Departement GE

2023/2024

5ème année

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Rapport mini projet : Edge AI

Melvyn Dehunde-Agblo|Jiawei Wu
22/01/2024

Table des matières

Table d'illustration

Introduction

L'intégration de l'Intelligence Artificielle (IA) dans les systèmes embarqués constitue une avancée majeure dans l'industrie des technologies. Ce rapport présente le processus de conception et de mise en œuvre d'un modèle d'IA sur la plateforme Jetson Nano, un microcontrôleur qui nous a été imposé dans le cadre d'un projet du Département Génie Électrique. Le but de ce projet est double : explorer les capacités de la Jetson Nano en matière d'IA embarquée et relever les défis techniques liés à l'utilisation de modèles d'IA sur du matériel spécialisé. Avant de plonger dans les détails techniques, il est essentiel de présenter brièvement le matériel utilisé.

La Jetson Nano est un ordinateur monocarte développé par NVIDIA. Il est conçu pour les applications d'IA et d'apprentissage automatique en périphérie de réseau (edge computing). Il comprend un processeur Quad-core ARM Cortex-A57, un GPU 128-core basé sur l'architecture Maxwell, et 4 GB de RAM LPDDR4. Sa connectivité variée, incluant USB, HDMI, GPIO, I2C, en fait un choix adapté pour les applications IoT (Internet des Objets). Bien que moins puissante que des modèles supérieurs comme le Jetson TX2 ou l'AGX Xavier, la Jetson Nano est parfaitement adaptée aux projets nécessitant un traitement IA en périphérie, avec des contraintes de budget et d'énergie. Ses limites incluent une puissance de calcul inférieure à celle des GPU de bureau et une dépendance à une carte microSD pour le stockage.

Le rapport suit une progression logique, débutant par une introduction au matériel et aux outils logiciels utilisés, en passant par la configuration de l'environnement de test et le choix du modèle d'IA, et enfin, en évaluant la performance du modèle à travers des mesures énergétiques et des résultats d'entraînement. Chaque étape est détaillée pour fournir une compréhension claire de la méthodologie appliquée et des défis surmontés. La synthèse de ces étapes offre une perspective globale sur l'efficacité des modèles d'IA en environnement contraint.

I. Prise en Main

Dans le cadre de notre projet utilisant la plateforme Youpi, nous avons franchi une étape essentielle en configurant et en sécurisant l'accès aux dispositifs nécessaires. La plateforme Youpi, facilite la gestion et l'utilisation des appareils.

La configuration et l'utilisation de la plateforme Youpi impliquent plusieurs étapes clés. Tout d'abord, nous avons créé nos comptes sur la plateforme, une étape essentielle pour accéder aux ressources. Ensuite, nous avons généré des clés SSH privées et publiques pour sécuriser nos connexions. Après cela, nous avons enregistré notre clé publique SSH dans notre profil Youpi, garantissant des connexions sécurisées. La configuration du client SSH a été effectuée en modifiant le fichier ~/.ssh/config avec des paramètres spécifiques. Cela nous a permis d'établir des connexions SSH aux appareils cibles, offrant un accès direct et sécurisé pour le développement et les tests. Enfin, la plateforme Youpi a facilité la réservation et l'utilisation des appareils nécessaires, comme les Jetson Nano, pour notre projet, ce qui a été un facteur clé dans l'avancement de notre travail.

II. Mise en Place de l'Architecture de Test

La figure 1 illustre un schéma de l'architecture qui utilise Ansible pour l'initialisation des machines. Au cœur de ce système se trouve un composant principal, "main", qui interagit avec divers modules, notamment "Tegrastats", "Shelly" et "Model. Les résultats du système sont gérés dans une section "Output", qui inclut les données sur les "Résultats d'entraînement" et la "Consommation", structurées dans des sous-dossiers associés à des tests spécifiques. Un processus distinct, "DockerRun", accède en lecture seule à ces données de sortie, impliquant une séparation entre l'exécution des applications ou des tests et le stockage des résultats. Cela met en évidence une structure organisée où la configuration, le traitement et le stockage des données sont clairement délimités.

