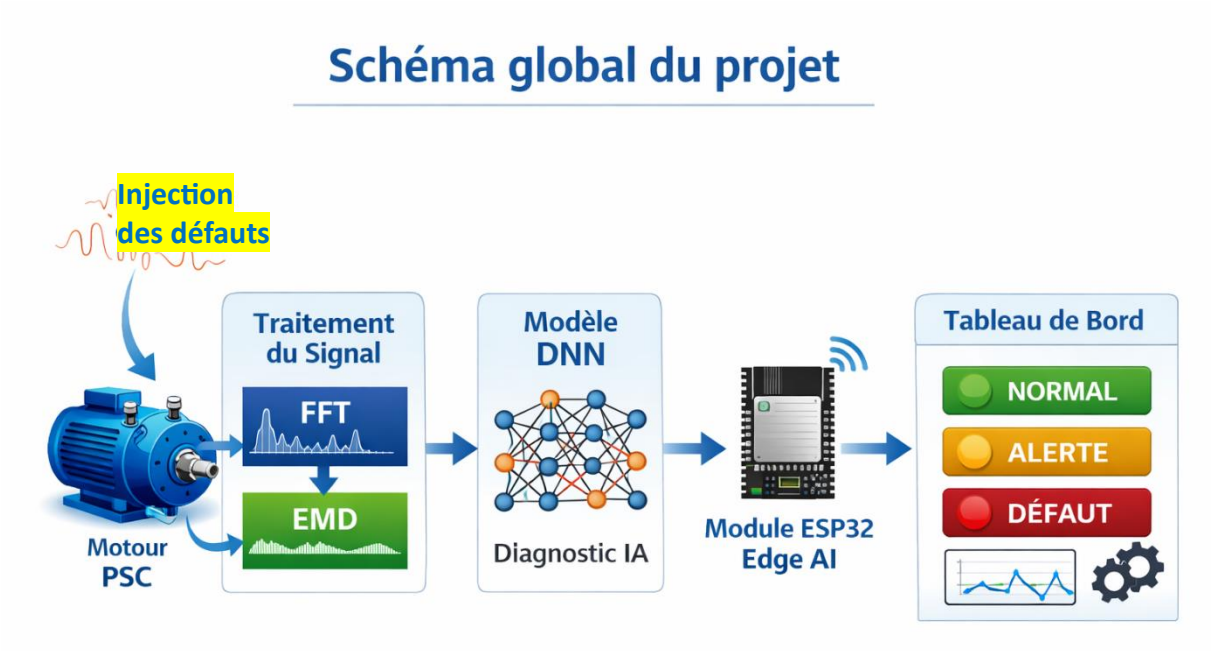

Figure 1 : Schéma global du projet

2. Contexte industriel et enjeux du diagnostic

2.1 L'évolution des pratiques de maintenance industrielle

La maintenance industrielle a connu une évolution significative au cours des dernières décennies, passant de simples interventions réactives à des stratégies de plus en plus

intelligentes et anticipatives.

Autrefois, la maintenance **corrective** consistait à réparer une machine uniquement après l'apparition d'une panne. Bien que cette approche soit facile à mettre en œuvre, elle entraîne souvent des arrêts de production coûteux et imprévisibles.

Pour pallier ces inconvénients, les industriels ont adopté la **maintenance préventive**, qui repose sur des inspections et des remplacements planifiés selon des intervalles de temps ou de fonctionnement. Cependant, cette méthode ne garantit pas que les pièces remplacées soient réellement défectueuses, ce qui peut engendrer des dépenses inutiles et une sous-utilisation des équipements.

Ces limitations ont conduit à l'émergence de la **maintenance prédictive**, une approche proactive basée sur la **surveillance continue de l'état des machines**. Elle vise à anticiper les pannes avant qu'elles ne se produisent, en s'appuyant sur l'analyse de signaux mesurés tels que les courants, les vibrations ou les températures.

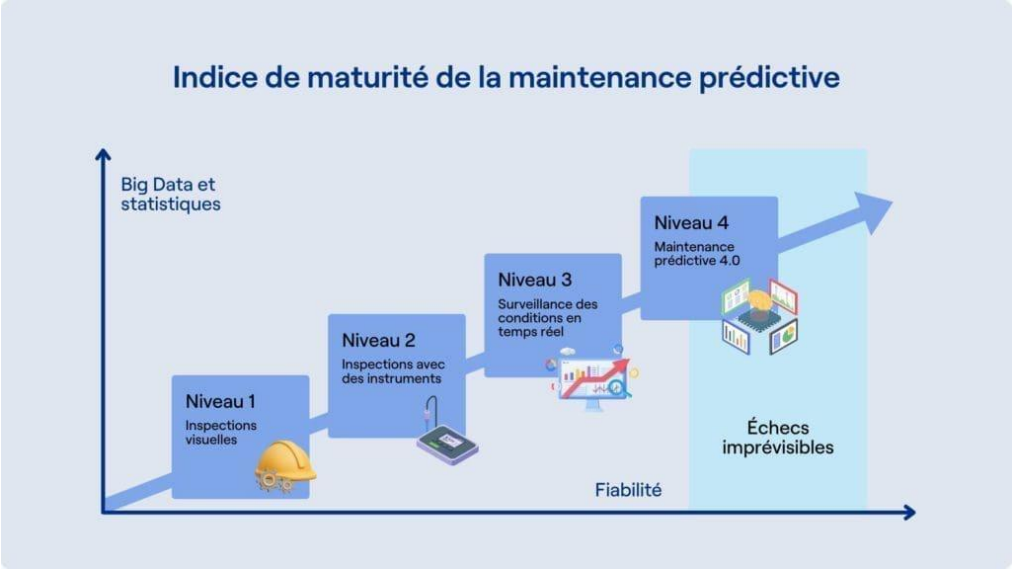

Figure 2: Indice de maturité de la maintenance prédictive

2.2 Vers la maintenance prédictive et intelligente

La maintenance prédictive (ou conditionnelle) représente aujourd'hui un pilier de l'**industrie 4.0**. Elle repose sur le principe selon lequel les machines "parlent" à travers leurs signaux, et qu'une analyse fine de ces signaux peut révéler les premiers signes d'une dégradation mécanique ou électrique.

L'objectif est de **détecter précocement** les anomalies afin d'optimiser la planification des interventions et d'allonger la durée de vie des équipements.

Les progrès récents dans les capteurs, le traitement numérique du signal et l'intelligence artificielle permettent désormais de transformer les données brutes issues des machines en **informations exploitables**.

Cette transition vers la maintenance intelligente s'appuie sur trois éléments clés :

1. **La mesure fiable des signaux** : courant, vibration, vitesse, couple.
2. **Le traitement avancé du signal** : extraction de caractéristiques via FFT, EMD, etc.
3. **L'analyse intelligente** : classification automatique grâce à des modèles d'apprentissage profond (DNN).

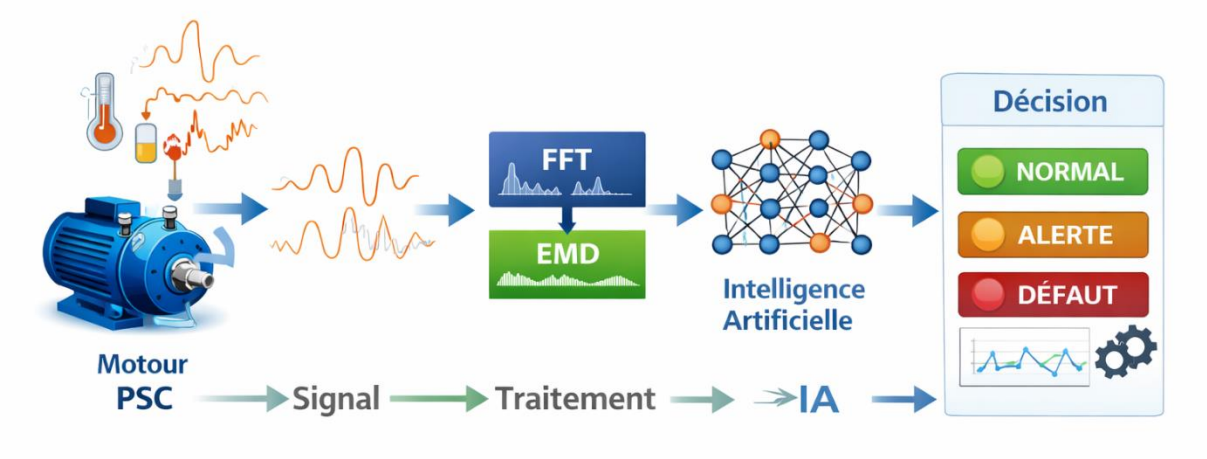

Figure 3 : pipeline du projet

2.3 Les enjeux économiques et technologiques

L'adoption de la maintenance prédictive dans le milieu industriel répond à plusieurs enjeux stratégiques :

- **Réduction des coûts de maintenance** : en intervenant uniquement quand cela est nécessaire, les entreprises évitent les remplacements prématurés et les arrêts coûteux.
- **Amélioration de la disponibilité des machines** : la surveillance continue réduit les arrêts imprévus et augmente la productivité.
- **Optimisation des ressources humaines et logistiques** : planification des interventions sur la base de données réelles plutôt que d'estimations.
- **Durabilité et efficacité énergétique** : en préservant les équipements, on diminue les déchets et la consommation d'énergie liée à la production de pièces de rechange.

D'un point de vue technologique, la maintenance prédictive s'inscrit dans la dynamique de l'**Industrie 4.0**, où les systèmes deviennent interconnectés, intelligents et capables d'auto-diagnostic.
L'intégration de l'**intelligence artificielle embarquée** (Edge AI) sur des plateformes à faible coût, telles que l'ESP32, ouvre la voie à des solutions locales, rapides et accessibles même aux petites structures industrielles.

2.4 Limites et défis actuels

Malgré ses nombreux avantages, la mise en œuvre de la maintenance prédictive présente encore plusieurs défis :

- **Qualité et bruit des données** : les signaux mesurés sont souvent bruités et non stationnaires, ce qui rend leur interprétation complexe.
- **Complexité des modèles physiques** : la modélisation exacte d'un moteur asynchrone reste difficile à cause de ses non-linéarités.
- **Coût computationnel des modèles IA** : les algorithmes de Deep Learning nécessitent une grande puissance de calcul, incompatible avec les microcontrôleurs standards.
- **Manque d'approches hybrides** : peu de travaux combinent simultanément les signaux électriques et vibratoires dans un même diagnostic.

Ces limites justifient le besoin de développer **de nouvelles approches intégrées**, alliant traitement du signal et intelligence artificielle légère, capables d'être déployées sur des systèmes embarqués tout en maintenant une précision élevée.

2.5 Conclusion partielle

Le diagnostic et la maintenance prédictive constituent aujourd'hui des axes essentiels de l'industrie moderne, alliant fiabilité, performance et durabilité. L'évolution vers l'**IA embarquée** et les systèmes de diagnostic autonomes transforme la manière dont les entreprises assurent la surveillance de leurs équipements.
Dans cette optique, la présente étude s'inscrit dans une démarche innovante visant à **développer un système intelligent de diagnostic embarqué**, capable d'exploiter conjointement les

signaux électriques et mécaniques pour détecter précocement les défaillances des moteurs asynchrones.

3. Les systèmes électromécaniques et leurs défaillances

3.1 Structure générale d'un système électromécanique

Les systèmes électromécaniques constituent un élément central des installations industrielles et tertiaires modernes. Ils assurent la conversion de l'énergie électrique en énergie mécanique afin d'entraîner divers équipements tels que les pompes, ventilateurs, compresseurs ou convoyeurs. Parmi ces systèmes, les **machines à moteur PSC (Permanent Split Capacitor)** occupent une place importante, notamment dans les applications à puissance faible ou moyenne, en raison de leur **simplicité de conception**, **robustesse**, **faible coût** et **maintenance réduite**.

Le moteur PSC est une **machine asynchrone monophasée** équipée d'un **condensateur permanent**, destiné à créer un déphasage entre l'enroulement principal et l'enroulement auxiliaire. Ce déphasage permet de générer un champ magnétique tournant quasi circulaire, garantissant un couple de démarrage satisfaisant et un fonctionnement stable.

Un moteur PSC est principalement constitué de :

- **Le stator**, comportant :
  - Un enroulement principal,
  - Un enroulement auxiliaire,
  - Un condensateur permanent assurant le déphasage électrique ;
- **Le rotor**, généralement de type cage d'écureuil, entraîné par induction électromagnétique ;
- **L'arbre mécanique**, assurant la transmission du couple ;
- **Les roulements**, garants de la stabilité mécanique et de la réduction des frottements.

Le fonctionnement global du moteur PSC dépend étroitement :

- De l'équilibre électrique entre les enroulements,
- De l'état du condensateur,
- De l'intégrité mécanique du rotor et des roulements,

- Et de l'alignement de l'arbre moteur.

Toute anomalie dans ces éléments se traduit par des **perturbations mesurables** dans les signaux de **courant**, de **couple électromagnétique**, de **vitesse** et de **vibration**, constituant ainsi une base exploitable pour le diagnostic.

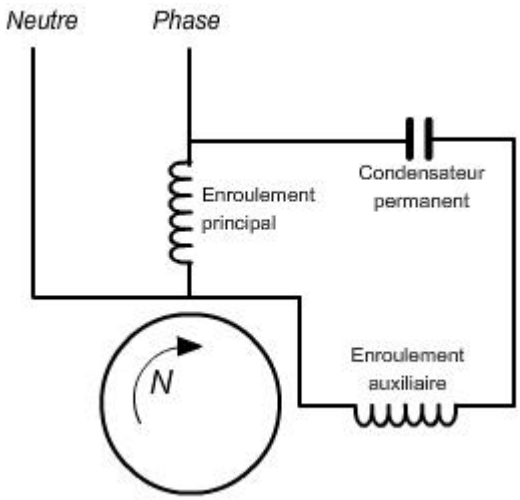

Figure 4 : schéma électrique du moteur PSC

3.2 Typologie des défaillances des machines PSC

Les machines PSC sont exposées à plusieurs types de défaillances, pouvant être classées selon leur origine :

- **Défauts électriques**, affectant :
  - Les enroulements statoriques,
  - Le condensateur permanent,
  - Ou l'équilibre du circuit monophasé ;
- **Défauts mécaniques**, liés :
  - Au désalignement de l'arbre,
  - Aux vibrations excessives,
  - Ou à la dégradation des roulements ;
- **Défauts combinés**, résultant de l'interaction entre phénomènes électriques et mécaniques.

Dans le cadre de ce projet, **sept classes de fonctionnement (C1 à C7)** ont été définies et simulées sous MATLAB afin de couvrir un spectre représentatif des situations réelles rencontrées sur les moteurs PSC.

3.3 Description des classes de fonctionnement (Machine PSC)

Classe C1 — État sain

Le moteur PSC fonctionne dans des conditions normales, sans défaut électrique ni mécanique.

- **Paramètres simulés** :
  - Résistances statoriques nominales,
  - Condensateur en état normal,
  - Couple et vitesse stables,
  - Absence de vibrations anormales.
- **Caractéristiques observées** :
  - Courant monophasé stable et périodique,
  - Spectre FFT propre, dominé par la fondamentale,
  - Signal vibratoire de faible amplitude.
- **Utilité** : Classe de référence pour l'apprentissage supervisé.

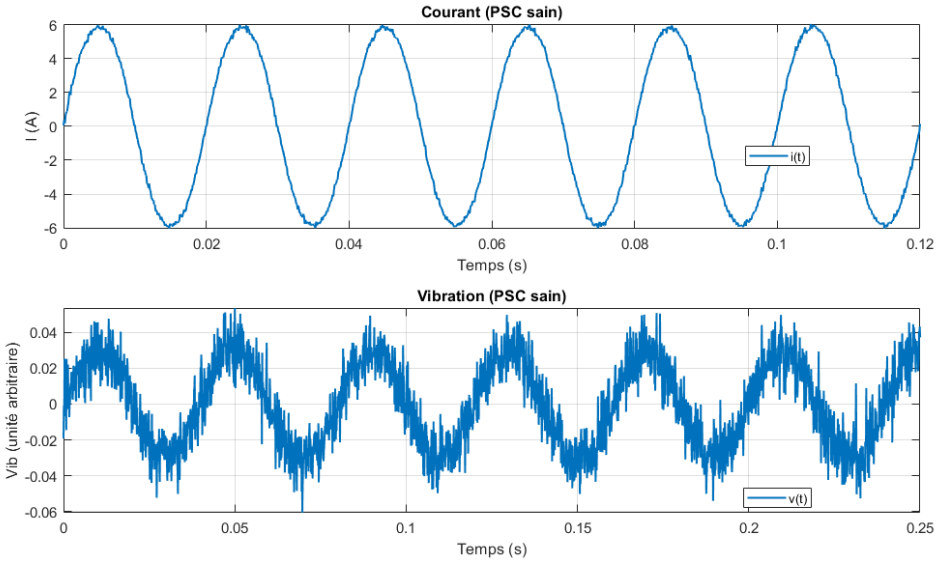

Figure 5 : Signales du courant et vibration (PSC sain)

Classe C2 — Défaut électrique statorique

Un défaut affecte partiellement l'enroulement principal ou auxiliaire, provoquant un déséquilibre électrique.

- **Paramètres simulés** :
    - Résistance statorique réduite de 30 à 70 %.
- **Effets observés** :
    - Augmentation du courant,
    - Distorsion de la forme d'onde,
    - Harmoniques élevées dans le spectre FFT.
- **Signal pertinent** : Courant statorique.

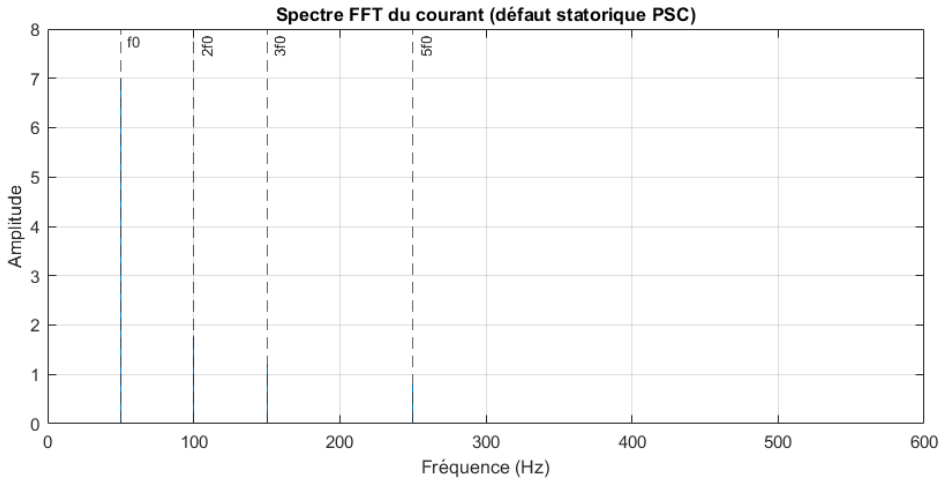

Figure 6 : Spectre FFT du courant (défaut statorique PSC)

Classe C3 — Déséquilibre électrique du circuit PSC

Un déséquilibre électrique global apparaît, souvent lié à une dégradation partielle du condensateur ou des enroulements.

- **Paramètres simulés** :
    - Déséquilibre des paramètres électriques.
- **Effets observés** :
    - Asymétrie du courant,

- Fluctuations du couple,
- Apparition de composantes parasites.

- **Signal pertinent** : Courant statorique.

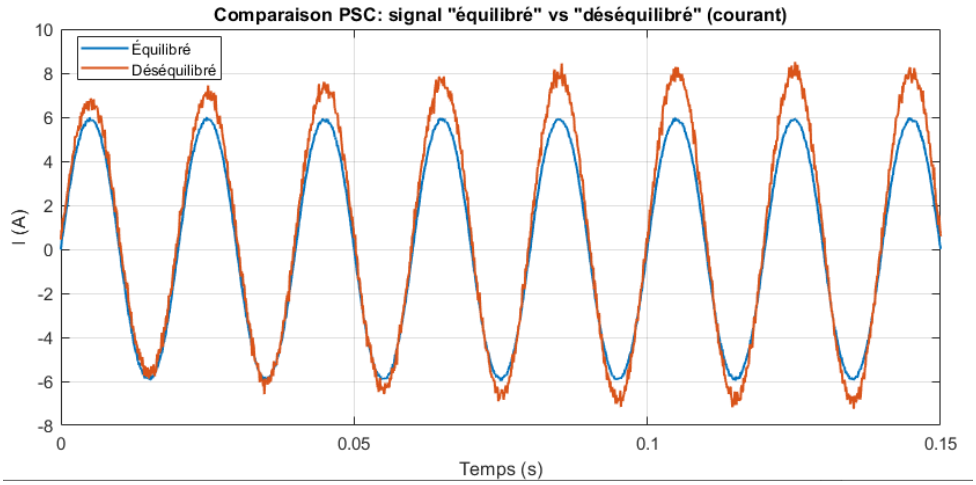

Figure 7 : Comparaison PSC : signal "équilibré" vs "déséquilibré" (courant)

Classe C4 — Défaut rotorique

Une anomalie mécanique interne du rotor modifie la dynamique électromagnétique.

- **Paramètres simulés** :
  - Augmentation de la résistance rotorique.
- **Effets observés** :
  - Oscillations périodiques du couple,
  - Modulation du courant,
  - Perte de rendement.
- **Signaux pertinents** : Courant statorique et couple électromagnétique.

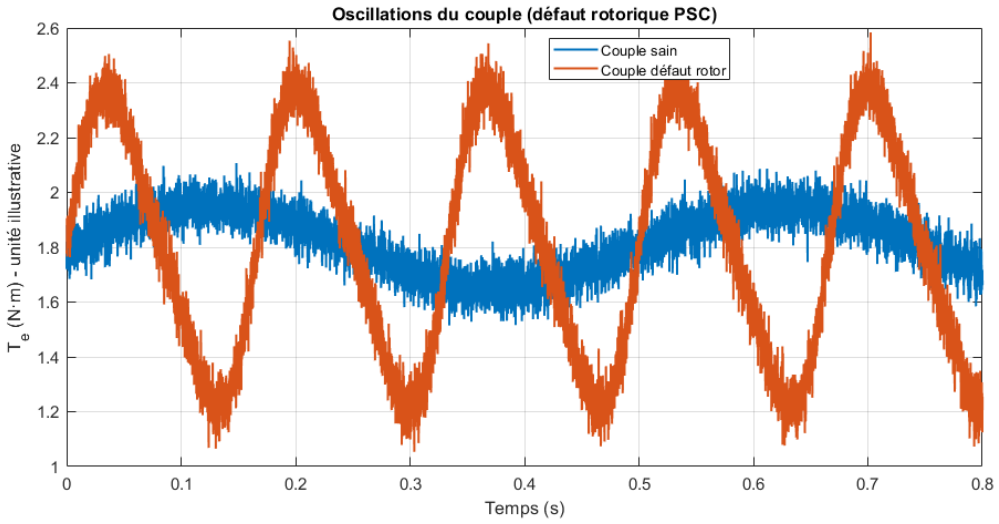

Figure 8 : Oscilations du couple (défaut rotorique PSC)

Classe C5 — Désalignement mécanique

Un désalignement entre le moteur et la charge entraîne une sollicitation mécanique excessive.

- **Paramètres simulés** :
  - Augmentation du gain associé à la vitesse mécanique.
- **Effets observés** :
  - Vibrations basses fréquences dominantes,
  - Augmentation de l'amplitude vibratoire.
- **Signal pertinent** : Signal de vibration.

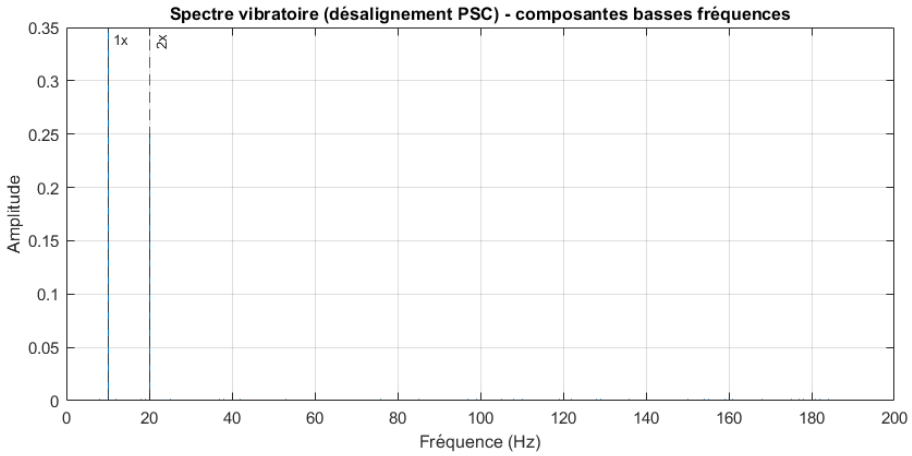

Figure 9 : Spectre vibratoire (désalignement PSC)-composantes basses fréquences

Classe C6 — Défaut de roulement

La dégradation des roulements génère des impacts mécaniques répétitifs.

- **Paramètres simulés** :
    - Augmentation des composantes haute fréquence liées au couple.
- **Effets observés** :
    - Vibrations impulsionnelles,
    - Harmoniques haute fréquence,
    - Augmentation de la kurtosis.
- **Signal pertinent** : Signal de vibration.

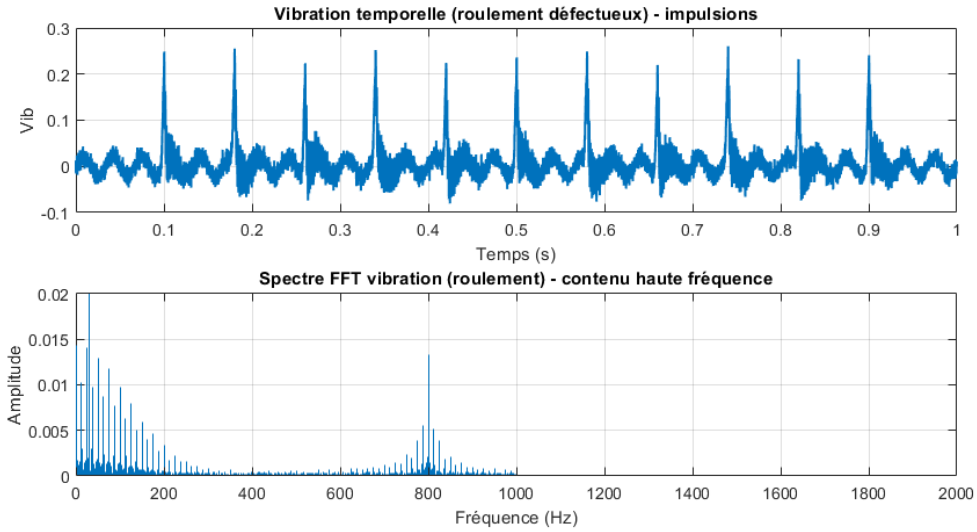

Figure 10 : Vibration temporelle (roulement défectueux) -impulsions

Classe C7 — Défauts combinés

Plusieurs défauts apparaissent simultanément (électrique + mécanique).

- **Paramètres simulés** :
    - Déséquilibre statorique + défaut rotorique ou roulement.
- **Effets observés** :
    - Distorsion complexe du courant,
    - Vibrations sévères,

- Interaction non linéaire des signatures.

- **Signaux pertinents** : Courant statorique et vibration.

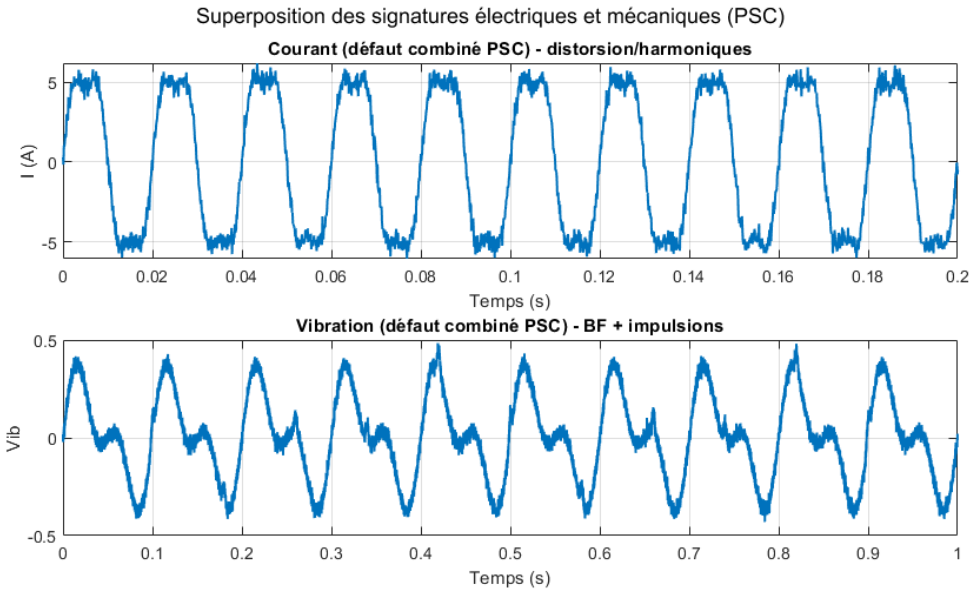

Figure 11 : Superposition des signatures électriques et mécaniques (PSC)

3.4 Analyse comparative des signatures observées

Classe	Type de défaut	Paramètre modifié	Signal principal	Indicateur caractéristique
C1	Sain	—	Courant, Vibration	Signal régulier
C2	Défaut statorique	Rs ↓	Courant	Harmoniques FFT
C3	Déséquilibre PSC	Paramètres électriques	Courant	Asymétrie
C4	Défaut rotorique	Rr ↑	Courant, Couple	Oscillations
C5	Désalignement	Gain vitesse ↑	Vibration	BF élevées
C6	Roulement	HF ↑	Vibration	Impulsions HF
C7	Combiné	Rs ↓ + Rr ↑	Courant + Vibration	Signature mixte

3.5 Conclusion partielle

Cette classification des défaillances constitue la base de la génération du jeu de données utilisé dans ce projet.

Chaque classe représente un état spécifique du moteur PSC, simulé sous MATLAB, permettant d'obtenir des signaux électriques et mécaniques réalistes.

Ces données servent ensuite à l'extraction de caractéristiques par **FFT et EMD**, avant leur exploitation par des **réseaux de neurones profonds**, en vue d'un **diagnostic embarqué sur ESP32**.

4. Méthodes de surveillance et d'analyse de signaux

4.1 Importance de l'analyse des signaux dans le diagnostic

Le diagnostic des machines électromécaniques repose en grande partie sur l'interprétation des signaux issus de capteurs tels que les **courants statoriques**, les **vibrations** ou le **couple électromagnétique**.

Ces signaux contiennent des informations essentielles sur l'état de santé de la machine, mais leur nature souvent **non linéaire** et **non stationnaire** rend leur traitement complexe. Ainsi, l'analyse de signaux constitue une étape fondamentale pour extraire des **indicateurs de défaillance** pertinents avant toute classification par intelligence artificielle.

Selon la nature du signal et du défaut, plusieurs approches peuvent être utilisées : les **méthodes classiques** (basées sur des outils de transformée de Fourier) et les **méthodes avancées adaptatives**, telles que la décomposition empirique en modes (EMD).

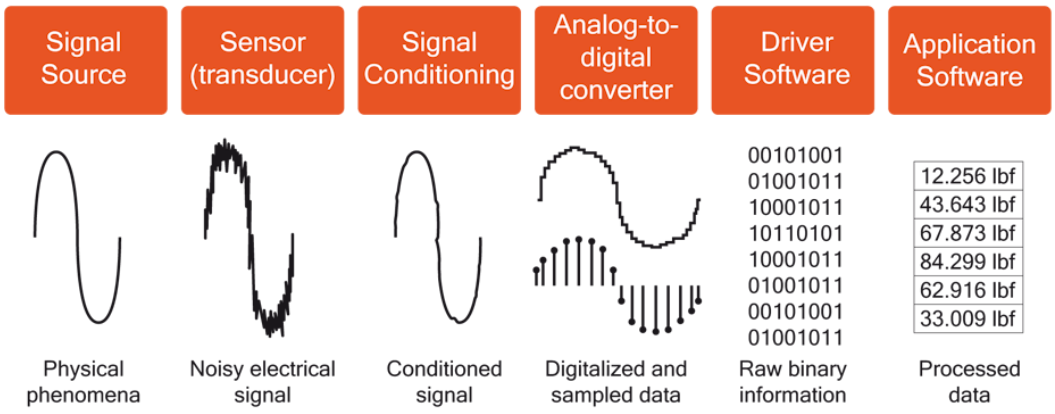

Figure 12 : Analyse des signaux

4.2 Méthodes classiques de traitement du signal

4.2.1 Analyse fréquentielle par la Transformée de Fourier (FFT)

La **Transformée de Fourier (FFT)** est l'un des outils les plus utilisés pour le diagnostic des moteurs asynchrones. Elle permet de représenter le contenu fréquentiel d'un signal temporel et d'identifier les fréquences caractéristiques associées à différents défauts.

L'équation générale de la transformée de Fourier discrète est donnée par :

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \quad k = 0,1, \dots \dots, N-1$$

$x(n)$ est le signal temporel discret,

$X(k)$ est la représentation fréquetielle complexe du signal

L'analyse spectrale des courants statoriques, appelée **Motor Current Signature Analysis (MCSA)**, permet d'identifier :

- Des **bandes latérales (sidebands)** autour de la fréquence fondamentale pour les défauts de rotor,
- Des **harmoniques anormales** pour les déséquilibres ou courts-circuits statoriques,
- Et des **composantes de basse fréquence** en cas de désalignement ou de vibrations.

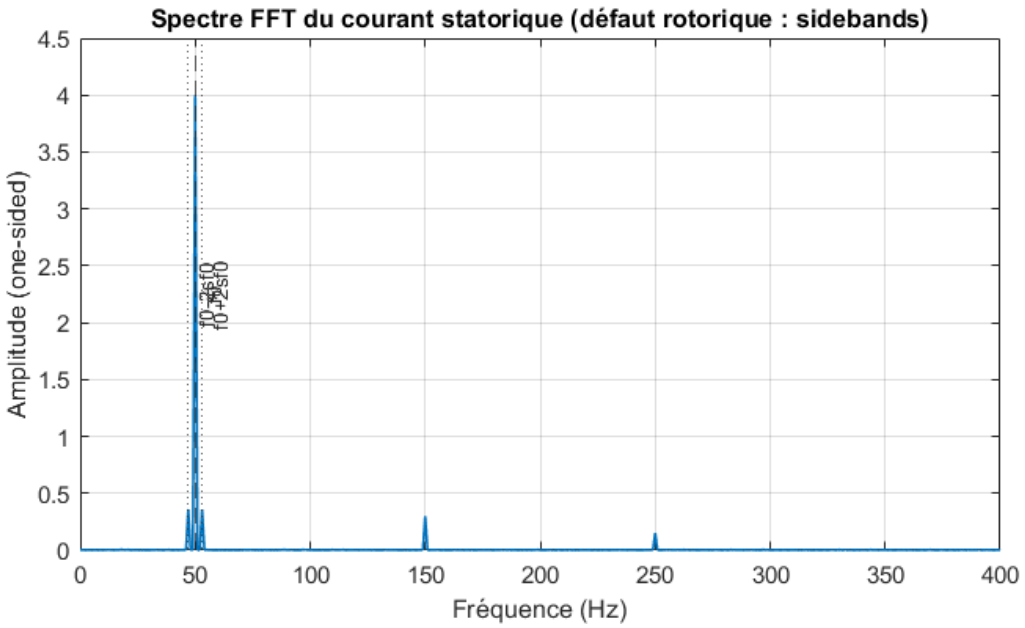

Figure 13 : Spectre FFT du courant statorique (défaut rotorique : sidebands)

Cependant, la FFT repose sur l'hypothèse que le signal est **stationnaire**. Or, les signaux réels des machines (surtout en régime transitoire) ne satisfont pas toujours cette condition.

4.2.2 Méthodes temporelles et statistiques

D'autres approches classiques consistent à calculer des **indicateurs temporels** simples tels que :

- la **valeur RMS** (Root Mean Square),
- Le **facteur de crête**,
- La **kurtosis (aplatissement)**,
- Ou la **skewness (asymétrie)**.

Ces paramètres offrent une première estimation de l'état du moteur mais sont peu sensibles aux variations subtiles et ne permettent pas toujours de distinguer plusieurs types de défauts.

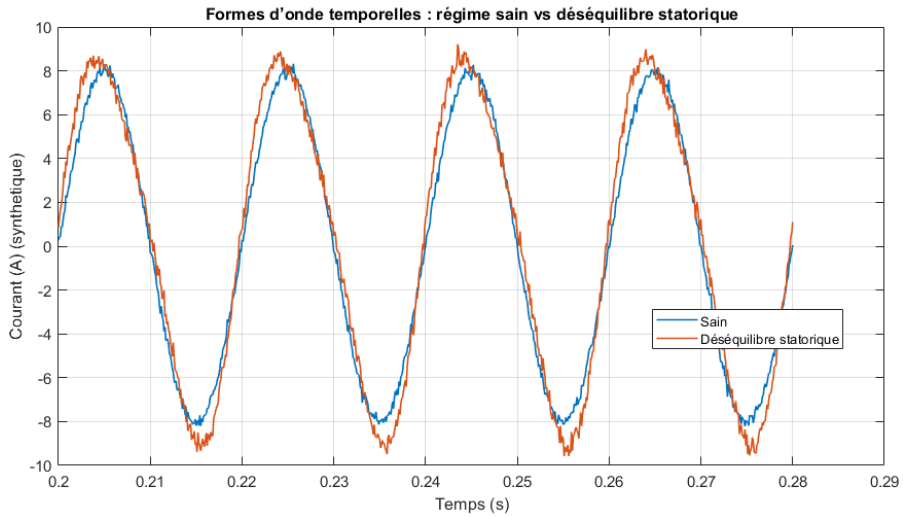

Figure 14 : Formes d'ondes temporelles : régime sain vs déséquilibre statorique

4.3 Limites des méthodes classiques

Les méthodes basées sur la FFT et l'analyse temporelle sont efficaces pour détecter les défauts **stationnaires** ou **périodiques**, mais elles montrent des limites face aux signaux :

- **Non linéaires**, issus d'interactions électromécaniques complexes,
- Ou **non stationnaires**, générés lors de transitoires, de démarrages ou de variations de charge.

Elles ne permettent pas d'analyser simultanément les variations temporelles et fréquentielles du signal.

C'est pour pallier ces limites qu'ont été développées les approches **temps-fréquence adaptatives**, dont la **Décomposition Empirique en Modes (EMD)**.