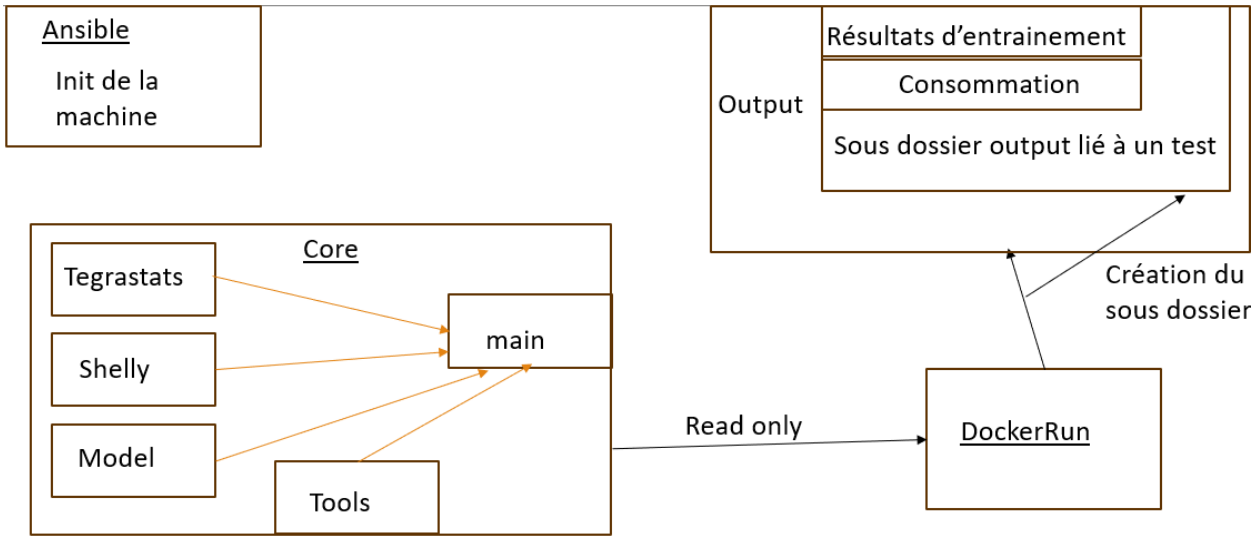


Figure 1 Architecture du projet

1. Ansible

Dans le cadre de notre projet, une étape essentielle a été la préparation et la configuration des dispositifs Jetson Nano. Cette phase a impliqué l'utilisation de l'outil d'automatisation Ansible pour garantir une configuration cohérente et efficace de chaque dispositif. Le but de cette démarche était de créer un environnement standardisé pour le développement et les tests de nos modèles d'IA. Un playbook spécifique a été élaboré pour installer et configurer les outils et les dépendances nécessaires sur chaque Jetson Nano. Les étapes clés de ce processus sont décrites ci-dessous :

1. Configuration de NTP

But : Synchroniser l'heure des dispositifs avec les serveurs NTP pour garantir la cohérence des logs et des processus.

Actions :

- Définition du fuseau horaire.
- Configuration et redémarrage du service systemd-timesyncd.

2. Nettoyage et Installation des Dépendances

But : Maintenir un environnement propre et à jour.

Actions :

- Mise à jour des paquets obsolètes.
- Installation de Python, Git, et des outils essentiels.

3. Installation et Configuration de Docker

But : Installer Docker pour utiliser Tensorflow

Actions :

- Installation de Docker et docker-py.
- Ajout des utilisateurs au groupe Docker.

4. Installation du Runtime NVIDIA Docker

But : Permettre l'utilisation des GPU NVIDIA dans les conteneurs Docker.

Actions :

- Installation de nvidia-docker2.

5. Préparation de l'Environnement Docker

But : Créer un environnement Docker standardisé pour les tests d'IA.

Actions :

- Téléchargement de l'image Docker melvyn212/edge_ai_jnano: latest dont nous allons parler en détail par la suite

Cette approche nous a permis de déployer rapidement des environnements de test cohérents et fiables, essentiels pour le développement et l'évaluation de notre solution.

Ce playbook a été intégré dans un executable pour pouvoir l'utiliser directement sur les appareils cibles.