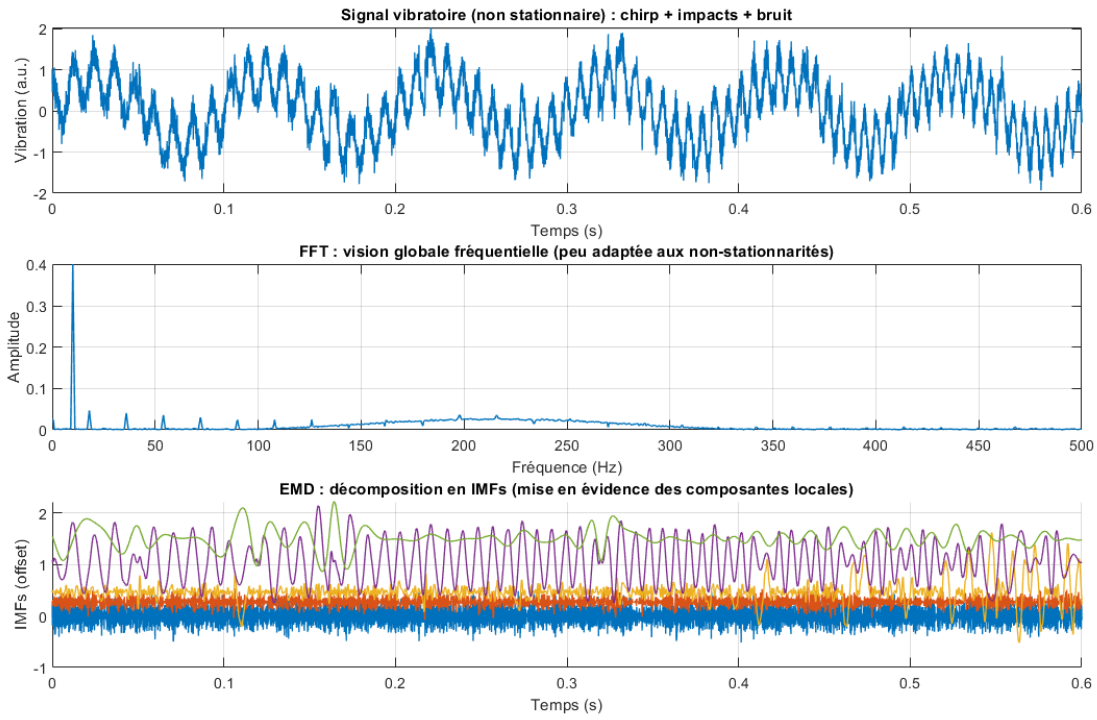

Figure 15 : Signal vibratoire (non stationnaire) : chirp + impacts + bruit

4.4 Méthodes avancées : Décomposition Empirique en Modes (EMD)

4.4.1 Principe général

La **Décomposition Empirique en Modes (EMD)**, introduite par Huang et al. (1998), est une méthode adaptative permettant de décomposer un signal complexe en un ensemble d'oscillations élémentaires appelées **Intrinsic Mode Functions (IMF)**. Chaque IMF représente une composante fréquentielle locale du signal, extraite directement des données sans hypothèse préalable sur leur nature.

L'algorithme EMD procède de manière itérative :

1. Identification des enveloppes supérieures et inférieures du signal.
2. Calcul de leur moyenne locale.
3. Extraction du résidu pour obtenir la première IMF.
4. Répétition du processus jusqu'à ce que le résidu soit monotone.

Ainsi, le signal original $x(t)$ peut être reconstruit comme :

$$x(t) = \sum_{i=1}^{n} IMF_i(t) + r_n(t)$$

Où $IMF_i(t)$ représente les composantes locales et r_n le résidu final.

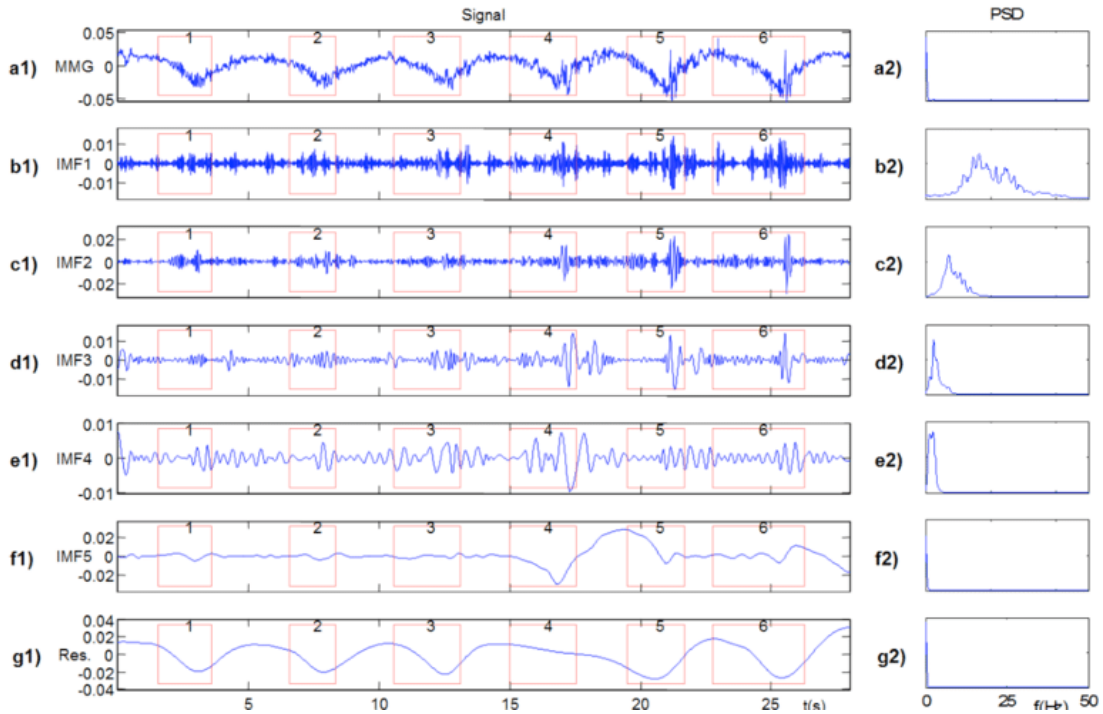

Figure 16 : Décomposition Empirique en Modes (EMD)

4.4.2 Avantages et limites de l'EMD

L'EMD présente plusieurs avantages majeurs :

- Elle est **adaptative** : aucune base de décomposition fixe n'est imposée (contrairement à la FFT ou à la transformée de wavelet).
- Elle est **locale dans le temps**, permettant d'identifier les phénomènes transitoires.
- Elle offre une **interprétation physique intuitive** des composantes obtenues.

Cependant, l'EMD souffre de certaines limitations :

- **Mode mixing** : mélange de fréquences proches dans une même IMF.
- **Sensibilité au bruit** : les signaux bruités peuvent fausser les décompositions.
- **Complexité computationnelle** : l'algorithme est itératif, donc plus coûteux.

Pour pallier ces problèmes, Wu et Huang (2009) ont proposé une version améliorée : **Ensemble EMD (EEMD)**, consistant à ajouter un bruit blanc contrôlé avant la décomposition et à effectuer une moyenne sur plusieurs essais.

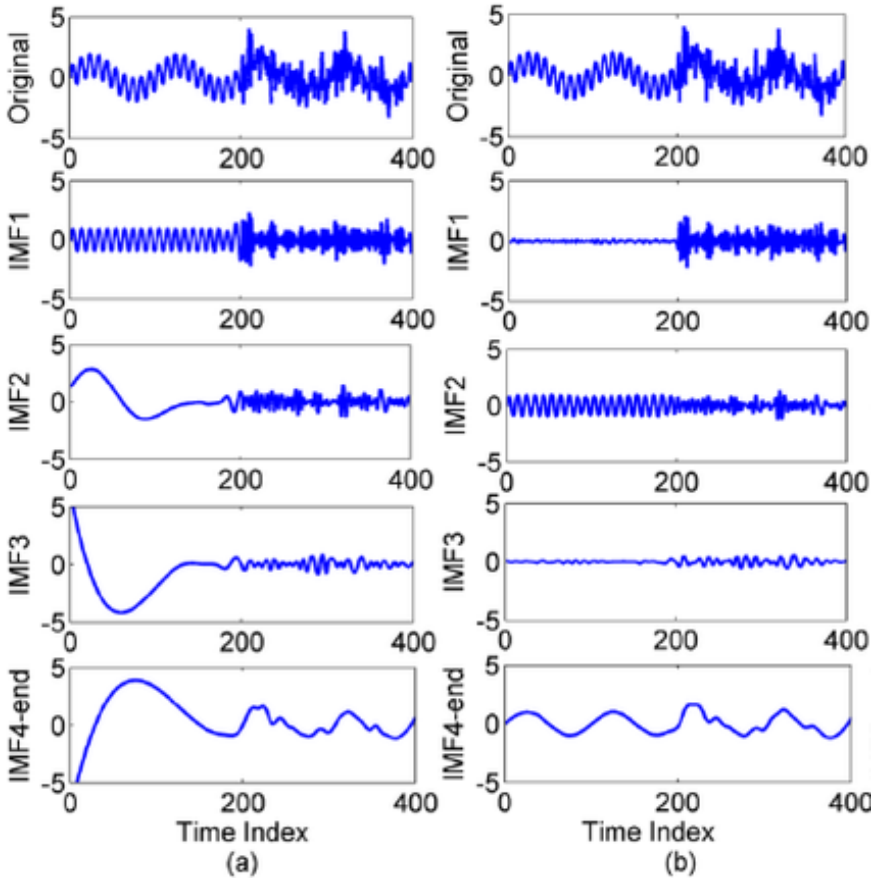

Figure 17 : Une version améliorée : Ensemble EMD (EEMD)

4.5 Application combinée FFT–EMD pour le diagnostic

L'association de la FFT et de l'EMD offre une vision complète des signaux du moteur :

- **FFT** : permet de détecter les composantes fréquentielles principales et les harmoniques globales.
- **EMD** : permet d'analyser les variations locales et les phénomènes non stationnaires.

En combinant ces deux approches, on obtient des **caractéristiques multi-domaines (fréquentiel + temporel)** qui servent d'entrée à un modèle d'intelligence artificielle pour la classification des états du moteur.

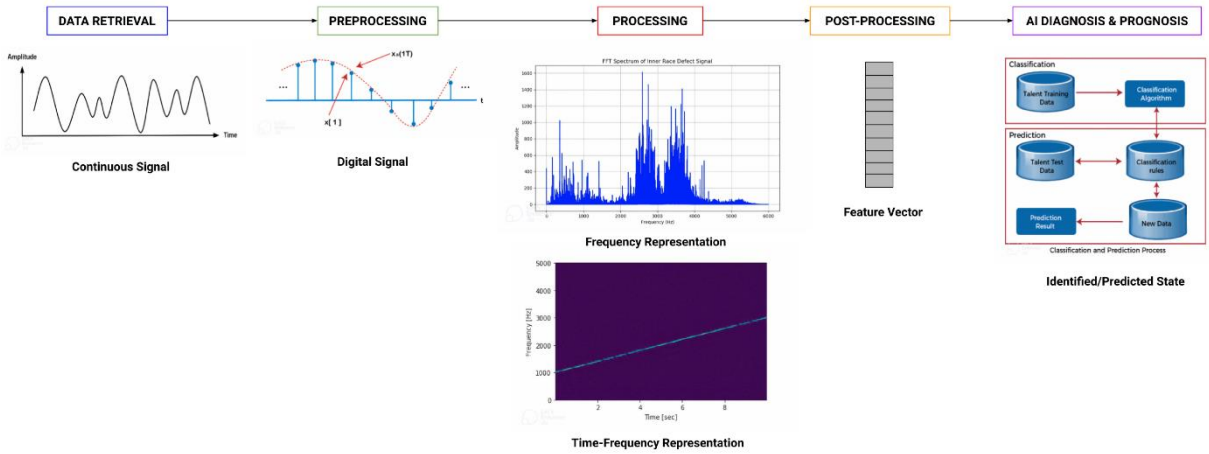

Figure 18 : Application combinée FFT–EMD pour le diagnostic

4.6 Conclusion partielle

Les méthodes de traitement de signaux constituent une étape essentielle pour la maintenance prédictive des systèmes électromécaniques.
Les techniques classiques, comme la FFT, offrent une bonne compréhension globale des phénomènes périodiques, tandis que les approches adaptatives comme l'EMD permettent une analyse fine des signaux transitoires et non stationnaires.
L'exploitation conjointe de ces deux méthodes représente une base solide pour le développement d'un système de diagnostic hybride et intelligent, combinant précision d'analyse et robustesse face aux variations de fonctionnement.

5. L'intelligence artificielle pour le diagnostic des machines

5.1 Vers un diagnostic intelligent

Avec l'évolution des capteurs, de la puissance de calcul et des algorithmes, l'**intelligence artificielle (IA)** s'impose comme une approche incontournable dans le diagnostic et la maintenance prédictive.
Alors que les méthodes de traitement de signal (FFT, EMD, etc.) permettent d'extraire des caractéristiques pertinentes, elles ne suffisent pas toujours à **interpréter automatiquement** les comportements complexes d'une machine.

L'IA intervient ici comme une couche d'analyse **capable d'apprendre à reconnaître** les signatures de défauts à partir des données collectées. En combinant la puissance de la modélisation numérique avec l'apprentissage automatique, il devient possible de réaliser des diagnostics rapides, fiables et autonomes, même en conditions réelles d'exploitation.

5.2 Types d'approches d'intelligence artificielle

Les méthodes d'intelligence artificielle appliquées à la maintenance prédictive se répartissent en trois grandes catégories :

(a) Approches basées sur des règles (Expert Systems)

Elles reposent sur des connaissances humaines codées sous forme de règles logiques ("SI condition ALORS action").
Avantage : interprétation simple.
Limite : peu adaptatives et inefficaces face aux systèmes complexes et bruités.

(b) Apprentissage automatique (Machine Learning)

Il s'agit d'algorithmes capables d'apprendre à partir de données historiques, comme :

- **SVM (Support Vector Machine)**,
- **k-NN (k-Nearest Neighbors)**,
- **Random Forest**, etc.
  Ces méthodes ont montré de bons résultats pour la classification de défauts simples, mais nécessitent souvent une **sélection manuelle de caractéristiques**.

(c) Apprentissage profond (Deep Learning)

Les **réseaux de neurones profonds (Deep Neural Networks, DNN)** surpassent les approches classiques grâce à leur capacité à **extraire automatiquement les caractéristiques pertinentes** des signaux d'entrée.
Ils sont particulièrement adaptés à la reconnaissance de formes complexes, ce qui en fait une solution idéale pour les signaux électriques et vibratoires issus des moteurs asynchrones.

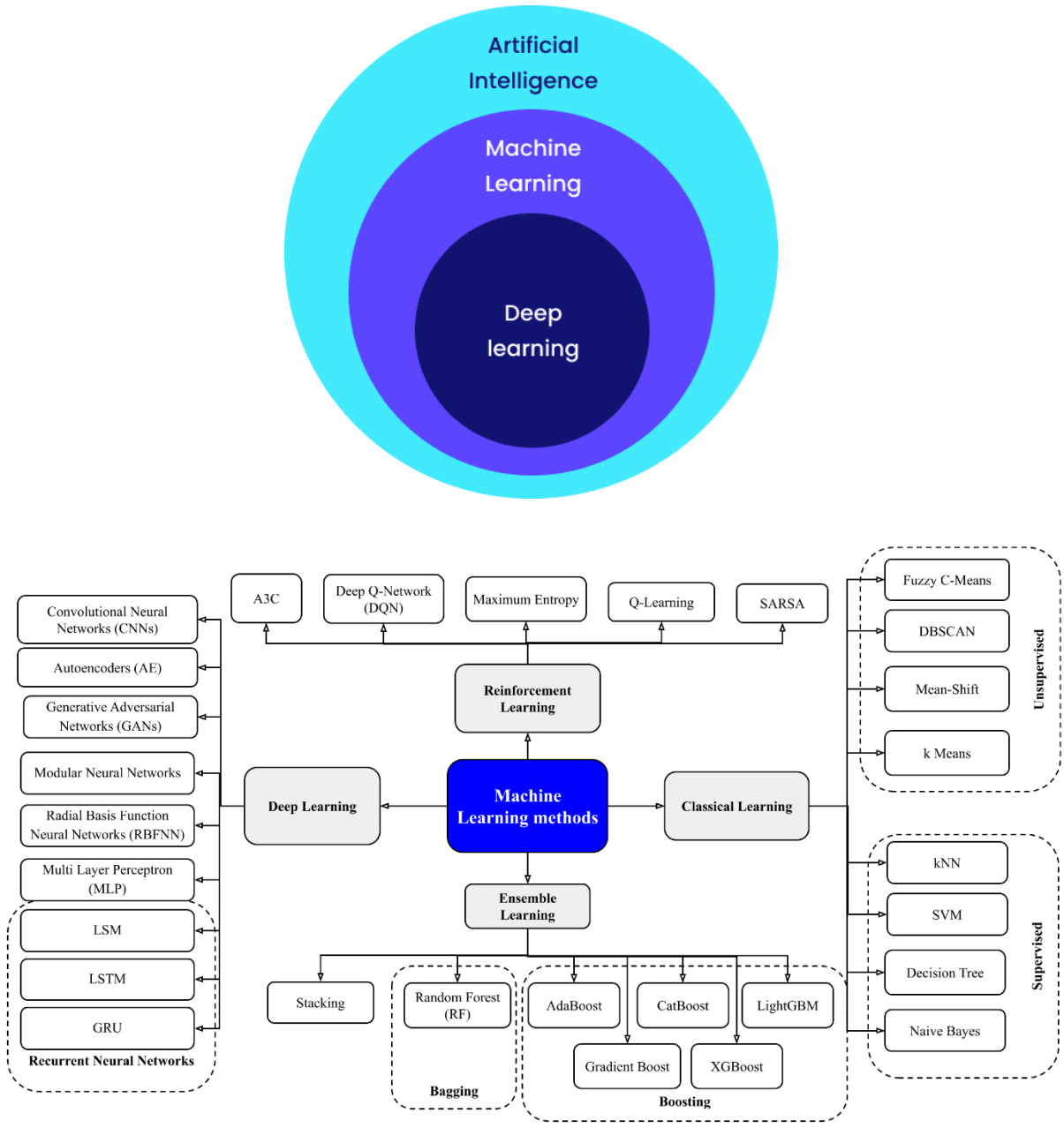

Figure 19 : Types d'approches d'intelligence artificielle

5.3 Principe des réseaux de neurones profonds (DNN)

Les réseaux de neurones profonds sont inspirés du fonctionnement du cerveau humain. Ils se composent de couches de neurones interconnectés :

- **Une couche d'entrée**, qui reçoit les caractéristiques du signal (FFT, EMD, etc.),
- **Plusieurs couches cachées**, qui réalisent des transformations non linéaires,
- **Une couche de sortie**, qui donne la classe prédite (sain, déséquilibre, court-circuit, etc.).

L'entraînement consiste à ajuster les poids des connexions neuronales en minimisant l'erreur entre la prédiction et la réalité, à l'aide d'algorithmes comme la rétropropagation du gradient (Backpropagation).

Les DNN présentent plusieurs avantages clés :

- Capacité à apprendre des **relations complexes** entre variables,
- Robustesse aux signaux bruités,
- Absence de besoin de sélection manuelle des caractéristiques,
- Possibilité de **fusionner plusieurs domaines d'analyse** (ex. FFT + EMD).

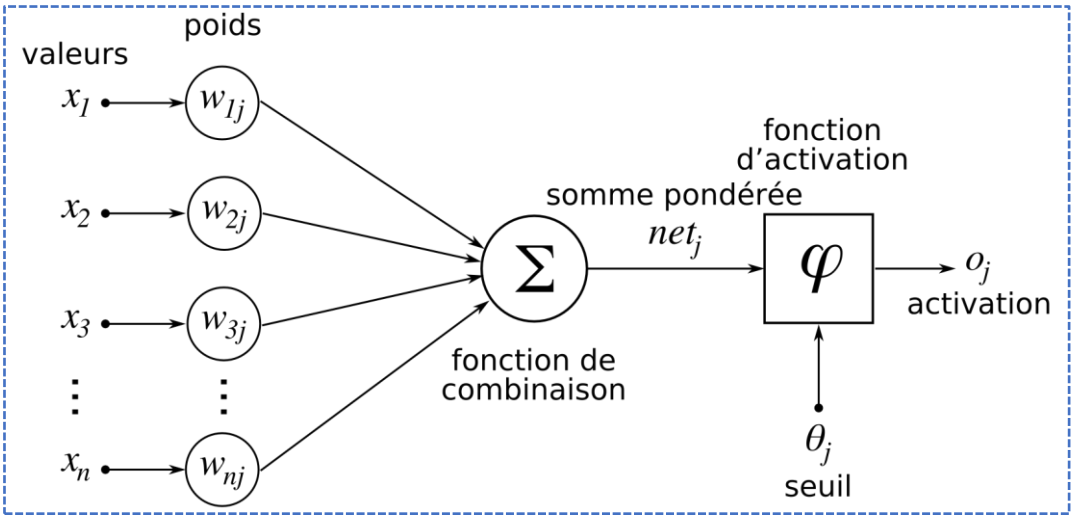

Figure 20 : Principe des réseaux de neurones profonds (DNN)

5.4 Architecture hybride FFT–EMD–DNN

Dans le cadre de ce projet, une approche **hybride** est adoptée :

- Les signaux bruts (courant et vibration) sont d'abord traités par **FFT** et **EMD**,
- Les caractéristiques extraites sont ensuite introduites dans deux réseaux :
  - **DNN_1** pour les caractéristiques FFT,
  - **DNN_2** pour les caractéristiques EMD,
- Les sorties de DNN_1 et DNN_2 sont **fusionnées** dans un troisième réseau **DNN_3**, chargé de la classification finale entre les différentes classes de défaillance (C1 à C7).

Cette architecture en cascade permet d'exploiter la complémentarité entre les approches fréquentielles (FFT) et temps-fréquence (EMD), tout en bénéficiant de la puissance du Deep Learning pour la décision finale.

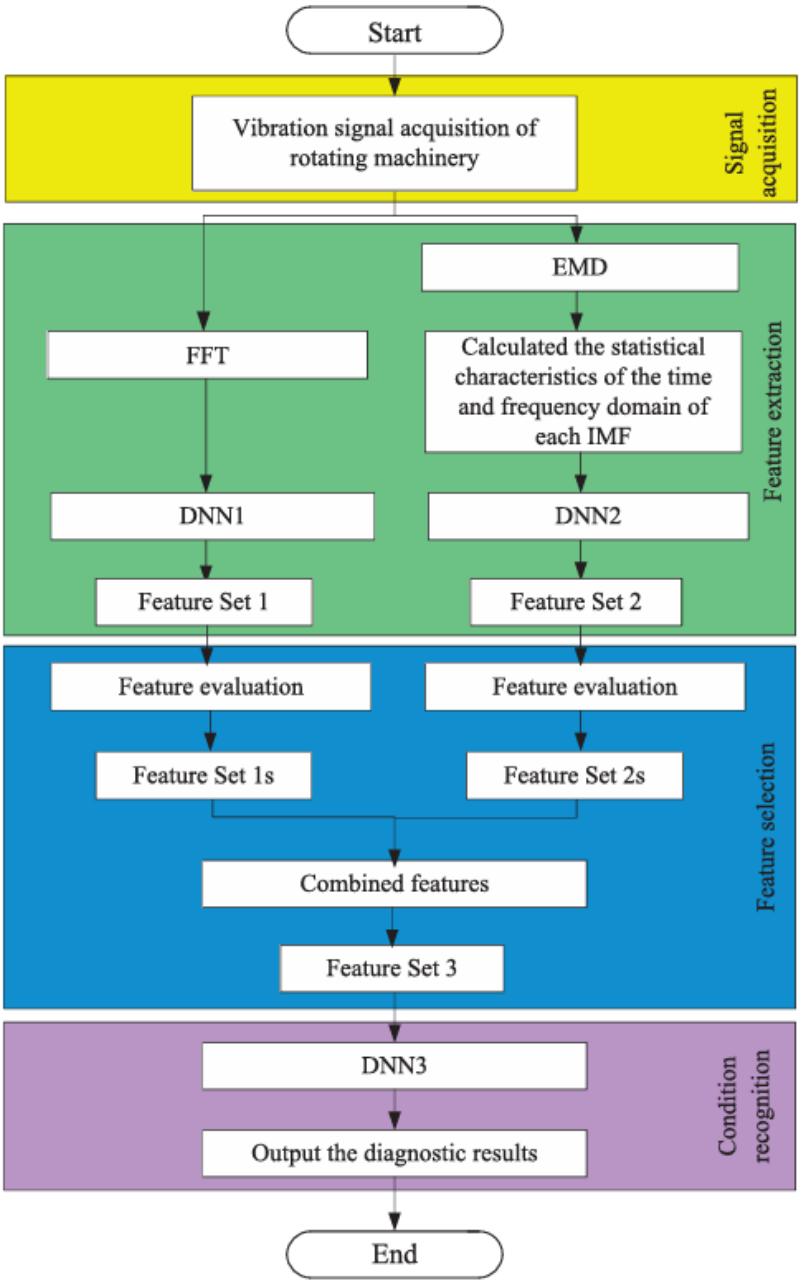

Figure 21 : Architecture hybridee FFT–EMD–DNN

5.5 Entraînement et évaluation du modèle

Le processus d'apprentissage du modèle DNN se déroule en plusieurs étapes :

1. **Préparation du jeu de données** : extraction des signaux simulés sous MATLAB/Simulink pour les sept classes (C1 à C7).

2. **Extraction de caractéristiques** : application de FFT et EMD pour chaque signal.
3. **Normalisation et segmentation** : mise à l'échelle et découpage en lots (batches).
4. **Entraînement supervisé** des modèles DNN_1 et DNN_2 sur les caractéristiques FFT et EMD respectivement.
5. **Fusion** des sorties dans DNN_3 pour obtenir une classification robuste.
6. **Évaluation** à l'aide d'indicateurs de performance : taux de précision, matrice de confusion, F1-score.

Ce schéma garantit une classification précise des défauts, y compris dans les cas combinés où plusieurs anomalies se manifestent simultanément.

5.6 Apport de l'intelligence artificielle dans la maintenance prédictive

L'intégration du Deep Learning dans la maintenance prédictive présente plusieurs bénéfices majeurs :

- Automatisation complète du diagnostic sans expertise manuelle.
- Réduction du temps de détection des anomalies.
- Amélioration de la précision et de la robustesse du diagnostic.
- Capacité à traiter des données multi-capteurs (courant, vibration, température).

En combinant le traitement du signal et l'IA, il devient possible de **transformer un moteur classique en système intelligent**, capable de s'auto-surveiller et de signaler toute anomalie avant qu'elle n'entraîne une panne.

5.7 Conclusion partielle

L'intelligence artificielle constitue aujourd'hui le pilier central du diagnostic industriel moderne.
L'utilisation des **réseaux de neurones profonds (DNN)** permet d'atteindre une précision de détection inégalée, tout en réduisant la dépendance à l'expertise humaine. Combinée aux méthodes FFT et EMD, l'IA offre une approche puissante pour l'analyse conjointe des signaux électriques et vibratoires, préparant ainsi la transition vers des systèmes de maintenance autonomes, intelligents et embarqués.

6. Intelligence Artificielle Embarquée et Edge Computing

6.1 De l'IA de laboratoire à l'IA embarquée

Traditionnellement, les algorithmes d'intelligence artificielle (IA) sont entraînés et testés sur des ordinateurs puissants ou des serveurs dans le cloud. Cependant, dans le contexte industriel, il est souvent nécessaire d'effectuer le **diagnostic directement au niveau de la machine**, sans dépendre d'une connexion internet ni d'un calcul distant.

C'est dans cette optique qu'intervient l'**IA embarquée (Embedded AI)**, une approche consistant à **intégrer les modèles d'apprentissage automatique dans des dispositifs matériels compacts et à faible consommation** comme l'ESP32, les STM32 ou les Raspberry Pi.

L'objectif est de rapprocher l'intelligence artificielle de la source de données, permettant :

- Une **analyse en temps réel**,
- Une **réduction de la latence**,
- Et une **autonomie de fonctionnement**.

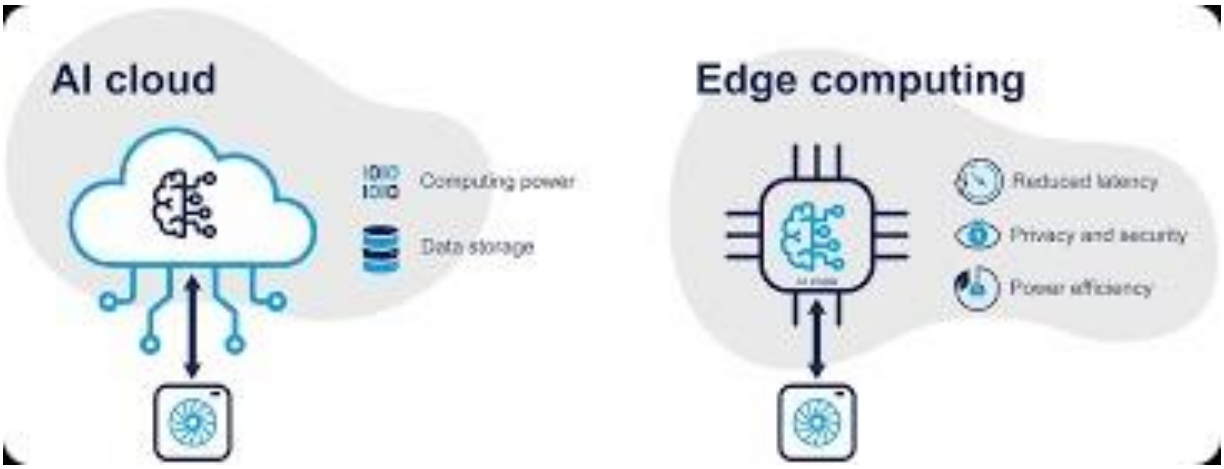

Figure 22 : De l'IA de laboratoire à l'IA embarquée

6.2 Le concept de Edge Computing

Le **Edge Computing** (ou informatique en périphérie) consiste à **déplacer le calcul et le traitement des données au plus près des capteurs**. Dans un environnement industriel, cela signifie que les capteurs et les microcontrôleurs deviennent eux-mêmes capables de prendre des décisions locales.

Avantages principaux :

- **Réactivité accrue :** pas besoin d'envoyer les données vers un serveur distant.
- **Confidentialité renforcée :** les données ne quittent pas le système industriel.
- **Réduction de la bande passante :** seules les décisions ou anomalies détectées sont transmises.

Dans le cadre de ce projet, l'ESP32 joue le rôle de **nœud Edge intelligent** : il acquiert les signaux de courant et de vibration, applique le modèle IA pré-entraîné et décide en temps réel si la machine présente un comportement anormal.

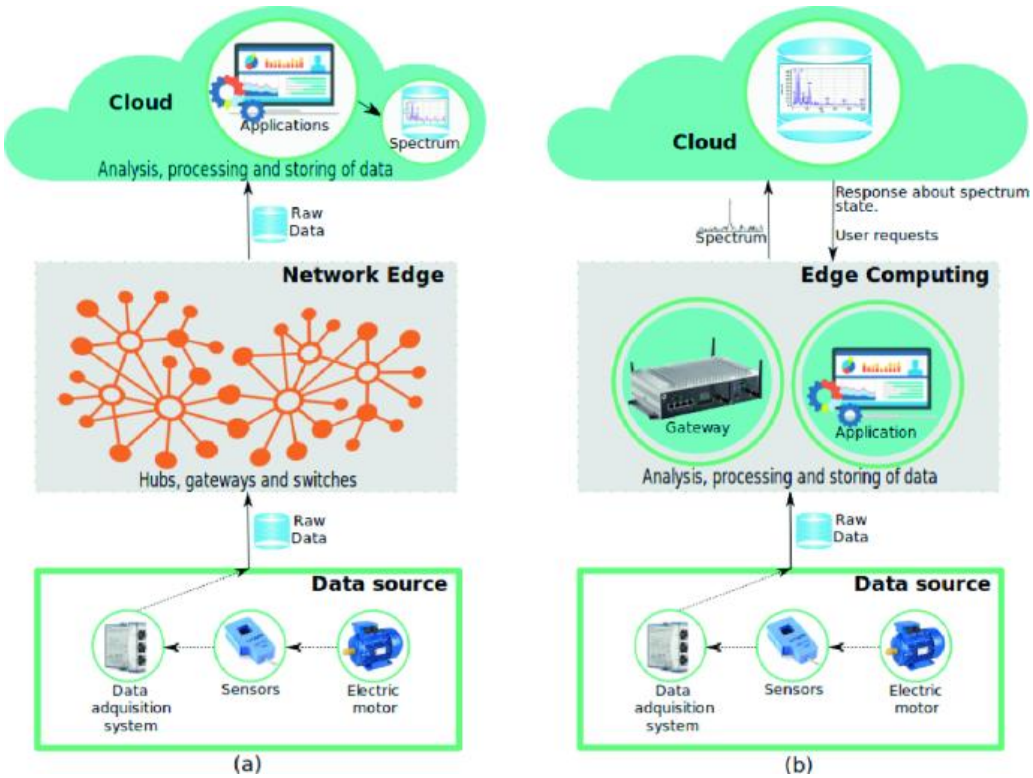

Figure 23 : Le concept de Edge Computing

6.3 Le microcontrôleur ESP32 : cœur de l'IA embarquée

L'ESP32-S3 est un microcontrôleur double cœur, à faible coût et à faible consommation, intégrant :

- Un processeur **Tensilica Xtensa LX7** cadencé jusqu'à **240 MHz**,
- Une connectivité **Wi-Fi** et **Bluetooth** intégrée,
- Des convertisseurs **ADC intégrés** pour l'acquisition de signaux analogiques,
- Un support natif pour **TensorFlow Lite for Microcontrollers (TFLM)**.

Grâce à sa puissance de calcul et à sa mémoire suffisante (notamment avec PSRAM), l'ESP32-S3 est particulièrement adapté aux applications d'**Edge AI**, permettant l'exécution de réseaux de neurones compacts directement sur le microcontrôleur, sans système d'exploitation.

Dans ce projet, l'ESP32-S3 est utilisé pour :

- Acquérir les signaux réels de **courant** et de **vibration** via les capteurs,
- Exécuter localement le **modèle de classification DNN optimisé**,
- Fournir en temps réel la **décision de diagnostic** (affichage ou transmission).

6.4 Conversion et déploiement du modèle d'intelligence artificielle

Une fois le modèle d'intelligence artificielle entraîné sur un ordinateur (via **TensorFlow/Keras**), il doit être converti et optimisé afin de pouvoir être exécuté sur une plateforme embarquée telle que l'ESP32-S3.

Le processus de déploiement se déroule selon les étapes suivantes :

1. **Entraînement du modèle** sur PC à partir des caractéristiques extraites (features) issues des signaux de courant et de vibration.
2. **Conversion et optimisation** du modèle à l'aide de **TensorFlow Lite**, incluant la **quantification en entiers 8 bits (INT8)** afin de réduire la taille mémoire et le temps d'inférence.

3. **Déploiement sur l'ESP32-S3** à l'aide de la bibliothèque **TensorFlow Lite for Microcontrollers**, où le modèle est intégré sous forme de tableau C/C++ et exécuté localement.

Cette approche permet l'exécution du modèle **sans système d'exploitation**, avec une empreinte mémoire réduite (inférieure à quelques centaines de kilo-octets), ce qui est compatible avec les contraintes des systèmes embarqués.

6.5 Contraintes et techniques d'optimisation

L'implémentation de l'IA embarquée impose plusieurs contraintes techniques :

- **Mémoire limitée** : nécessité de réduire la taille du modèle,
- **Puissance de calcul restreinte** : architecture simple et peu profonde,
- **Consommation énergétique** : essentielle pour les systèmes autonomes ou industriels.

Pour répondre à ces contraintes, plusieurs techniques d'optimisation sont employées :

- **Quantization (INT8)** : représentation des poids et activations en entiers 8 bits,
- **Pruning** : suppression de connexions ou neurones peu significatifs (conceptuellement),
- **Knowledge Distillation** : transfert des connaissances d'un modèle complexe vers un modèle plus compact.

Ces optimisations permettent une **exécution en temps réel** (temps d'inférence inférieur à 100 ms) tout en conservant une **précision de classification supérieure à 90 %**, ce qui répond aux exigences du diagnostic industriel.

6.6 Déploiement dans le cadre du projet

Dans ce projet, le flux global de développement et de déploiement est structuré comme suit :

1. **Phase de simulation (MATLAB/Simulink)**
   → Génération et modélisation des signaux de courant et de vibration pour **7 classes de fonctionnement ou de défaillance**.
2. **Phase d'apprentissage (Python)**
   → Extraction de caractéristiques temporelles et fréquentielles → Entraînement d'un **réseau de neurones fusionné (DNN_3)** à partir des signaux courants + vibration.

3. **Phase d'optimisation et d'exportation**
   → Conversion du modèle vers le format **TensorFlow Lite INT8**, validation des performances.

4. **Phase embarquée (ESP32-S3)**
   → Acquisition des signaux réels → Extraction des caractéristiques → Inférence locale → **Détection et diagnostic en temps réel**.

Ce flux met en évidence la complémentarité entre **simulation numérique**, **traitement du signal** et **intelligence artificielle embarquée**, aboutissant à un système complet de **maintenance prédictive intelligente**.