2. Docker

Nous avons développé une image Docker illustré par le code 1, sur mesure pour les Jetson Nano, en s'appuyant sur une image de base de TensorFlow optimisée par NVIDIA, spécifiquement adaptée à l'Intelligence Artificielle Embarquée. Le Dockerfile inclut des mises à jour du système, la configuration de la locale en UTF-8, la définition des variables d'environnement pour Python, ainsi que l'installation de bibliothèques Python essentielles telles que CodeCarbon pour le suivi de l'empreinte carbone, Matplotlib pour la visualisation graphique, Scikit-Learn pour le machine learning, et d'autres outils nécessaires

```
# Image de base TensorFlow pour Jetson nano source :
https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-tensorflow
FROM nvcr.io/nvidia/l4t-tensorflow:r32.4.4-tf1.15-py3

# Mise à jour et mise à niveau des paquets
RUN apt-get update && apt-get upgrade -y

# Installation des locales et configuration de la locale UTF-8
RUN apt-get install -y locales && \
locale-gen en_US.UTF-8

# Définir les variables d'environnement pour la locale
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8

# Définir PYTHONIOENCODING à utf-8
ENV PYTHONIOENCODING=utf-8

RUN pip3 install codecarbon

RUN pip3 install --upgrade matplotlib

RUN apt-get install -y git

RUN pip3 install git+https://github.com/thu-ml/zhusuan.git

RUN pip3 install git+https://github.com/haowen-xu/tfsnippet.git@v0.2.0-alpha1

RUN pip3 install imageio==2.4.1

RUN pip3 install fs==2.3.0
RUN pip3 install tqdm==4.28.1
RUN pip3 install click==7.0
RUN pip3 install six==1.11.0
RUN pip3 install tensorflow_probability==0.8
RUN pip3 install numpy

RUN apt-get install -y jq


# Installer scikit-learn et nettoyer
RUN apt-get install -y python3-sklearn && \
rm -rf /var/lib/apt/lists/*
```

```
CMD ["python3","EdgeAI/main.py"]
```

Code 1 : Image Docker

Nous avons développé un script Bash pour faciliter le lancement de conteneurs Docker sur les Jetson Nano illustré par le code 2. Ce script configure l'environnement nécessaire pour exécuter nos modèles d'IA en utilisant l'image Docker personnalisée que nous avons créé.

Fonctionnalités du Script

Utilisateur et Chemins : Définition de l'utilisateur et des chemins pour les statistiques du système, le code, les sorties, et les bibliothèques CUDA qui signifie Compute Unified Device Architecture, une plateforme de calcul parallèle et un modèle de programmation inventés par NVIDIA. Elle permet aux développeurs d'utiliser une version modifiée du langage de programmation C (ainsi que d'autres langages tels que C++, Python et Fortran) pour écrire des programmes qui peuvent être exécutés en parallèle sur les processeurs graphiques (GPU) de NVIDIA..

Tegrastats est un utilitaire spécifique aux dispositifs NVIDIA Jetson qui fournit des informations détaillées sur l'utilisation des ressources système, comme le GPU, la CPU, et la mémoire et la consommation. Dans notre script d'exécution Docker, nous intégrons tegrastats pour surveiller et enregistrer les performances du système pendant l'exécution de nos modèles d'IA.

Image Docker : Utilisation de l'image melvyn212/edge_ai_jnano:latest.

Montage de Volumes : Montage des chemins nécessaires dans le conteneur pour assurer l'accès aux fichiers et bibliothèques requis.

Support CUDA : Configuration pour permettre au conteneur d'accéder aux ressources CUDA sur l'hôte, crucial pour les calculs liés à l'IA.

Le script lance un conteneur Docker en mode interactif (-it), avec suppression automatique après son utilisation (--rm), et avec le support des GPU NVIDIA (--runtime nvidia). Les chemins nécessaires sont montés dans le conteneur pour assurer l'accès aux fichiers et aux bibliothèques CUDA de l'hôte.

Objectif

Le script a pour objectif de faciliter l'exécution de nos modèles d'IA dans un environnement Docker. Voici les points clés :

- Fournir le Code au Conteneur Docker

Chemin du Code : Le script définit le chemin d'accès à nos scripts et modèles d'IA, qui sont ensuite montés dans le conteneur Docker pour l'exécution.

Exécution dans Docker : L'environnement isolé et contrôlé de Docker assure une exécution cohérente et reproductible de nos modèles.

- Capturer les Résultats d'Exécution

Dossier Output : Un dossier spécifié (output) est utilisé pour stocker les résultats et les données générés par nos modèles d'IA.

Montage du Dossier : Ce dossier est monté dans le conteneur Docker, permettant ainsi la sauvegarde directe des résultats depuis l'environnement d'exécution.

- Utilisation de Tegrastats

Surveillance des Performances : tegrastats est monté dans le conteneur pour permettre le suivi en temps réel des performances du système pendant l'exécution des modèles.

Analyse des Données : Les données collectées par tegrastats peuvent être utilisées pour optimiser les performances de nos modèles et pour comprendre l'impact de l'IA sur les ressources matérielles.

```
#!/bin/bash
USER="adehundeag"

# Définissez les chemins dans des variables
TEGRASTATS_PATH="/usr/bin/tegrastats"
CODE_PATH="/home/${USER}/Edge-AI-For-SHM/Core"
OUTPUT_PATH="/home/${USER}/Edge-AI-For-SHM/output"
DOCKER_IMAGE="melvyn212/edge_ai_jnano:latest"

# Chemin vers les bibliothèques CUDA sur l'hôte
CUDA_LIB_PATH="/usr/local/cuda/lib64"

# Exécutez le conteneur Docker avec les volumes montés et le support CUDA
sudo docker run -it --rm --runtime nvidia \
-v "${TEGRASTATS_PATH}:${TEGRASTATS_PATH}:ro" \
-v "${CODE_PATH}:/EdgeAI" \
-v "${OUTPUT_PATH}:/output" \
-v "${CUDA_LIB_PATH}:${CUDA_LIB_PATH}:ro" \
-e LD_LIBRARY_PATH="${CUDA_LIB_PATH}:$LD_LIBRARY_PATH" \
"${DOCKER_IMAGE}"
```

Code 2 : Script Docker

3. Outils de mesure énergétiques

a. Tegrastats

Conception des Scripts

Tegrastats.py

Objectif : Automatiser le lancement et l'arrêt du processus tegrastats pour la collecte de données de puissance sur les Jetson Nano.

Fonctionnalités Clés :

Paramétrage Dynamique : Permet de spécifier l'intervalle de collecte des données et le mode verbeux.

Gestion des Processus : Utilise subprocess.Popen pour lancer tegrastats et os.killpg pour l'arrêter proprement, garantissant ainsi une collecte de données fiable.

Enregistrement des Données : Redirige les sorties de tegrastats vers un fichier log pour une analyse ultérieure.

Parse.py

Objectif : Analyser et structurer les données collectées par tegrastats dans un format exploitable (CSV).

Fonctionnalités Clés :

Extraction de Données : Utilise des expressions régulières pour extraire des informations spécifiques des logs (comme l'utilisation de la RAM, du CPU, et du GPU).

Création de Fichiers CSV : Transforme les données extraites en un fichier CSV structuré, facilitant l'analyse et la visualisation des données.

Utilisation Pratique

Collecte de Données avec Tegrastats.py

Mise en Place : Nous spécifions la periode de collecte à 1000ms et le fichier de log dans tegrastats.py.

Lancement (illustré par le code 3) : Le script lance tegrastats, qui commence à enregistrer les données de performance du système dans le fichier de log.

```
#tegrastats
interval = 1000 #ms
tegr_log_file = os.path.join(output_path, 'tegra_output_log.txt')
tegr_csv_file=os.path.join(output_path, 'tegra_output_log.csv')
verbose = False
tegrastats = Tegrastats(interval, tegr_log_file, verbose)
process, current_time=tegrastats.run()
```

Code 3 : lancement de Tegrastats

Analyse des Données avec Parse.py

Traitement Post-Collecte : Après avoir arrêté tegrastats (illustré par le code 4), nous utilisons parse.py pour traiter le fichier de log.

Création de CSV : Le script parse.py lit le fichier de log, extrait les données pertinentes et les enregistre dans un fichier CSV pour une analyse et une visualisation facilitée.

```
#tegrastats
tegrastats.stop(process)
parser = Parse(interval, tegr_log_file,current_time)
parser.parse_file()
```

Code 4: Arrêt de Tegrastats

La conception et l'application de tegrastats.py et parse.py dans notre projet illustrent notre approche méthodique pour la collecte et l'analyse de données de performance. Ces scripts jouent un rôle fondamental dans l'optimisation de nos modèles d'IA, en nous fournissant des données sur l'efficacité opérationnelle et énergétique.

b. Shelly

Dans notre projet, l'intégration de prises connectées Shelly, contrôlables à distance, nous permettent de surveiller et de réguler la consommation électrique des dispositifs utilisés. L'usage de telles technologies est utile dans les projets d'IA embarquée où la gestion de l'énergie est un aspect critique de la performance et de la durabilité.

Utilisation du Script Shelly pour la Surveillance de l'Énergie

Script et Commandes Shelly

Le script fourni, shelly.py, est un outil utilisé pour interagir avec les prises connectées Shelly. Il permet d'exécuter une variété de commandes pour contrôler les dispositifs Shelly, exécuter des scripts personnalisés, et appeler des API de scripts pour des interactions spécifiques. Ce script offre une flexibilité et une automatisation dans la gestion de l'alimentation électrique et le suivi de la consommation d'énergie.