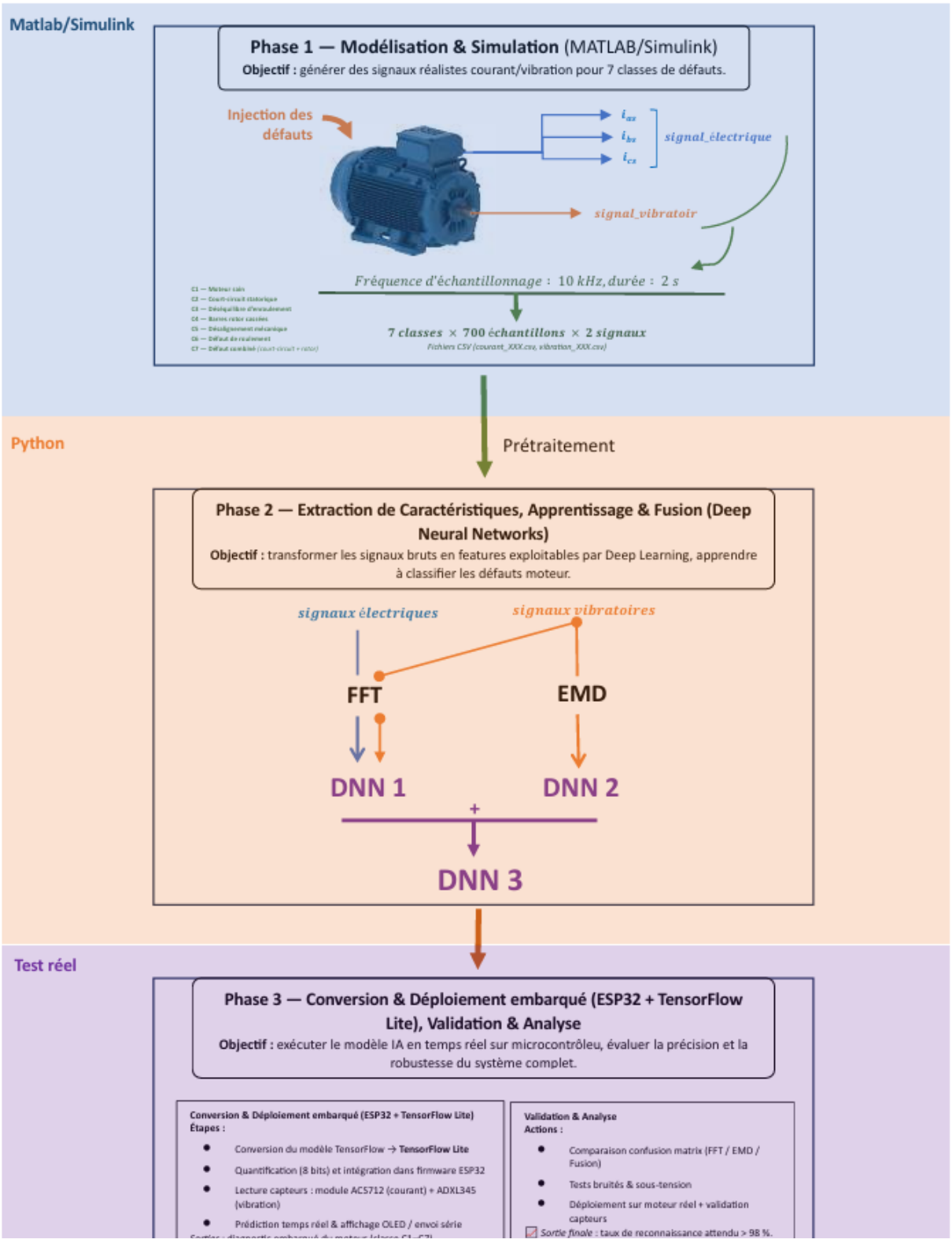

Figure 24 : Déploiement dans le cadre du projet

6.7 Conclusion partielle

L'intégration de l'intelligence artificielle embarquée marque une étape clé vers l'industrie 4.0. Grâce aux plateformes Edge comme l'ESP32, il devient possible de transformer des machines industrielles classiques en systèmes **autonomes, intelligents et connectés**, capables de prévenir leurs propres défaillances.
Ce projet s'inscrit pleinement dans cette vision en combinant des outils logiciels puissants (MATLAB, Python, TensorFlow) et une exécution matérielle légère mais efficace.

7. Conclusion générale du chapitre

Le diagnostic intelligent des machines électromécaniques s'impose aujourd'hui comme un **enjeu stratégique majeur** dans le contexte de l'industrie 4.0.
Les systèmes de production modernes exigent des solutions capables d'assurer à la fois **performance, fiabilité et disponibilité**. Or, les pannes imprévues des moteurs asynchrones restent l'une des principales causes de pertes économiques dans le monde industriel.

Au fil de ce chapitre, plusieurs points clés ont été mis en évidence :

1. **Les systèmes électromécaniques**, notamment les moteurs asynchrones, présentent une grande variété de **défaillances possibles** — électriques, mécaniques ou combinées — dont la détection précoce repose sur l'analyse précise des signaux issus de capteurs.

2. Les **méthodes classiques** de traitement du signal (FFT, indicateurs temporels) permettent de détecter des anomalies stationnaires, mais elles sont limitées pour les signaux **non linéaires** et **non stationnaires**, typiques des régimes transitoires.

3. Les approches **adaptatives**, telles que la **Décomposition Empirique en Modes (EMD)** et ses variantes (EEMD), offrent une meilleure compréhension des phénomènes locaux, permettant une **analyse temps-fréquence plus fine** des signaux de vibration et de courant.

4. L'**intelligence artificielle**, en particulier les **réseaux de neurones profonds (DNN)**, apporte une dimension nouvelle : elle permet **l'apprentissage automatique des motifs caractéristiques** de défaillance sans dépendre d'un modèle physique explicite. L'architecture **hybride FFT–EMD–DNN** constitue une solution robuste pour la

classification multi-défauts, combinant la richesse des signaux et la puissance du Deep Learning.

5. Enfin, l'émergence de l'**IA embarquée** et du **Edge Computing**, rendue possible par des plateformes comme l'**ESP32**, ouvre la voie à des systèmes **intelligents, autonomes et réactifs**, capables d'assurer un diagnostic en temps réel directement au cœur de la machine, sans infrastructure lourde.

Ainsi, le projet proposé s'inscrit dans une approche complète et cohérente :

- **Simulation et modélisation** des signaux de défaut sous MATLAB/Simulink,
- **Extraction et traitement des caractéristiques** via FFT et EMD,
- **Apprentissage et classification** par Deep Learning sur Python,
- **Implémentation embarquée** sur ESP32 pour un diagnostic autonome et prédictif.

Cette démarche illustre parfaitement la convergence entre **génie électromécanique**, **traitement du signal**, **intelligence artificielle** et **systèmes embarqués**, fondement même de l'ingénierie intelligente moderne.

CHAPITRE 2 : Modélisation et génération des données sous MATLAB/Simulink

1 Introduction du chapitre

Objectif du chapitre

Ce chapitre est consacré à la **modélisation, la simulation et la génération des données** utilisées dans le cadre de ce projet. Il décrit la mise en œuvre d'un modèle détaillé de **machine asynchrone monophasée à condensateur permanent (PSC)** développé sous l'environnement **MATLAB/Simulink**, permettant de reproduire fidèlement le comportement dynamique de la machine dans différents états de fonctionnement.

Le modèle proposé permet de simuler aussi bien le **régime sain** que plusieurs **scénarios de défaillances électriques et mécaniques**, représentatifs des conditions réelles d'exploitation industrielle. Ces scénarios sont définis de manière paramétrique afin d'assurer une grande flexibilité et une reproductibilité des simulations.

Afin d'automatiser le processus, des **scripts MATLAB externes** ont été développés pour :

- Configurer les paramètres du modèle selon les différents cas de fonctionnement,
- Piloter les simulations Simulink,
- Extraire les signaux pertinents (courant statorique, vitesse, couple et vibration),
- Et constituer une **base de données structurée sous forme de fichiers CSV**.

Cette base de données constitue le socle de la phase d'apprentissage automatique, qui sera détaillée dans le chapitre suivant. Elle garantit une **cohérence entre la modélisation physique du système et les données exploitées par les algorithmes d'intelligence artificielle**, assurant ainsi la fiabilité des résultats obtenus.

2. Présentation du moteur PSC et choix du modèle

2.1 Le moteur à condensateur permanent (PSC)

Le moteur à condensateur permanent, communément désigné par l'acronyme **PSC (Permanent Split Capacitor)**, est un **moteur asynchrone monophasé** largement répandu dans les applications domestiques et industrielles de faible à moyenne puissance. Il est notamment utilisé dans les **ventilateurs**, **pompes**, **compresseurs**, **systèmes de climatisation (HVAC)** et divers équipements électromécaniques nécessitant une rotation continue et fiable.

Contrairement aux moteurs asynchrones triphasés, le moteur PSC fonctionne à partir d'une **alimentation monophasée**, ce qui le rend particulièrement adapté aux environnements où une

alimentation triphasée n'est pas disponible. Son principe de fonctionnement repose sur la création d'un **champ magnétique tournant artificiel**, obtenu grâce à l'association de deux enroulements statoriques et d'un condensateur permanent.

Structure et principe de fonctionnement

Le moteur PSC est constitué principalement de :

- **Un enroulement principal**, directement alimenté par la source monophasée.
- **Un enroulement auxiliaire**, déphasé par rapport à l'enroulement principal.
- **Un condensateur permanent**, connecté en série avec l'enroulement auxiliaire, permettant de créer un déphasage de courant quasi constant.
- **Un rotor en cage d'écureuil**, similaire à celui des moteurs asynchrones classiques.

Le condensateur permanent introduit un déphasage électrique entre les courants des deux enroulements, ce qui engendre un **champ magnétique tournant** approximatif. Ce champ permet le démarrage et le fonctionnement continu du moteur sans nécessiter de dispositif de démarrage supplémentaire.

Caractéristiques principales du moteur PSC

Le moteur PSC présente plusieurs avantages qui expliquent son large usage industriel :

- **Simplicité de conception** : absence de mécanismes de commutation ou de relais de démarrage.
- **Fonctionnement robuste et fiable**, avec peu de composants sujets à l'usure.
- **Coût réduit**, aussi bien en fabrication qu'en maintenance.
- **Fonctionnement silencieux**, particulièrement apprécié dans les applications domestiques.
- **Bon compromis entre performance et consommation énergétique** pour des charges modérées.

Cependant, comme toute machine électromécanique, le moteur PSC reste sensible à diverses **défaillances électriques et mécaniques**, telles que les déséquilibres d'enroulements, les défauts de roulement ou les problèmes de désalignement. Ces défaillances se traduisent par des perturbations mesurables dans les **signaux de courant, de vitesse, de couple et de vibration**,

ce qui en fait un excellent candidat pour les études de **diagnostic intelligent et de maintenance prédictive**.

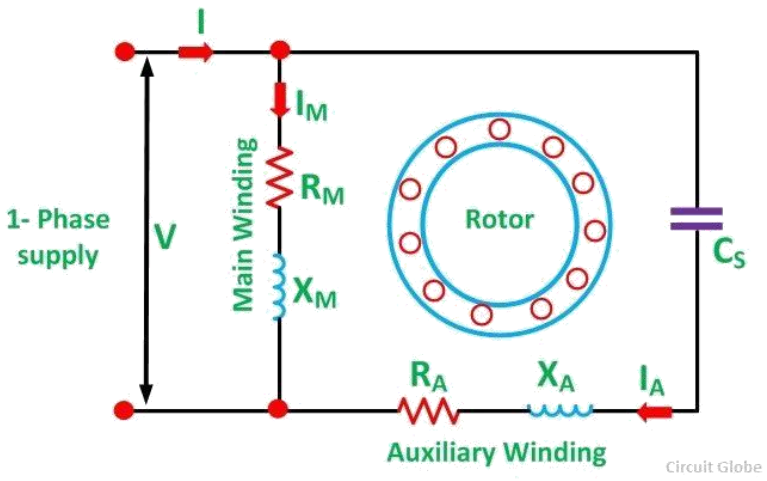

Figure 25 : Le moteur à condensateur permanent (PSC)

2.2 Justification du choix du moteur PSC

Le moteur à condensateur permanent (PSC) a été retenu dans ce projet en raison de sa **forte représentativité industrielle** et de son **adéquation avec les contraintes de l'intelligence artificielle embarquée**. Il constitue un excellent compromis entre **réalisme applicatif**, **simplicité de modélisation** et **compatibilité matérielle** avec des plateformes à ressources limitées telles que l'ESP32.

Tout d'abord, le moteur PSC est **largement répandu** dans de nombreux secteurs industriels et domestiques (ventilation, pompage, climatisation, systèmes HVAC). Cette large diffusion garantit que les résultats obtenus dans ce projet sont **directement transposables à des cas réels**, ce qui renforce la pertinence pratique de l'étude.

Ensuite, bien que monophasé, le moteur PSC présente des **mécanismes de défaillance similaires à ceux des moteurs asynchrones triphasés**, notamment :

- Les défauts électriques liés aux enroulements,
- Les défauts mécaniques (désalignement, roulements),
- Les défauts combinés électromécaniques.

Ces similitudes permettent d'appliquer des **méthodes de diagnostic éprouvées** (FFT, EMD, Deep Learning), tout en travaillant sur un système plus simple et plus accessible expérimentalement.

Enfin, le moteur PSC est particulièrement **compatible avec les approches d'Edge AI**. Les signaux qu'il génère (courant, vibration, vitesse, couple) peuvent être traités par des modèles de taille réduite, adaptés aux contraintes de calcul, de mémoire et de consommation énergétique imposées par les microcontrôleurs embarqués comme l'ESP32. Cela en fait un support idéal pour le déploiement d'un **diagnostic intelligent embarqué en temps réel**, objectif central de ce projet.

Figure 26 : Circuit d'aération

3. Modélisation du moteur PSC sous Simulink

3.1 Architecture générale du modèle Simulink

Le modèle Simulink développé dans ce projet vise à reproduire de manière fidèle le **comportement électromécanique global d'un moteur à condensateur permanent (PSC)**. Il permet de simuler aussi bien le fonctionnement en régime sain que différents scénarios de défauts électriques et mécaniques, tout en générant des signaux exploitables pour le diagnostic intelligent.

L'architecture du modèle repose sur une **décomposition modulaire**, facilitant la compréhension, la modification des paramètres et l'analyse des résultats. Le système est structuré autour de quatre sous-ensembles principaux :

1. **Le sous-système d'alimentation électrique**,
2. **Le sous-système électromagnétique du moteur**,
3. **Le sous-système mécanique du rotor**,
4. **Le sous-système de mesure et de sortie des signaux**.

Cette organisation permet d'isoler clairement les phénomènes électriques et mécaniques, tout en conservant leur interaction dynamique.

3.1.1 Sous-système d'alimentation électrique

Le moteur PSC est alimenté par une **source monophasée sinusoïdale**, modélisée par une tension de la forme :

$$v(t) = V_m \sin(\omega t)$$

Où :

- V_m Est l'amplitude de la tension d'alimentation,
- $\omega = 2\pi f$ Est la pulsation électrique.

L'enroulement auxiliaire est associé à un **condensateur permanent**, dont le rôle est de créer un **déphasage du courant** par rapport à l'enroulement principal, permettant ainsi la génération d'un champ magnétique tournant.
Ce déphasage est fondamental pour assurer le démarrage et le fonctionnement correct du moteur.

3.1.2 Modélisation électromagnétique du moteur PSC

Le modèle électromagnétique décrit le comportement des **enroulements principal et auxiliaire**, chacun étant caractérisé par une résistance et une inductance propre. Les équations électriques peuvent être exprimées sous la forme :

$$v_p = R_p\, i_p + L_p \frac{di_p}{dt} + e_p$$

$$v_a = R_a\, i_a + L_a \frac{da}{dt} + e_a$$

Où :

- $i_p\ et\ i_a$ Sont les courants des enroulements principal et auxiliaire,
- $R_p, R_a\ , et\ L_p, L_a$, La représentent les résistances et inductances correspondantes,
- $e_p\ et\ e_a$ sont les forces électromotrices induites.

Ces forces électromotrices dépendent directement de la **vitesse de rotation du rotor**, traduisant le couplage entre les domaines électrique et mécanique.

3.1.3 Sous-système mécanique du rotor

La dynamique mécanique du moteur est modélisée à partir de l'équation fondamentale du mouvement rotatif :

$$J\frac{d\omega_r}{dt} = T_e - T_L - B\omega_r$$

Avec :

- J : moment d'inertie du rotor,
- ω_r : vitesse angulaire du rotor,
- T_e : couple électromagnétique,
- T_L : couple de charge,
- B : coefficient de frottement visqueux.

Le couple électromagnétique T_e est généré à partir des courants statoriques et du flux magnétique, ce qui permet de reproduire les effets des défauts rotorique ou mécaniques (désalignement, variations de charge, défauts de roulements).

3.1.4 Sous-système de mesure et signaux de sortie

Le dernier sous-système assure l'extraction des **grandeurs mesurables**, nécessaires au diagnostic :

- **Courant statorique** (enroulement principal),
- **Vitesse de rotation** ω_r,
- **Couple électromagnétique** T_e,
- **Signal de vibration**, synthétisé à partir de la dynamique mécanique (vitesse, couple, perturbations haute fréquence).

Ces signaux sont transmis vers des blocs *To Workspace* afin d'être exploités ultérieurement par des scripts MATLAB pour l'extraction de caractéristiques et la génération de fichiers CSV.

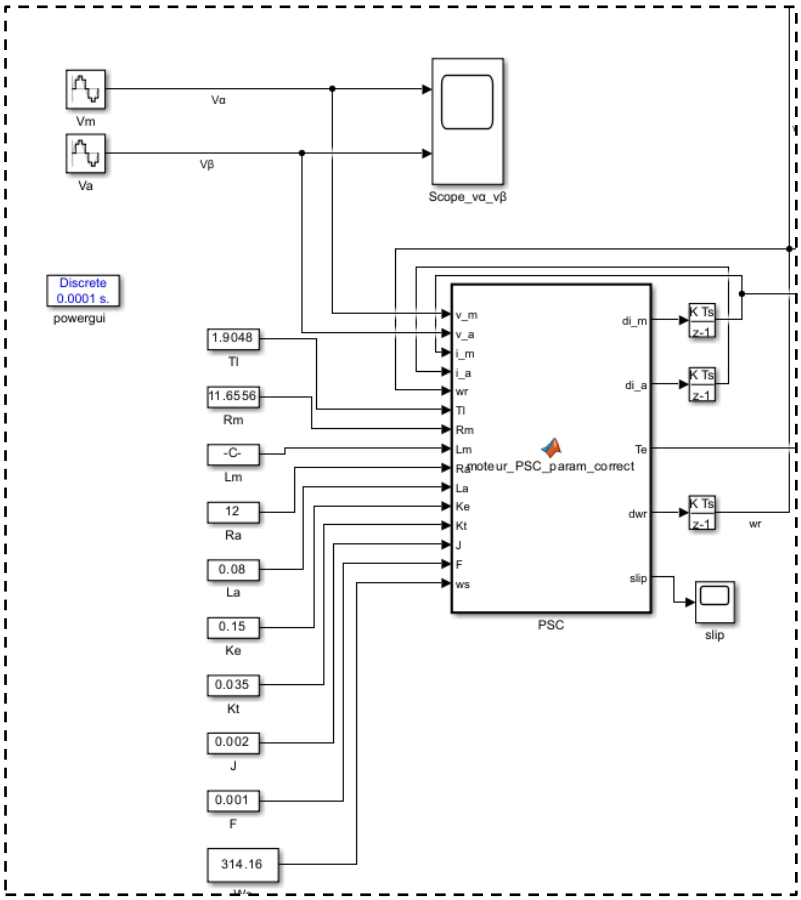

Figure 27 : Modélisation du moteur (PSC)

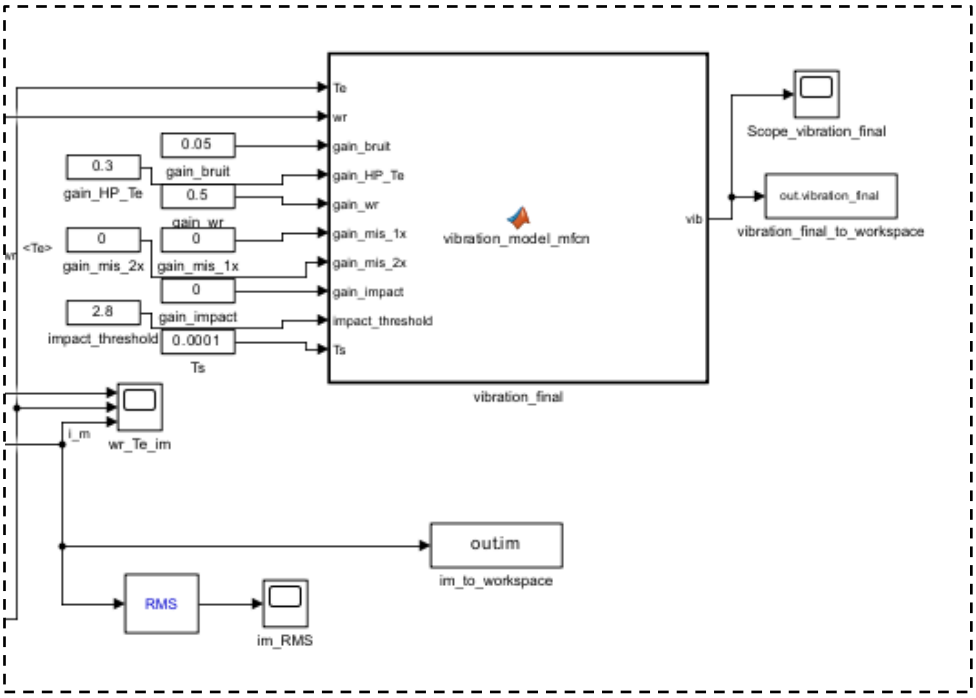

Figure 28 : Modèle de la fonction vibratoire (MATLAB function)

- **Conclusion partielle**

Cette architecture Simulink permet de **reproduire fidèlement les interactions électromécaniques** du moteur PSC tout en offrant une grande flexibilité pour l'introduction de défauts paramétriques. Elle constitue ainsi une base solide pour la génération de jeux de données réalistes destinés à l'apprentissage et à l'évaluation des modèles d'intelligence artificielle présentés dans les chapitres suivants.