Scripts Spécifiques

PowerTracker.js : Un script utilisé pour surveiller la consommation d'énergie. Il conserve les 100 dernières mesures de puissance et les rend disponibles sur demande.

En utilisant le script PowerTracker.js, nous pouvons recueillir des données détaillées sur la consommation d'énergie des appareils connectés aux prises Shelly. La sortie de ces données est affichée dans le terminal sous forme de JSON. Nous avons créé deux autres fichiers pour obtenir plus de 100 mesures et enregistrer les données dans un fichier.csv

shellydata.py

Ce fichier Python est conçu pour interagir directement avec les scripts Shelly, convertir les données JSON en CSV, et gérer les processus de collecte de données.

Fonctionnalités Clés

Interaction avec les Scripts Shelly : Crée, exécute, arrête et supprime les scripts Shelly.

Conversion JSON en CSV : Transforme les données énergétiques JSON en un format CSV structuré.

Gestion des Processus : Lance et arrête des processus pour la collecte de données.

Utilisation Pratique

Création et Lancement de Script Shelly : Utilise create_script et start_script pour démarrer la collecte des données énergétiques.

Arrêt et Conversion des Données : Après la collecte, utilise stop_process, stop_script, et call_script pour arrêter la collecte et convertir les données en CSV.

getData.sh

Ce script bash illustré par le code 5 est utilisé pour automatiser la collecte continue de données à partir des dispositifs Shelly.

Fonctionnalités Clés

Boucle de Collecte Continue : Récupère les dernières données énergétiques et les ajoute à un fichier JSON.

Traitement des Erreurs : Gère les interruptions et les erreurs pour assurer une terminaison propre du script.

Utilisation Pratique

Démarrage du Script : Exécution de getData.sh en spécifiant le chemin du fichier log pour commencer la collecte des données.

Arrêt du Script : Le script peut être interrompu manuellement ou via un signal pour arrêter la collecte.

```
#!/bin/bash

finish() {
    # Ne fait rien à l'interruption, juste sortir du script
    exit
}

handle_error() {
    echo "Erreur lors de l'exécution de la commande. Arrêt du script."
    finish
}

trap finish SIGINT SIGTERM
trap handle_error ERR

logfile="${1:-/EdgeAI/Shelly/log.json}"

if [ ! "$logfile" ]; then
    echo "Usage: $0 path_to_log_file"
    exit 1
fi

# Initialiser le fichier avec un tableau vide []
echo '[]' > "$logfile"

while true; do
    output=$(./EdgeAI/Shelly/shelly.py -p scripts call 1 "api?yield" | jq '.[-1]')
    if [ ! -z "$output" ]; then
        # Utiliser jq pour ajouter les nouvelles données au tableau dans le fichier JSON
        jq --argjson new_data "$output" '. += [$new_data]' "$logfile" > temp.json && mv temp.json
"$logfile"
```

```
    fi
    sleep 1
done
```

Code 5 : Acquisition des données de Shelly depuis le terminal

Utilisation Pratique

- Mise en Place et Collecte de Données

1. Création et Lancement du Script Shelly

Création du Script :

Nous commençons par créer un script Shelly nommé 'power' depuis la machine hôte, en utilisant le fichier PowerTracker.js.

Lancement du Script :

Le script illustré par le code 6 est ensuite lancé en appelant start_script avec l'ID du script (1 dans ce cas) et un endpoint spécifique ("api?yield").

2. Démarrage de la Collecte de Données

Initialisation de la Collecte :

Un processus de collecte de données est démarré avec getdata, en spécifiant le chemin du fichier log JSON.

```
#SHELLY

create_script('power', '/EdgeAI/Shelly/PowerTracker.js')
start_script(1,"api?yield")
shelly_log_file = os.path.join(output_path, 'log.json')
shelly_process = getdata(shelly_log_file)
shelly_csv_file=os.path.join(output_path, 'shelly.csv')
```

Code 6 : Lancement de Shelly

- Surveillance et Arrêt de la Collecte

1. Surveillance du Code

Code à Monitorer :

Pendant que shelly_process est en cours d'exécution, le code que nous souhaitons surveiller est exécuté.

2. Arrêt du Processus et du Script

Arrêt de la Collecte :

Une fois la surveillance terminée, le processus de collecte est arrêté en utilisant stop_process(shelly_process).

Arrêt et Suppression du Script Shelly :

Le script Shelly est ensuite arrêté et supprimé en utilisant stop_script(1) et delete_script(1) respectivement.

- Transformation et Sauvegarde des Données

1. Conversion des Données JSON en CSV

Chemin des Fichiers :

Les chemins des fichiers log et CSV sont définis en utilisant le chemin de sortie (output_path).

Exécution de la Conversion :

Le script illustré par dans le code 7 incluant call_script est utilisé pour convertir les données JSON du fichier log en un fichier CSV pour une analyse plus facile.

```
# #SHELLY

stop_process(shelly_process)
stop_script(1)
delete_script(1)
call_script(shelly_log_file,shelly_csv_file)
```

Code 7 : Arrêt de Shelly

Les fichiers shellydata.py et getData.sh travaillent ensemble pour fournir une solution complète pour la collecte et le traitement des données énergétiques des prises Shelly. Cette combinaison offre une flexibilité dans la surveillance de la consommation énergétique, essentielle pour l'optimisation des performances et l'efficacité énergétique dans les projets d'IA embarquée.

c. TOOLS

Fonction de Création de Répertoires

La fonction create_output_directory illustré par le code 8 dans notre script joue un rôle clé en organisant et en stockant les sorties graphiques. Cette fonction génère automatiquement un nom de répertoire unique basé sur la date et l'heure actuelles, assurant ainsi que chaque session de collecte de données a son propre espace de stockage distinct. En créant un nouveau répertoire pour chaque exécution du script, nous évitons la surcharge et la confusion avec les données précédemment collectées. Cette méthode de

nommage ("%Y_%m_%d_%H_%M_%S") et de stockage facilite non seulement le suivi des résultats sur une base temporelle mais assure également une organisation systématique des données pour des références et analyses futures. La fonction modifie également les permissions du répertoire pour assurer un accès facile aux données sauvegardées.

```
def create_output_directory(base_path):
    # Obtenir l'heure actuelle
    current_time = datetime.datetime.now()
    # Formater l'heure pour l'inclure dans le nom du dossier (par exemple,
'output_2023_12_08_15_30_00')
    time_str = current_time.strftime("%Y_%m_%d_%H_%M_%S")
    directory_name = f"output_{time_str}"
    full_path = os.path.join(base_path, directory_name)

    os.makedirs(full_path)
    os.chmod(full_path,0o777)

    return full_path
```

Code 8 : Création du répertoire de sortie

Fonction de Tracé de Graphiques

La fonction plot illustré par le code 9 est conçue pour visualiser les données de puissances. Elle lit plusieurs fichiers CSV, chacun contenant des données telles que le temps et la consommation d'énergie, et trace ces données sur un même graphique pour faciliter la comparaison. Cette approche visuelle est essentielle pour l'analyse de la consommation d'énergie, car elle permet de mettre en évidence les tendances, les anomalies ou les modèles dans les données collectées. Les graphiques générés offrent une compréhension immédiate et intuitive des performances énergétiques des dispositifs surveillés. En outre, cette fonction permet une personnalisation étendue, où chaque série de données peut être étiquetée et stylisée différemment, offrant ainsi une clarté dans l'interprétation des résultats.

```
file_info_list = [
     (tegr_csv_file,    'Time    (mS)',    'Current    POM_5V_IN    Power    Consumption
(mW)',"Tegrastats",1),
    (tegr_csv_file,    'Time    (mS)',    'Average    POM_5V_IN    Power    Consumption
(mW)',"Tegrastats_AVG",1),
     (shelly_csv_file, 'timestamp', 'power',"Shelly",0)



        # Ajouter d'autres fichiers et colonnes selon le besoin et ne pas oublier le
skiprows a la fin
]

file_info_list_1 = [
    (training_log, 'Step', 'Training Loss',"Loss",0)

        # Ajouter d'autres fichiers et colonnes selon le besoin et ne pas oublier le
skiprows a la fin
]


output_file = os.path.join(output_path, 'power_consumption_plot.png')
```

```
output_file_1 = os.path.join(output_path, 'Loss.png')

plot(file_info_list, output_file)
plot_loss(file_info_list_1, output_file_1)
```

Code 9 : fonction de tracage

4. Contrôle CPU/GPU

Nous avons créé un script Bash nommé set_freq.sh qui permet de gérer les performances du CPU et du GPU. Il requiert des privilèges root pour fonctionner. Le script offre plusieurs fonctions : il peut activer ou désactiver un cœur de CPU, régler la fréquence maximale du CPU et du GPU en pourcentage de leur capacité maximale, et maximiser les performances globales du système en utilisant l'outil jetson_clocks. Les utilisateurs peuvent choisir entre maximiser les performances, minimiser les fréquences, ou ajuster manuellement les fréquences en pourcentage pour des cœurs CPU spécifiques et pour le GPU.