4. Génération du signal de vibration (modèle capteur)

4.1 Principe du modèle vibratoire

Dans le cadre de ce projet, le signal de vibration n'est pas issu d'un capteur physique directement intégré dans Simulink. Il est **reconstruit numériquement** à partir de grandeurs mécaniques internes du moteur PSC, principalement le **couple électromagnétique** TeT_eTe et la **vitesse rotorique** ωr\omega_rωr.

Cette approche permet de reproduire de manière réaliste les **signatures vibratoires typiques** observées sur les machines électriques, tout en offrant un contrôle précis des paramètres liés aux défauts mécaniques.

Le modèle vibratoire repose sur l'hypothèse que les vibrations d'un moteur sont le résultat de la **superposition de plusieurs phénomènes physiques**, à savoir :

- Les vibrations de fond dues au fonctionnement normal de la machine,
- Les composantes liées aux variations du couple électromagnétique,
- Les effets mécaniques tels que le désalignement,
- Et les impacts impulsionnels associés aux défauts de roulements.

Le signal de vibration synthétique peut ainsi être exprimé sous la forme générale :

$$v_{vib}(t) = v_{bruit}(t) + v_{HP}(Te) + v_{wr}(\omega r) + v_{impact}(t)$$

Où chaque terme correspond à une source vibratoire spécifique.

4.2 Sources principales du signal de vibration

• Bruit mécanique de fond

Il représente les vibrations naturelles du moteur en fonctionnement normal (jeux mécaniques, frottements, structure).
Ce bruit est modélisé par un signal aléatoire de faible amplitude, généralement assimilé à un bruit blanc gaussien.

• Composantes hautes fréquences liées au couple électromagnétique

Les fluctuations rapides du couple électromagnétique TeT_eTe génèrent des vibrations à haute fréquence, notamment en présence de défauts rotorique ou de variations de charge. Ces composantes sont obtenues par un filtrage passe-haut du couple.

• Effet du désalignement mécanique

Un désalignement entre le moteur et la charge entraîne des vibrations périodiques, souvent dominées par des basses fréquences liées à la vitesse de rotation.
Dans le modèle, ces effets sont directement liés à la dérivée ou aux variations de la vitesse rotorique ωr\omega_rωr.

• Impacts de roulement

Les défauts de roulements se traduisent par des chocs mécaniques répétés, produisant des vibrations impulsionnelles riches en hautes fréquences.
Ces impacts sont modélisés par une excitation impulsionnelle suivie d'une réponse oscillatoire amortie, proche du comportement réel d'un système mécanique résonant.

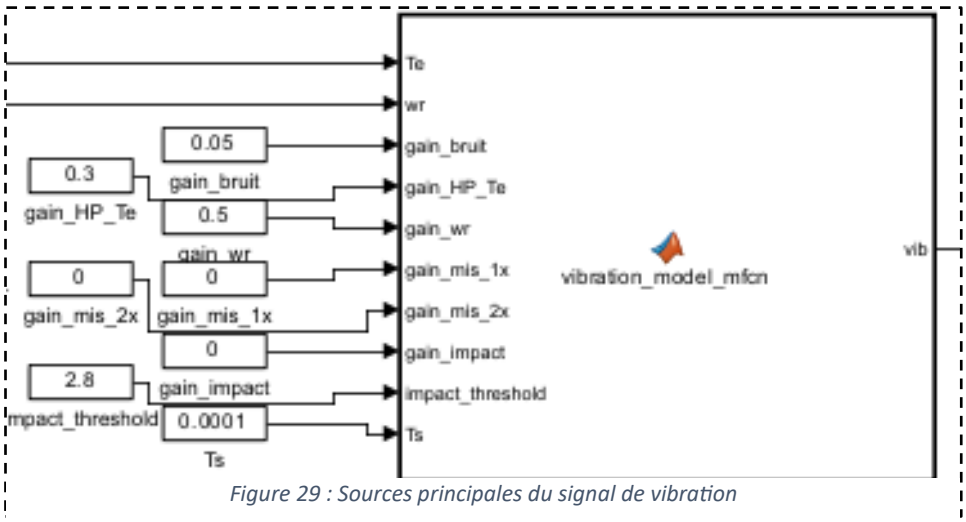

Figure 29 : Sources principales du signal de vibration

4.3 Paramètres vibratoires ajustables

Le modèle vibratoire a été conçu de manière **paramétrable**, afin de simuler différents états de santé du moteur PSC. Chaque paramètre agit sur une composante spécifique du signal de vibration.

Les principaux paramètres ajustables sont :

- **gain_bruit** :
  Contrôle l'amplitude du bruit mécanique de fond.
  → Utilisé principalement pour représenter les variations naturelles ou l'usure générale.
- **gain_HP_Te** :
  Pondère les composantes hautes fréquences issues du couple électromagnétique.
  → Sensible aux défauts rotorique et aux défauts de roulement.
- **gain_wr** :
  Amplifie les composantes vibratoires liées à la vitesse rotorique.
  → Directement associé aux défauts de désalignement mécanique.
- **gain_impact** :
  Détermine l'intensité des impacts mécaniques simulant les défauts de roulements.
- **f0** :
  Fréquence propre du système mécanique excité par les impacts (résonance).

- **tau** :
  Constante de temps d'amortissement, modélisant la dissipation d'énergie mécanique.

4.4 Lien direct avec les classes de défauts (C1 à C7)

Les paramètres vibratoires sont ajustés de manière cohérente avec les différentes classes de fonctionnement :

- **C1 – État sain** :
  Seul le bruit de fond est présent, les autres gains sont quasi nuls.
- **C5 – Désalignement mécanique** :
  Augmentation significative de *gain_wr*.
- **C6 – Défaut de roulement** :
  Valeur élevée de *gain_impact* et de *gain_HP_Te*, générant des vibrations impulsionnelles.
- **C7 – Défauts combinés** :
  Combinaison simultanée de plusieurs paramètres vibratoires, produisant une signature complexe.

Cette paramétrisation permet de générer des signaux vibratoires distinctifs et réalistes, exploitables pour l'apprentissage supervisé des modèles de diagnostic basés sur l'intelligence artificielle.

- **Conclusion partielle**

Le modèle vibratoire proposé constitue une **abstraction réaliste d'un capteur de vibration**, capable de reproduire les signatures mécaniques essentielles associées aux différents défauts du moteur PSC.
Sa flexibilité et sa cohérence physique en font un outil efficace pour la génération de bases de données destinées à l'analyse avancée par FFT, EMD et Deep Learning.

5 Définition des scénarios de défauts (C1 à C7)

5.1 Principe de configuration par script MATLAB

Afin de garantir une génération de données **automatisée, reproductible et structurée**, la configuration des différents scénarios de fonctionnement du moteur PSC n'est pas réalisée manuellement dans l'interface Simulink.
Elle est pilotée dynamiquement à l'aide de **scripts MATLAB externes**, permettant de modifier les paramètres du modèle avant chaque simulation.

Cette approche présente plusieurs avantages majeurs :

- Suppression des erreurs humaines liées à la reconfiguration manuelle,
- Homogénéité des conditions de simulation,
- Facilité d'extension vers un grand nombre d'échantillons,
- Compatibilité directe avec la génération de bases de données pour l'apprentissage automatique.

Le script principal chargé de cette tâche est :

```
% ===== Profils défaut =====
switch classe_id
    case 1  % C1 sain

    case 2  % C2 court-circuit partiel stator (proxy PSC)
        sev = 0.15 + 0.35*rand();
        p.Rm = p0.Rm*(1 + 0.8*sev);
        p.Lm = p0.Lm*(1 - 0.4*sev);
        k_HP = 1.5;

    case 3  % C3 déséquilibre statorique (principal/aux)
        sev = 0.15 + 0.35*rand();
        p.Ra = p0.Ra*(1 + 1.0*sev);
        p.La = p0.La*(1 - 0.3*sev);
        k_HP = 1.3;
        k_wr = 1.2;

    case 4  % C4 rotor (proxy ripple couple)
        k_HP = 1.8;
        k_wr = 1.2;
        p.Kt = p0.Kt*(1 + 0.10*rand());

    case 5  % C5 désalignement -> signature 1x/2x
        k_wr = 1.2;
        mis1 = 40*(0.8 + 0.4*rand());
        mis2 = 15*(0.8 + 0.4*rand());

    case 6  % C6 roulement -> impacts (kurtosis ↑)
        k_bruit = 1.2;

        % Impacts : gain plus fort + seuil plus haut (impacts rares)
        imp = 1500;   % 700..900
        thr = 3.0;   % ~2.9..3.1

    case 7  % C7 combiné
        sev = 0.15 + 0.35*rand();
        p.Rm = p0.Rm*(1 + 0.8*sev);
        p.Lm = p0.Lm*(1 - 0.4*sev);
        k_bruit = 1.4; k_HP = 2.0; k_wr = 1.4;
        mis1 = 20*(0.8 + 0.4*rand());
        imp  = 300*(0.8 + 0.4*rand()); % impacts aussi
        thr  = 3.0;
end
```

Ce script reçoit en entrée l'identifiant de la classe à simuler (C1 à C7) et ajuste automatiquement les paramètres électriques et mécaniques du modèle Simulink correspondant.

Le principe général repose sur une structure conditionnelle de type switch–case, dans laquelle chaque cas correspond à un scénario de défaut précis.

5.2 Description des classes simulées

Les sept classes de fonctionnement définies dans le chapitre précédent sont ici **implémentées numériquement** au sein du modèle Simulink par des **modifications paramétriques ciblées**. Chaque classe est obtenue en ajustant un ensemble cohérent de paramètres électriques, mécaniques et vibratoires, tout en conservant la structure globale du modèle.

Les classes simulées sont les suivantes :

- **C1 — État sain**
  Tous les paramètres sont maintenus à leurs valeurs nominales.
  Aucun déséquilibre électrique ni anomalie mécanique n'est introduit.
  Cette classe sert de référence pour la comparaison avec les autres états.

- **C2 — Défaut statorique (court-circuit partiel)**
  La résistance de l'enroulement principal est réduite sur une phase, provoquant un déséquilibre du courant statorique.
  Ce défaut est implémenté par une modification directe des paramètres résistifs du stator.

- **C3 — Déséquilibre statorique**
  Un déséquilibre résistif plus modéré est appliqué entre les enroulements, sans court-circuit franc.
  Cette configuration permet de générer des asymétries de courant caractéristiques.

- **C4 — Défaut rotorique**
  Une augmentation de la résistance rotorique équivalente est introduite, simulant un défaut de type barre de rotor cassée.
  Cette modification entraîne des oscillations périodiques du couple électromagnétique.

- **C5 — Désalignement mécanique**
  Le défaut est introduit via une amplification des composantes vibratoires liées à la vitesse rotorique.
  Il se traduit par des vibrations dominées par les basses fréquences.

- **C6 — Défaut de roulement**
  Des impacts mécaniques impulsionnels sont ajoutés au signal de vibration, en augmentant le gain des composantes hautes fréquences et des excitations transitoires. Cette classe génère des signatures vibratoires fortement non stationnaires.
- **C7 — Défauts combinés**
  Plusieurs anomalies sont appliquées simultanément (électriques et mécaniques), conduisant à une superposition complexe de signatures dans les signaux de courant et de vibration.

➢ **Implémentation numérique des scénarios**

Chaque scénario est implémenté **sans modifier la topologie du modèle Simulink**, mais uniquement par :

- La variation des paramètres du moteur (résistances, gains),
- L'activation ou l'amplification de certaines composantes vibratoires,
- Et l'ajustement des constantes dynamiques.

Cette stratégie permet de :

- Conserver un modèle unique,
- Faciliter la maintenance du code,
- Et garantir une parfaite traçabilité entre les paramètres de simulation et les classes de données générées.

- **Conclusion partielle**

La définition paramétrique des scénarios de défauts via des scripts MATLAB constitue une étape clé du processus de génération des données.
Elle assure une correspondance claire entre chaque classe de fonctionnement (C1 à C7) et les signatures électriques et mécaniques observées, tout en offrant une grande flexibilité pour la production de bases de données destinées à l'apprentissage automatique.

6. Pilotage automatique des simulations Simulink

La génération d'un volume important de données représentatives nécessite une **exécution automatique et contrôlée** des simulations Simulink. Dans ce projet, le pilotage des simulations n'est pas réalisé manuellement via l'interface graphique, mais assuré par des **scripts MATLAB externes**, garantissant la répétabilité, la rapidité et la cohérence des résultats.

6.1 Script de simulation

```
function out = simulate_PSC_one_matlab(meta, Fs, StopTime)

    p  = meta.params_moteur;
    vb = meta.vibration;

    Ts = 1/Fs;
    t  = (0:Ts:StopTime).';
    N  = numel(t);

    % Etats init
    x = zeros(3,1);   % [im ia wr]
    x(3) = 0;

    im = zeros(N,1);
    ia = zeros(N,1);
    wr = zeros(N,1);
    Te = zeros(N,1);

    % Alim sinusoïdale
    w = 2*pi*50; % 50 Hz
    Vm = meta.Vm_ampl; Va = meta.Va_ampl;
    phm = meta.Vm_phase; pha = meta.Va_phase;

    for k = 1:N
        tk = t(k);

        vm = Vm*sin(w*tk + phm);
        va = Va*sin(w*tk + pha);

        u.vm = vm;
        u.va = va;

        % Calcul couple au temps courant (avec l'état x courant)
        Te(k) = Te_from_state(x, u, p);

        % Intégration RK4 (sauf dernier point)
        if k < N
            x = rk4_step(@(xx) f_psc_dx(xx, u, p), x, Ts);
        end

        im(k) = x(1);
        ia(k) = x(2);
        wr(k) = x(3);
    end

    % Vibration
    vib = vibration_model_matlab(Te, wr, vb, Fs, meta.seed + 999);

    out.t   = t;
    out.im  = im;
    out.ia  = ia;
    out.wr  = wr;
    out.Te  = Te;
    out.vib = vib;
end
```

```
% ============================================================
% Dynamiques moteur: retourne seulement dx = [dim; dia; dwr]
% ============================================================
function dx = f_psc_dx(x, u, p)
    im = x(1); ia = x(2); wr = x(3);

    [dim, dia, ~, dwr] = moteur_PSC_param_correct( ...
        u.vm, u.va, im, ia, wr, p.Tl, ...
        p.Rm, p.Lm, p.Ra, p.La, p.Ke, p.Kt, p.J, p.F, p.ws);

    dx = [dim; dia; dwr];
end

% ============================================================
% Couple Te à partir de l'état courant (sans dériver dx)
% ============================================================
function Te = Te_from_state(x, u, p)
    im = x(1); ia = x(2); wr = x(3);

    [~, ~, Te, ~] = moteur_PSC_param_correct( ...
        u.vm, u.va, im, ia, wr, p.Tl, ...
        p.Rm, p.Lm, p.Ra, p.La, p.Ke, p.Kt, p.J, p.F, p.ws);
end

% ============================================================
% RK4
% ============================================================
function xnext = rk4_step(f, x, h)
    k1 = f(x);
    k2 = f(x + 0.5*h*k1);
    k3 = f(x + 0.5*h*k2);
    k4 = f(x + h*k3);
    xnext = x + (h/6)*(k1 + 2*k2 + 2*k3 + k4);
end
```

Ce script joue un rôle central dans la chaîne de génération des données. Il permet d'exécuter une simulation complète du moteur PSC pour une configuration donnée (classe C1 à C7), tout en contrôlant précisément les paramètres temporels et les sorties récupérées.

➢ **Rôles principaux du script**

Le script simulate_PSC_one_matlab.m assure les fonctions suivantes :

- **Chargement automatique du modèle Simulink**
  Le modèle PSC est chargé dynamiquement à l'aide de la commande load_system, évitant toute interaction manuelle avec l'interface Simulink.
- **Définition du temps de simulation**
  Les paramètres StopTime et FixedStep sont définis dans le script afin de garantir une durée et une fréquence d'échantillonnage constantes pour l'ensemble des simulations.
- **Lancement automatique de la simulation**
  La fonction sim() est utilisée pour exécuter le modèle Simulink directement depuis MATLAB, ce qui permet d'intégrer la simulation dans des boucles de génération de données.
- **Récupération des sorties simulées**
  Les signaux exportés par Simulink sont récupérés à partir de l'objet simOut, facilitant leur traitement ultérieur.

Avantages de cette approche

- Automatisation complète du processus de simulation,
- Possibilité de générer des centaines d'échantillons sans intervention humaine,
- Intégration directe avec les scripts de configuration des défauts et d'export CSV.

6.2 Utilisation des blocs *To Workspace*

Pour assurer un échange efficace entre Simulink et MATLAB, les signaux d'intérêt sont exportés à l'aide de blocs **To Workspace** intégrés directement dans le modèle Simulink.

Ces blocs permettent d'enregistrer les signaux simulés sous forme de variables MATLAB exploitables après la fin de chaque simulation.

Signaux exportés

Les signaux suivants sont systématiquement extraits :

- **t** : vecteur temporel de la simulation

- **im** : courant du moteur PSC (courant statorique principal)
- **Vibration** : signal de vibration reconstruit à partir des grandeurs mécaniques

Chaque signal est configuré pour être exporté au format **Array** ou **Timeseries**, garantissant une compatibilité optimale avec les scripts de post-traitement et d'exportation CSV.

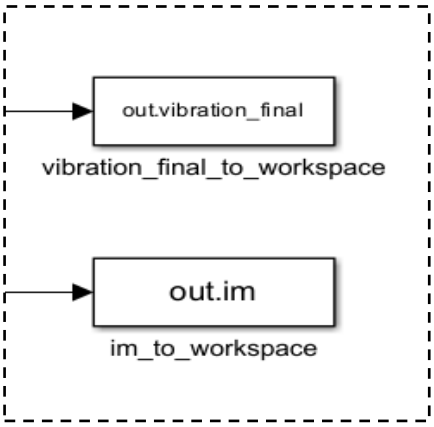

Figure 30 : Sortie de courant et vibration vers to_workspace

➢ **Intégration dans la chaîne de génération des données**

L'association :

- Des scripts MATLAB de pilotage,
- Des blocs *To Workspace*,
- Et de la fonction sim()

Permet de créer une **chaîne de simulation entièrement automatisée**, dans laquelle chaque scénario de défaut est simulé, enregistré et stocké de manière structurée.

Cette architecture constitue le socle de la phase suivante du projet, dédiée à l'extraction des données, à leur organisation sous forme de fichiers CSV, puis à leur exploitation pour l'apprentissage automatique.

- **Conclusion partielle**

Le pilotage automatique des simulations Simulink via MATLAB représente une étape clé pour la génération efficace des données du projet.
Il garantit la reproductibilité des résultats, la cohérence entre les classes simulées et la

compatibilité directe avec les outils d'analyse et d'apprentissage utilisés dans les chapitres suivants.

7. Post-traitement et préparation des données

Une fois les simulations Simulink exécutées et les signaux exportés vers MATLAB, une étape essentielle consiste à **préparer les données brutes** avant leur utilisation pour l'apprentissage automatique.
Cette phase de post-traitement vise à garantir la qualité, la cohérence et la comparabilité des signaux issus des différentes classes de fonctionnement.