Nous avons aussi créé un autre script nommé state.sh qui permet d'obtenir les fréquence maximum d'utilisation des processeurs, que nous avons inclut dans le main

III. Choix du Model [1]

1. Les Donnée : Séries Temporelles

La détection d'anomalies dans le contexte de la surveillance de la santé des structures repose sur l'analyse des données de séries temporelles, qui sont des ensembles de points de données recueillis ou enregistrés à des intervalles successifs de temps. Cette méthode est cruciale pour observer et comprendre l'évolution des indicateurs de performance d'une structure, tels que la contrainte, la déformation, ou les vibrations au fil du temps. Les séries temporelles permettent de détecter des comportements non conformes ou inhabituels qui pourraient signaler une dégradation ou une anomalie dans la structure, facilitant ainsi une intervention préventive.

2. OmniAnomaly

OmniAnomaly est une architecture de réseau de neurones récurrent conçue pour l'identification des anomalies dans les séries temporelles multivariées, créée par des chercheurs de l'Université Tsinghua. Cette méthode repose sur l'hypothèse que les anomalies sont des événements rares et inattendus qui se distinguent nettement des schémas habituels observés dans la majorité des données. L'approche

fondamentale d'OmniAnomaly est de former des représentations latentes solides qui reflètent les tendances régulières des données de séries temporelles multivariées, en intégrant les aspects de dépendance temporelle et de stochasticité des séries. Par conséquent, plus les données s'éloignent de ces schémas réguliers, plus elles sont susceptibles d'être identifiées comme étant anormales. Ainsi, OmniAnomaly met en avant un réseau neuronal récurrent doté de variables stochastiques qui respecte une dépendance temporelle bien définie entre ces variables, permettant de détecter efficacement les anomalies.

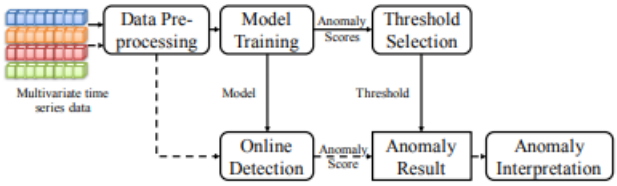


Figure 2 Data pipeline [1]

Comme le montre la figure 2, la structure globale d'OmniAnomaly se compose de deux parties : la formation hors ligne et la détection en ligne.
Le prétraitement des données est un module commun à la formation hors ligne et à la détection en ligne. Au cours du prétraitement des données, l'ensemble de données est transformé par la normalisation des données, puis il est segmenté en séquences par le biais de fenêtres coulissantes de longueur T + 1. Après le prétraitement, une série temporelle multivariée d'entraînement, couvrant généralement une période de temps (par exemple, quelques semaines), est envoyée au module d'apprentissage du modèle pour apprendre un modèle qui capture les modèles normaux des séries temporelles multivariées et produit un score d'anomalie pour chaque observation. Ces scores d'anomalie sont utilisés par le module de sélection de seuil pour choisir automatiquement un seuil d'anomalie selon la méthode POT. Cette procédure de formation hors ligne peut être effectuée régulièrement, par exemple une fois par semaine ou par mois.

3. VAE : Variational Auto Encoder

Dans cette section, nous expliquons comment OmniAnomaly fonctionne pour la détection d'anomalies en visualisant les représentations de l'espace Z, l'espace latent. OmniAnomaly est un modèle basé sur la reconstruction des données d'entrainement. Pour une observation d'entrée, OmniAnomaly la compresse en une représentation de l'espace z de faible dimension, puis utilise cette représentation pour la reconstruire. Pendant l'apprentissage du modèle, OmniAnomaly apprend les représentations des comportements normaux des données d'apprentissage. Si une observation d'entrée est anormale, sa représentation dans l'espace z et la valeur reconstruite sont toujours normales, de sorte que la probabilité de reconstruction est faible.