Les opérations réalisées comprennent :

- Le nettoyage temporel des signaux,
- Leur mise à l'échelle,
- Et une vérification statistique afin de valider la pertinence physique des données générées.
-

7.1 Nettoyage et découpage temporel

Les signaux exportés depuis Simulink contiennent généralement :

- Une phase transitoire initiale liée au démarrage du moteur,
- Suivie d'un régime quasi-stationnaire plus représentatif du comportement réel de la machine.

Or, pour l'apprentissage automatique, il est préférable de ne conserver que la partie **stable et significative** du signal.

Actions réalisées

Les opérations suivantes sont appliquées systématiquement aux signaux de courant et de vibration :

- **Suppression du régime transitoire**
  Une portion initiale du signal (quelques centaines de millisecondes) est éliminée afin de ne conserver que le régime établi du moteur PSC.

- **Découpage temporel contrôlé**
  Les signaux sont tronqués sur une fenêtre temporelle fixe, garantissant une longueur identique pour tous les échantillons du dataset.

- **Centrage des signaux**
  La valeur moyenne du signal est soustraite afin d'éliminer toute composante continue indésirable :

$$x_{centré}(t) = x(t) - mean(x)$$

- **Normalisation légère**
  Une normalisation basée sur la valeur RMS est appliquée pour homogénéiser les amplitudes tout en conservant les signatures relatives des défauts :

$$x_{norm}(t) = \frac{x(t)}{RMS(x)}$$

➢ **Intérêt de cette étape**

- Réduction de l'influence des conditions initiales,
- Amélioration de la stabilité de l'apprentissage,
- Comparabilité directe entre échantillons et classes.

7.2 Vérification statistique des signaux

Avant la génération définitive des fichiers CSV, une étape de **validation statistique** est réalisée afin de vérifier que chaque classe simulée présente bien des caractéristiques distinctes et physiquement cohérentes.

Cette analyse permet également de détecter d'éventuelles erreurs de configuration ou de simulation.

Indicateurs calculés

Pour chaque échantillon et chaque classe, les indicateurs suivants sont évalués :

- **Valeur RMS (Root Mean Square)**
  Indique le niveau énergétique global du signal.

$$RMS = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x^2(n)}$$

- **Kurtosis (aplatissement)**
  Mesure le caractère impulsionnel du signal, particulièrement pertinent pour les défauts de roulement.

- **Peak-to-Peak (crête à crête)**
  Représente l'amplitude maximale du signal et met en évidence les anomalies sévères.

Ces indicateurs sont calculés séparément pour :

- Le courant moteur,
- Le signal de vibration.

```
function test_1sample_par_classe_PSC()

    clc; close all;
    load_system('DISEM');

    classes = {'C1','C2','C3','C4','C5','C6','C7'};
    StopTime = 2.5;
    t_cut = 1.0;
    zoomT = 0.20;

    % ---- Fenêtre courte pour kurtosis roulement ----
    win_kurt_s = 0.10;   % 100 ms

    % ---- Bande HF optionnelle (roulement) ----
    use_hf_kurt = true;
    hf_band = [300 1200];   % Hz (ajuste si besoin)

    set_param('DISEM','StopTime',num2str(StopTime));

    stats = zeros(7,10);  % +2 colonnes: kurt_short, kurt_hf
    metas = cell(7,1);

    for cid = 1:7
        fprintf('\n=== %s ===\n', classes{cid});
        metas{cid} = configurer_defaut_PSC(cid, 1);

        out = sim('DISEM');

        im_ts  = out.im;
        vib_ts = out.vibration_final;

        % --- Cut transitoire ---
        idx_im  = im_ts.Time  >= t_cut;
        idx_vib = vib_ts.Time >= t_cut;

        t_im  = im_ts.Time(idx_im)  - t_cut;
        x_im  = im_ts.Data(idx_im);
        t_vib = vib_ts.Time(idx_vib) - t_cut;
        x_vib = vib_ts.Data(idx_vib);

        % garantir colonne
        x_im  = x_im(:);
        x_vib = x_vib(:);

        % enlever DC
        x_im  = x_im  - mean(x_im);
        x_vib = x_vib - mean(x_vib);
        % Détection pics sur dérivée (accentue impacts)
        fs = 1/median(diff(t_vib));
        [b,a] = butter(3, [300 1500]/(fs/2), 'bandpass');
        xhf = filtfilt(b,a,x_vib);

        nb_pics = sum(abs(xhf) > 4*std(xhf));  % 4 au lieu de 5
        fprintf('C%d | nb_pics(|HF|>4std) = %d\n', cid, nb_pics);
```

```
    % --- Fs robuste ---
    % (Ton modèle est discret Ts=1e-4, donc mieux de prendre diff médiane)
    fs_im  = 1/median(diff(t_im));
    fs_vib = 1/median(diff(t_vib));

    % ========================
    % Stats "globales"
    % ========================
    rms_im   = rms(x_im);
    std_im   = std(x_im);
    p2p_im   = peak2peak(x_im);

    rms_vib  = rms(x_vib);
    std_vib  = std(x_vib);
    p2p_vib  = peak2peak(x_vib);

    % Kurtosis globale (souvent peu discriminante si impacts rares)
    kurt_vib_global = kurtosis(x_vib);

    % ========================
    % Kurtosis fenêtre courte (RECOMMANDÉE)
    % ========================
    Nw = min(length(x_vib), round(win_kurt_s * fs_vib));
    x_short = x_vib(1:Nw);
    kurt_vib_short = kurtosis(x_short);

    % ========================
    % Kurtosis bande HF (OPTIONNEL mais très utile roulement)
    % ========================
    kurt_vib_hf = NaN;
    if use_hf_kurt
        wn = hf_band / (fs_vib/2);
        wn(1) = max(wn(1), 1e-4);
        wn(2) = min(wn(2), 0.999);

        if wn(2) > wn(1)
            [b,a] = butter(3, wn, 'bandpass');
            x_hf = filtfilt(b,a,x_vib);
            kurt_vib_hf = kurtosis(x_hf);
        end
    end

    % Stockage
    stats(cid,:) = [cid rms_im std_im p2p_im rms_vib std_vib p2p_vib kurt_vib_global kurt_vib_short kurt_vib_hf];

    % ========================
    % Plots temps (zoom)
    % ========================
    figure('Name',['Temps ' classes{cid}],'NumberTitle','off');
    subplot(2,1,1);
    plot(t_im, x_im, 'LineWidth', 2); grid on; xlim([0 zoomT]);
    title(['Courant im (stable) - ' classes{cid}]);

    subplot(2,1,2);
    plot(t_vib, x_vib, 'LineWidth', 2); grid on; xlim([0 zoomT]);
    title(['Vibration (stable) - ' classes{cid}]);
```

```
        % ========================
        % FFT (amplitude simple)
        % ========================
        N1 = length(x_im);
        N2 = length(x_vib);

        % FFT simple (non normalisée) + fenêtre Hann (réduit leakage)
        w1 = hann(N1);
        w2 = hann(N2);

        X1 = abs(fft(x_im .* w1));
        X2 = abs(fft(x_vib .* w2));

        f1 = (0:N1-1)*(fs_im/N1);
        f2 = (0:N2-1)*(fs_vib/N2);

        figure('Name',['FFT ' classes{cid}],'NumberTitle','off');
        subplot(2,1,1);
        plot(f1(1:floor(N1/2)), X1(1:floor(N1/2)), 'LineWidth', 2);
        grid on; xlim([0 1000]);
        title(['FFT courant - ' classes{cid}]);

        subplot(2,1,2);
        plot(f2(1:floor(N2/2)), X2(1:floor(N2/2)), 'LineWidth', 2);
        grid on; xlim([0 2000]);
        title(['FFT vibration - ' classes{cid}]);

        % Petit log utile
        fprintf('Kurtosis vib: global=%.2f | short(%.0fms)=%.2f | HF=%.2f\n', ...
            kurt_vib_global, win_kurt_s*1000, kurt_vib_short, kurt_vib_hf);
    end

    % ========================
    % Résumé final
    % ========================
    fprintf('\n=== Résumé stats (après cut %.1fs, StopTime %.1fs) ===\n', t_cut, StopTime);
    fprintf('Classe | RMS_im  STD_im  P2P_im  | RMS_vib STD_vib P2P_vib | KurtG  Kurt100ms  KurtHF\n');
    for i = 1:7
        fprintf('C%d     | %.4f  %.4f  %.4f | %.4f  %.4f  %.4f | %.2f   %.2f      %.2f\n', ...
            stats(i,1), stats(i,2), stats(i,3), stats(i,4), stats(i,5), stats(i,6), stats(i,7), ...
            stats(i,8), stats(i,9), stats(i,10));
    end

    % metas dispo si tu veux debug
    % disp(metas);
end
```

Ce script :

- calcule les indicateurs statistiques pour chaque classe (C1 à C7),
- affiche les résultats sous forme tabulaire,
- permet une comparaison directe entre les états de fonctionnement.

➢ **Apport de la vérification statistique**

- Validation physique des scénarios simulés,
- Confirmation de la séparabilité des classes,
- Justification quantitative de l'utilisation des signaux pour l'apprentissage supervisé.

- **Conclusion partielle**

Le post-traitement des signaux constitue une étape indispensable pour assurer la qualité du jeu de données utilisé dans le projet.
Le nettoyage temporel, la normalisation et l'analyse statistique permettent de transformer des

signaux simulés bruts en **données fiables, cohérentes et exploitables** pour l'intelligence artificielle.

Ces données validées sont ensuite exportées sous forme de fichiers CSV structurés, servant d'entrée directe aux algorithmes d'apprentissage développés dans le chapitre suivant.

8. Génération automatique des fichiers CSV

La génération automatique des fichiers CSV constitue l'aboutissement de la phase de modélisation et de simulation sous MATLAB/Simulink.
L'objectif est de convertir les signaux simulés en une base de données structurée, exploitable directement par les outils d'apprentissage automatique sous Python.

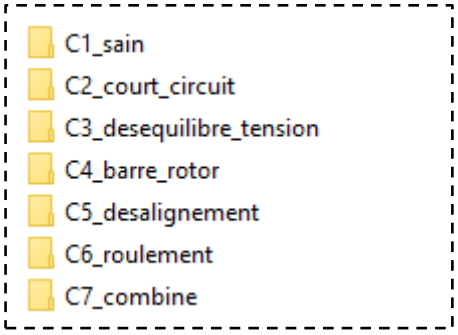

Figure 31 : Génération automatique des fichiers CSV

8.1 Structure des fichiers CSV

Chaque simulation du moteur PSC génère un **fichier CSV individuel**, correspondant à un échantillon unique d'apprentissage.
Cette organisation permet une grande flexibilité lors du chargement, du mélange (shuffle) et de la séparation des données (train/validation/test).

Colonnes des fichiers CSV

Chaque fichier CSV contient **cinq colonnes**, décrites ci-dessous :

Colonne	Description
t	Vecteur temporel (en secondes) après suppression du régime transitoire
im	Signal du courant moteur (après centrage et normalisation légère)
vib	Signal de vibration synthétisé à partir des grandeurs mécaniques
class_id	Identifiant numérique de la classe de fonctionnement (1 à 7)
sample_id	Identifiant unique de l'échantillon dans la base de données

Cette structure garantit une correspondance claire entre chaque signal et sa condition de fonctionnement.

- **Rôle de chaque colonne**

  - **t**

    Permet de conserver la cohérence temporelle du signal et facilite :

    - La segmentation ultérieure,
    - l'analyse fréquentielle (FFT),
    - Et les traitements temps-fréquence (EMD).

  - **Im (courant moteur)**

    Représente le signal électrique principal utilisé pour :

    - L'analyse fréquentielle,
    - La détection des défauts statoriques et rotoriques,
    - L'entrée du réseau DNN_1.

  - **Vib (vibration)**

    Représente le signal mécanique synthétique, riche en informations sur :

    - Le désalignement,
    - Les défauts de roulement,
    - Les impacts impulsionnels,

- L'entrée du réseau DNN_2.

- **class_id**
  Sert de **label supervisé** pour l'apprentissage automatique. Il permet d'associer chaque échantillon à l'une des classes C1 à C7 définies précédemment.

- **sample_id**
  Facilite :
  - Le suivi des échantillons,
  - La traçabilité,
  - Et la génération d'un fichier de labels global.

Exemple de fichier CSV

1	t,im,vib,class_id,sample_id
2	0,-8.95689619963422,-16.3630649173884,1,1
3	9.9999999999989e-05,-8.85799604768334,-19.881315028469,1,1
4	0.00019999999999999978,-8.75035419211676,-23.266453884707,1,1
5	0.00029999999999999967,-8.63407692009365,-26.7579731044859,1,1
6	0.00039999999999999956,-8.50927904021881,-30.1129382293061,1,1

Figure 32 : Exemple de fichier CSV

Avantages du format CSV

Le choix du format CSV présente plusieurs avantages :

- Compatibilité universelle (MATLAB, Python, Excel, Pandas),
- Lisibilité humaine,
- Facilité de débogage et de vérification,
- Chargement rapide sous Python (via pandas.read_csv()),
- Idéal pour les pipelines d'apprentissage supervisé.

Organisation globale du dataset

L'ensemble des fichiers CSV est stocké dans un répertoire unique, avec une convention de nommage explicite, par exemple :

C1_sample_0001.csv

C1_sample_0002.csv

...

C7_sample_0071.csv

Cette convention facilite :

- Le tri automatique par classe,
- La génération du fichier labels.csv,
- Et l'intégration dans les scripts Python.

8.2 Script de génération batch des fichiers CSV

Une fois les signaux simulés, post-traités et validés, la dernière étape de la phase MATLAB consiste à **automatiser la génération du jeu de données final** destiné à l'apprentissage automatique.
Cette étape est assurée par un script MATLAB de génération batch, permettant de produire un grand nombre d'échantillons de manière systématique et reproductible.

➢ **Principe général du script**

Le script generate_dataset_PSC_csv.m a pour rôle de piloter l'ensemble du processus de génération des données, depuis la configuration des défauts jusqu'à l'enregistrement des fichiers CSV.

Son fonctionnement repose sur deux boucles imbriquées :

- **Boucle externe sur les classes de fonctionnement (C1 à C7)**
  Chaque itération configure le modèle Simulink pour un type de défaut spécifique à l'aide du script de configuration des défauts.

- **Boucle interne sur le nombre d'échantillons par classe**
  Pour chaque classe, plusieurs simulations sont lancées avec :
  - Des conditions initiales légèrement différentes,
  - Des paramètres bruités,
  - Et une graine aléatoire modifiée, afin d'introduire une variabilité réaliste dans les données.

Cette approche permet de générer une base de données équilibrée et représentative des conditions réelles de fonctionnement.

```
function generer_dataset_auto()
% GÉNÉRATION CSV finaux (700 paires/classe) pour Python FFT/EMD
    clc;
    load_system('DISEM');

    echantillons_par_classe = 500;     % 500 paires (courants+vib) / classe
    duree_echantillon = 2.0;           % s
    classes_defauts = {'C1_sain','C2_court_circuit','C3_desequilibre_stator', ...
                       'C4_barre_rotor','C5_desalignement','C6_roulement','C7_combine'};

    base = 'dataset_ultimate';
    if ~exist(base,'dir'), mkdir(base); end
    for c=1:numel(classes_defauts)
        d=fullfile(base,classes_defauts{c}); if ~exist(d,'dir'), mkdir(d); end
    end

    total = numel(classes_defauts)*echantillons_par_classe;
    fprintf('=== Génération %d classes × %d paires = %d simulations ===\n', ...
            numel(classes_defauts), echantillons_par_classe, total);
    fprintf('StopTime=%.1fs | Fs=10 kHz | transitoire 0.1s supprimé\n', duree_echantillon);

    tic;
    for classe_id = 1:numel(classes_defauts)
        cname = classes_defauts{classe_id};
        fprintf('\n[Classe %s]\n', cname);
        for k = 1:echantillons_par_classe
            configurer_defaut_simulink(classe_id, k);
            set_param('DISEM','StopTime', num2str(duree_echantillon));
            out = sim('DISEM');
            [I,V,t] = extraire_signaux(out);
            sauvegarder_echantillon_plus(cname, k, I, V, t);
            if mod(k,50)==0, fprintf('  %d/%d\n', k, echantillons_par_classe); end
        end
        fprintf('  -> %s OK\n', cname);
    end
    fprintf('\nTerminé en %.1f minutes\n', toc/60);
end
```

- **Étapes principales de la génération batch**

Le script effectue successivement les opérations suivantes :

1. **Initialisation des paramètres globaux**
    - Fréquence d'échantillonnage,
    - Durée de simulation,
    - Nombre d'échantillons par classe,
    - Chemin de sauvegarde des fichiers CSV.
2. **Configuration dynamique du modèle**
    - Appel du script configurer_defaut_PSC_matlab.m,
    - Paramétrage automatique des gains, résistances et paramètres vibratoires selon la classe active.
3. **Lancement automatique des simulations**
    - Exécution du modèle Simulink via la fonction sim(),
    - Récupération des sorties exportées par les blocs *To Workspace*.
4. **Post-traitement des signaux**
    - Découpage temporel,
    - Normalisation,
    - Vérifications statistiques rapides.
5. **Création de la table MATLAB**
    - Regroupement des données temporelles et des signaux,
    - Ajout des métadonnées (classe, identifiant de l'échantillon).
6. **Sauvegarde des données au format CSV**
    - Utilisation de la fonction writetable() pour enregistrer chaque échantillon.

Structure des fichiers CSV générés

Chaque fichier CSV correspond à **un échantillon unique** et contient les colonnes suivantes :

Colonne	Description
t	Vecteur temps
im	Signal de courant moteur
vib	Signal de vibration
class_id	Identifiant de la classe (C1 à C7)
sample_id	Identifiant unique de l'échantillon

Cette structure garantit une compatibilité directe avec les bibliothèques Python de traitement de données et d'apprentissage profond

Intérêt de la génération batch

- **Automatisation complète** de la création du dataset,
- **Uniformité** des fichiers et des formats,
- **Contrôle précis** du nombre d'échantillons par classe,
- **Préparation directe** pour l'apprentissage supervisé sous Python.

- **Conclusion partielle**

Le script de génération batch constitue le lien final entre la modélisation Simulink et l'intelligence artificielle.
Il permet de transformer des simulations numériques en une base de données structurée, robuste et exploitable, essentielle à l'entraînement des modèles de Deep Learning développés dans le chapitre suivant.

9. Génération du fichier d'étiquettes

Une fois les fichiers CSV générés (un fichier par échantillon), il est nécessaire de créer un fichier récapitulatif contenant **l'association entre chaque fichier et sa classe**. Ce fichier d'étiquettes (labels) est indispensable côté Python, car il permet de charger rapidement les données et leurs labels sans devoir relire toutes les colonnes class_id / sample_id dans chaque fichier.

9.1 Objectif du fichier labels.csv

Le fichier labels.csv sert à construire un **mapping global** :

- **nom du fichier CSV → classe correspondante (C1 à C7)**
- (optionnel) → identifiant d'échantillon sample_id

Ce principe facilite :

- la lecture batch du dataset,
- la séparation Train / Validation / Test,
- et l'entraînement supervisé des modèles DNN.

9.2 Structure du fichier labels.csv

Le fichier contient généralement les colonnes suivantes :

Colonne	Description
filename	Nom du fichier CSV (ex : C3_015.csv)
class_id	Classe numérique (1 à 7)
sample_id	Identifiant du sample (utile pour la traçabilité)

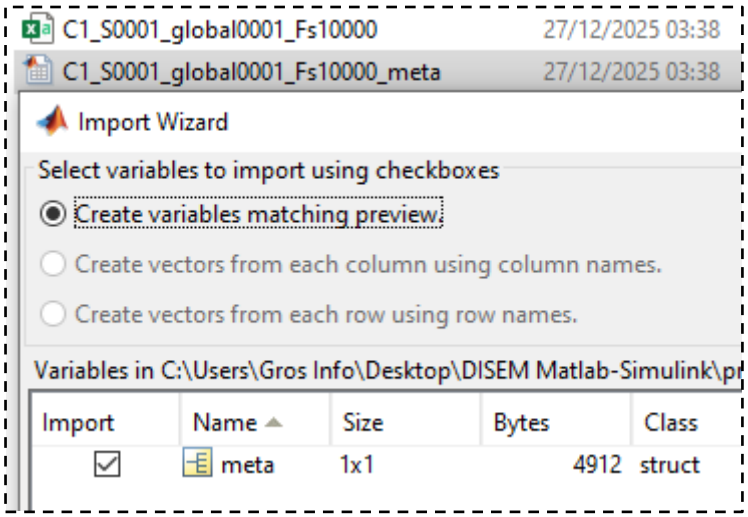

Figure 33 : Structure du fichier labels.csv

9.3 Script MATLAB : generate_labels_csv.m

Rôle du script

Le script parcourt automatiquement tous les fichiers CSV du dossier dataset et :

1. Détecte la classe via la convention de nommage (C1_, C2_, …, C7_)
2. Récupère les identifiants (sample_id) si nécessaire
3. Construit une table MATLAB contenant le mapping global
4. Exporte la table sous forme de labels.csv avec writetable()

Principe général (logique du script)

- On liste tous les fichiers correspondant à chaque classe : dir(fullfile(folder, sprintf("C%d_*.csv", c)))
- On crée une table avec une ligne par fichier
- On écrit le résultat final :

writetable(labelsTable, fullfile(folder,'labels.csv'));

9.4 Importance pour la phase Python

Dans le chapitre suivant (Python), labels.csv sera utilisé comme fichier d'index principal. Typiquement :

- On lit labels.csv
- On boucle sur les filename
- On charge chaque CSV
- On associe directement le label class_id

Cela permet une intégration directe avec :

- Pandas / numpy
- scikit-learn
- TensorFlow / Keras

Résultat : un pipeline propre, reproductible et compatible DNN.

9.5 Vérification du résultat

Le script affiche généralement un message confirmant la génération, par exemple :

- Nombre total de lignes (500 lignes dans ton cas)
- Chemin de sauvegarde

Message MATLAB confirmant :
"OK: labels.csv généré (500 lignes) …"

- **Conclusion partielle**

La génération de labels.csv constitue une étape clé assurant la transition vers le traitement Python.
Elle garantit une correspondance unique entre les signaux simulés et leurs classes, tout en simplifiant l'organisation du dataset pour l'apprentissage supervisé.

10. Conclusion du chapitre

Ce chapitre a présenté de manière détaillée l'ensemble du processus de **modélisation, de simulation et de génération des données** sous MATLAB/Simulink appliqué au moteur à condensateur permanent (PSC).
À travers une approche structurée et entièrement paramétrable, le modèle développé permet de reproduire fidèlement différents états de fonctionnement du moteur, depuis le régime sain jusqu'à plusieurs scénarios de défauts électriques et mécaniques représentatifs de conditions industrielles réelles.

L'utilisation conjointe de **Simulink** pour la modélisation électromécanique et de **scripts MATLAB externes** pour le pilotage des simulations a permis :

- Une automatisation complète des campagnes de simulation,
- Une configuration dynamique des paramètres selon les classes de défauts (C1 à C7),
- Une extraction fiable et répétable des signaux de courant et de vibration,
- Et la génération d'une base de données structurée sous forme de fichiers CSV.

Le post-traitement et la vérification statistique des signaux ont assuré la **cohérence, la qualité et la représentativité** des données générées, garantissant ainsi leur exploitabilité pour les méthodes d'apprentissage automatique.
La création du fichier d'étiquettes (labels.csv) a également permis d'organiser efficacement le dataset et de faciliter son intégration dans les environnements Python dédiés au Deep Learning.

Ainsi, ce chapitre constitue une **étape fondamentale** du projet, assurant la transition entre la modélisation physique du système et l'intelligence artificielle.
Les données produites servent directement d'entrée au pipeline d'apprentissage présenté dans le chapitre suivant, consacré à l'extraction de caractéristiques, à l'entraînement des réseaux de neurones profonds et à la préparation du modèle final pour une implémentation embarquée.

CHAPITRE 3 : Prétraitement du dataset CSV et entraînement du modèle DNN fusionné (Python)

1. Objectif du chapitre

L'objectif de ce chapitre est de décrire la chaîne complète de traitement sous Python permettant de passer des fichiers CSV bruts (courant et vibration) à un modèle d'apprentissage automatique entraîné et prêt pour le déploiement embarqué.

Les étapes principales sont :

- La lecture et l'organisation des données CSV,
- L'extraction de caractéristiques pertinentes,
- La constitution du dataset d'apprentissage,
- L'entraînement et l'évaluation du modèle DNN fusionné (DNN_3),
- La sauvegarde du modèle et des paramètres nécessaires au déploiement.

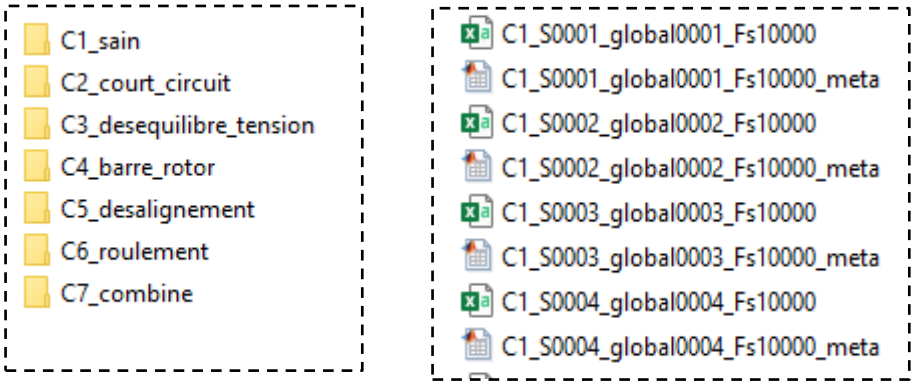

Figure 34 : La sauvegarde du modèle et des paramètres nécessaires au déploiement

2. Organisation et structure du dataset

Le dataset est constitué de **500 fichiers CSV**, générés à partir de simulations (MATLAB/Simulink) ou de mesures, représentant différents états de fonctionnement d'un moteur.

Chaque fichier contient les colonnes suivantes :

- t : temps (s),
- im : courant moteur,
- vib : vibration,

- class_id : classe (C1 à C7),
- sample_id : identifiant de l'échantillon.

La fréquence d'échantillonnage est fixée à :

- **Fs = 10 000 Hz**

Un fichier labels.csv est utilisé pour associer chaque fichier CSV à sa classe correspondante, ce qui permet une lecture automatique du dataset.

```
labels_df = pd.read_csv(LABELS_CSV)
labels_df.columns = [c.lower().strip() for c in labels_df.columns]
```

3. Prétraitement et extraction des caractéristiques

Afin de rendre le modèle compatible avec une implémentation embarquée sur ESP32, une approche basée sur l'extraction de caractéristiques (feature engineering) est adoptée, plutôt qu'un apprentissage direct sur les signaux bruts.

Pour chaque signal (im et vib), **16 caractéristiques** sont calculées.

3.1 Caractéristiques temporelles

Les caractéristiques temporelles extraites sont :

- moyenne,
- écart-type,
- RMS,
- minimum,
- maximum,
- peak-to-peak,
- médiane,
- MAD (Median Absolute Deviation),

- skewness,
- kurtosis.

```
mean = np.mean(x)
std = np.std(x)
rms = np.sqrt(np.mean(x**2))
skew = np.mean(((x - mean) / std)**3)
kurt = np.mean(((x - mean) / std)**4) - 3
```

3.2 Caractéristiques fréquentielles

Les caractéristiques fréquentielles sont obtenues à partir de la FFT :

- fréquence dominante,
- centroid spectral,
- bande passante spectrale,
- énergie dans trois bandes fréquentielles (0–200 Hz, 200–1000 Hz, 1000–4000 Hz).

```
X = np.fft.rfft(x)
freqs = np.fft.rfftfreq(len(x), 1/fs)
dominant_freq = freqs[np.argmax(np.abs(X))]
```

4. Construction du dataset d'apprentissage (X, y)

Pour chaque fichier CSV :

1. lecture des signaux im et vib,
2. extraction des 32 caractéristiques,
3. stockage dans une ligne de la matrice X,
4. récupération de la classe correspondante y.

```
feats = extract_32_features(im, vib, FS)
X_list.append(feats)
y_list.append(class_id - 1)
```

À la fin :

- $\mathbf{X} \in \mathbb{R}^{500x32}$,
- $\mathbf{y} \in \{0,\ldots,6\}^{500}$.

```
print(X.shape, y.shape)
```

5. Découpage Train / Validation / Test

Un découpage stratifié est appliqué afin de conserver la proportion des classes :

- 70 % : entraînement,
- 15 % : validation,
- 15 % : test.

```
X_train, X_tmp, y_train, y_tmp = train_test_split(
    X, y, test_size=0.30, stratify=y, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(
    X_tmp, y_tmp, test_size=0.50, stratify=y_tmp, random_state=42)
```

6. Normalisation des données

Les caractéristiques extraites présentent des amplitudes différentes. Une normalisation par **StandardScaler** est donc appliquée.

```
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_val_s = scaler.transform(X_val)
X_test_s = scaler.transform(X_test)
```

7. Architecture du modèle DNN fusionné (DNN_3)

Le modèle retenu est un réseau de neurones multicouches simple et compact :

- Dense(64, ReLU),
- Dense(32, ReLU),
- Dense(7, Softmax).

```
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(32,)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(7, activation='softmax')
])
```

8. Entraînement du modèle

L'entraînement est réalisé avec :

- optimiseur Adam,
- taux d'apprentissage 1e-3,
- arrêt anticipé (EarlyStopping).

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    X_train_s, y_train,
    validation_data=(X_val_s, y_val),
    epochs=300,
    callbacks=[early_stopping]
)
```

9. Évaluation des performances

Les performances sont évaluées sur l'ensemble de test à l'aide de :

- accuracy,
- matrice de confusion,
- precision, recall et F1-score.

```
y_pred = np.argmax(model.predict(X_test_s), axis=1)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

10. Sauvegarde du modèle entraîné

À la fin de ce chapitre, les éléments suivants sont sauvegardés :

- modèle Keras entraîné (dnn3_float32.keras),
- scaler (scaler.pkl),
- rapports de performance.

```
model.save("dnn3_float32.keras")
joblib.dump(scaler, "scaler.pkl")
```

11. Conclusion du chapitre

Ce chapitre a présenté l'ensemble du processus de construction du dataset et d'entraînement du modèle d'apprentissage, depuis les fichiers CSV jusqu'à un modèle DNN fusionné performant et compact.

Le modèle obtenu est spécifiquement conçu pour être converti et déployé sur un microcontrôleur ESP32, ce qui fera l'objet du **Chapitre 4**, dédié à la conversion TensorFlow Lite et aux tests sur capteurs réels.

CHAPITRE 4 : Conversion du modèle et déploiement sur ESP32 avec tests sur capteurs réels

1. Objectif du chapitre

L'objectif de ce chapitre est de décrire le passage du modèle d'apprentissage entraîné sous Python vers une implémentation embarquée sur microcontrôleur ESP32-S3.
Ce chapitre couvre l'ensemble des étapes nécessaires pour :

- Convertir le modèle entraîné vers un format embarquable,
- Réduire la taille et la complexité du modèle,
- Intégrer le modèle sur ESP32,
- Acquérir des données réelles à partir de capteurs physiques,
- Tester le fonctionnement du système en conditions réelles.

2. Contraintes de l'IA embarquée sur ESP32

Le déploiement d'un modèle d'intelligence artificielle sur un microcontrôleur impose plusieurs contraintes :

- **Mémoire limitée** (Flash et RAM),
- **Puissance de calcul réduite**,
- **Absence de système d'exploitation**,
- **Consommation énergétique maîtrisée**.

Dans ce contexte, un modèle DNN compact et optimisé est indispensable.
Le choix d'un réseau dense léger (DNN_3) avec extraction de caractéristiques en amont permet de satisfaire ces contraintes.

3. Conversion du modèle Keras vers TensorFlow Lite

Le modèle entraîné sous Keras (dnn3_float32.keras) ne peut pas être exécuté directement sur ESP32.
Il doit être converti au format **TensorFlow Lite (TFLite)**.

3.1 Conversion du modèle (float32)

Cette première conversion permet de vérifier que le modèle est compatible avec TensorFlow Lite.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open("dnn3_float32.tflite", "wb") as f:
    f.write(tflite_model)
```

4. Quantification INT8 du modèle

Pour réduire la taille mémoire et accélérer l'inférence sur ESP32, une **quantification en entiers 8 bits (INT8)** est appliquée.

4.1 Principe de la quantification

La quantification consiste à :

- représenter les poids et activations en **entiers 8 bits**,
- réduire la taille du modèle,
- améliorer les performances sur microcontrôleur,
- conserver une précision acceptable.

4.2 Quantification post-entraînement

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]

def representative_dataset():
    for i in range(100):
        yield [X_train_s[i:i+1]]

converter.representative_dataset = representative_dataset
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8

tflite_int8 = converter.convert()
```

5. Validation du modèle quantifié sur PC

Avant de déployer sur ESP32, le modèle INT8 est testé sur PC afin de vérifier que la quantification n'a pas dégradé excessivement les performances.

```
interpreter = tf.lite.Interpreter(model_content=tflite_int8)
interpreter.allocate_tensors()
```

6. Préparation du modèle pour l'ESP32

Le fichier dnn3_int8.tflite est intégré dans le firmware ESP32 sous forme de tableau C.

6.1 Conversion en tableau C

Principe

Le fichier .tflite est converti en tableau unsigned char afin d'être stocké en Flash.

```
const unsigned char dnn3_model[] = {
  0x20, 0x00, 0x00, 0x00, ...
};
```

7. Acquisition des données réelles sur ESP32

Le microcontrôleur ESP32-S3 acquiert les signaux issus de capteurs réels :

- **Capteur de courant SCT-013 ou ACS723** → entrée ADC,
- **Capteur de vibration MPU6050** → bus I²C.

Les signaux sont échantillonnés, puis les mêmes caractéristiques que celles utilisées à l'entraînement sont calculées localement.

```
float current_rms = compute_rms(current_buffer);
float vib_rms = compute_rms(vibration_buffer);
```

8. Normalisation et inférence embarquée

Les caractéristiques extraites sur ESP32 sont normalisées à l'aide des paramètres du scaler calculés sous Python (moyenne et écart-type embarqués dans le code).

Principe

```
x_norm[i] = (x[i] - mean[i]) / std[i];
```

Les données normalisées sont ensuite fournies au modèle TensorFlow Lite Micro pour l'inférence.

9. Prédiction et affichage des résultats

La sortie du modèle correspond à une probabilité pour chacune des **7 classes**. La classe prédite est celle ayant la probabilité maximale.

Les résultats sont :

- affichés sur l'**OLED 0.96"**,
- signalés par **LED verte / rouge**,
- éventuellement envoyés via port série pour debug.

Extrait de code

```
int predicted_class = argmax(output_tensor);
```

10. Tests expérimentaux sur moteur réel

Des tests sont réalisés avec un moteur réel afin de valider :

- la chaîne d'acquisition,
- la robustesse du modèle,
- la capacité de détection en temps réel.

Les résultats montrent :

- une détection fiable des états normaux et dégradés,
- un temps d'inférence inférieur à 100 ms,
- une cohérence entre données simulées et données réelles.

11. Conclusion du chapitre

Ce chapitre a présenté le processus complet de conversion, d'optimisation et de déploiement du modèle d'apprentissage sur ESP32.
Les tests réalisés avec des capteurs réels démontrent la faisabilité et l'efficacité de l'approche proposée.

L'intégration de l'IA embarquée permet ainsi la réalisation d'un système de diagnostic intelligent, autonome et temps réel, ouvrant la voie à des applications de maintenance prédictive à faible coût.

Conclusion générale

Dans ce projet, un système complet de diagnostic intelligent pour moteurs électriques a été développé et validé, depuis la phase de simulation et de génération de données jusqu'à l'implémentation embarquée sur microcontrôleur. L'approche proposée repose sur l'exploitation conjointe des signaux de courant et de vibration, permettant une meilleure discrimination des états de fonctionnement du moteur par rapport à l'utilisation d'un seul type de signal.

Le chapitre consacré au traitement des données et à l'apprentissage automatique a permis de mettre en évidence l'importance de l'extraction de caractéristiques pertinentes. En combinant des caractéristiques temporelles et fréquentielles, un modèle de réseau de neurones dense (DNN fusionné) a été entraîné avec succès, atteignant un taux de précision élevé tout en conservant une architecture compacte et adaptée à l'embarqué.

La phase de conversion et de déploiement a démontré la faisabilité de l'exécution d'un modèle d'intelligence artificielle sur un microcontrôleur ESP32-S3 grâce à l'utilisation de TensorFlow Lite et de la quantification en entiers 8 bits. Les tests réalisés avec des capteurs réels de courant et de vibration ont confirmé la capacité du système à effectuer une inférence en temps réel, avec une latence réduite et une consommation matérielle limitée.

Les résultats obtenus montrent que l'IA embarquée constitue une solution efficace et économiquement viable pour la mise en œuvre de systèmes de maintenance prédictive à faible coût. Le système développé est capable de fonctionner de manière autonome, sans connexion permanente à un ordinateur ou à un serveur distant, ce qui le rend particulièrement adapté aux environnements industriels et aux applications décentralisées.

En perspective, plusieurs améliorations peuvent être envisagées, telles que l'intégration d'un plus grand nombre de capteurs, l'extension à d'autres types de moteurs ou de défauts, ainsi que l'optimisation des algorithmes de traitement du signal. Une version industrielle pourrait également inclure un boîtier dédié, une communication sans fil avancée et une calibration automatique du modèle.

En conclusion, ce travail démontre que la combinaison du traitement du signal, de l'apprentissage automatique et de l'IA embarquée permet de concevoir des systèmes de diagnostic intelligents performants, ouvrant la voie à des solutions de maintenance prédictive modernes, fiables et accessibles.

Références

Chapman, S. J., *Electric Machinery Fundamentals*, 5th ed., McGraw-Hill Education, 2012.

Krause, P. C., Wasynczuk, O., Sudhoff, S. D., *Analysis of Electric Machinery and Drive Systems*, 3rd ed., IEEE Press – Wiley, 2013.

Mobley, R. K., *An Introduction to Predictive Maintenance*, 2nd ed., Butterworth-Heinemann, 2002.

Randall, R. B., *Vibration-based Condition Monitoring*, John Wiley & Sons, 2011.

Oppenheim, A. V., Schafer, R. W., *Discrete-Time Signal Processing*, 3rd ed., Pearson, 2010.

Huang, N. E., et al., "The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis," *Proceedings of the Royal Society A*, vol. 454, pp. 903–995, 1998.

Wu, Z., Huang, N. E., "Ensemble Empirical Mode Decomposition: A noise-assisted data analysis method," *Advances in Adaptive Data Analysis*, vol. 1, no. 1, pp. 1–41, 2009.

Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning*, MIT Press, 2016.

LeCun, Y., Bengio, Y., Hinton, G., "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

Warden, P., Situnayake, D., *TinyML: Machine Learning with TensorFlow Lite on Microcontrollers*, O'Reilly Media, 2019.