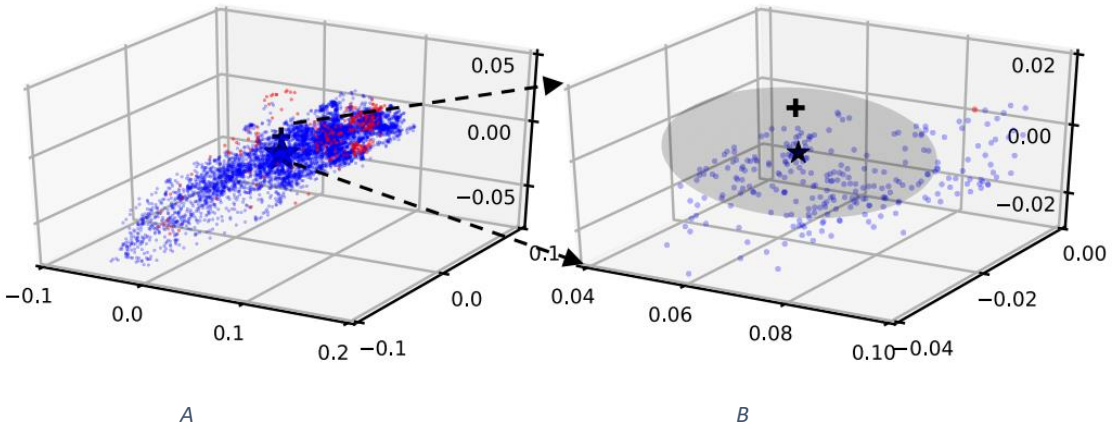


Figure 3 Espace latent z [1]

Dans la figure A, l'espace latent tridimensionnel (espace-z) généré par l'apprentissage sur un ensemble de données de surveillance de machines (SMD) est illustré. L'observation révèle que les points représentant les anomalies se superposent considérablement avec ceux des données normales, suggérant que leurs représentations latentes sont similaires dans l'espace-z. Cela est exemplifié par une expérience où, si nous prenons une observation normale et modifions artificiellement ses valeurs pour simuler une anomalie (comme illustré dans la figure 1), les représentations latentes de l'observation originale et de sa version modifiée restent proches l'une de l'autre dans l'espace-z. Ce phénomène met en évidence que, même en présence d'anomalies, les variables latentes apprises par OmniAnomaly tendent à maintenir un caractère 'normal'. Ces constatations démontrent l'aptitude d'OmniAnomaly à capturer de manière fiable les motifs standards des observations dans les variables de l'espace-z, même lorsque des données anormales sont introduites.

4. Evaluation

Pour évaluer l'efficacité d'OmniAnomaly dans la détection d'anomalies au sein de séries temporelles multivariées, l'étude en question a réalisé des comparaisons avec quatre techniques de pointe non supervisées : LSTM avec seuillage dynamique non paramétrique (LSTM-NDT), EncDec-AD, DAGMM, et LSTM-VAE. D'après les résultats illustrés dans la figure 4 de l'article, OmniAnomaly dépasse les performances de ces méthodes sur les ensembles de données MSL et SMD en termes de score F1, qui est une mesure harmonique de la précision et du rappel. Bien que sur l'ensemble de données SMAP, son score F1 soit légèrement inférieur à celui de la meilleure méthode de référence, OmniAnomaly présente une amélioration significative par rapport au modèle LSTM-NDT, augmentant le score F1 de 0,09 pour l'ensemble de données Total. En termes de robustesse, OmniAnomaly affiche une précision et un rappel supérieurs à 0,74 sur les trois ensembles de données testés, ce qui démontre une constance dans la performance que les autres modèles de référence ne parviennent pas à atteindre. Cela suggère que OmniAnomaly est non seulement compétitif, mais aussi plus stable et fiable pour la détection d'anomalies dans divers contextes de séries temporelles multivariées.

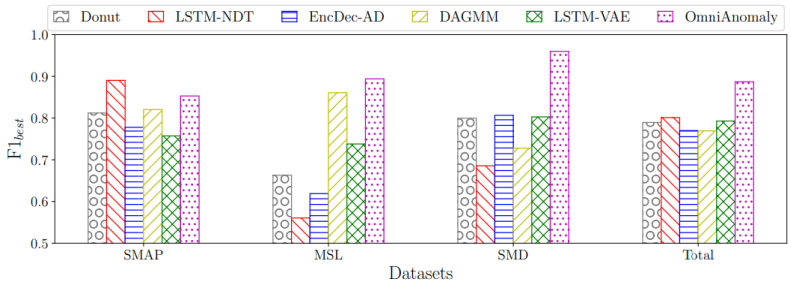


Figure 4 Score F1 dans l'article source [1]

5. Intégration dans notre architecture

Initialement, notre modèle exigeait des ressources qui dépassaient les capacités du GPU, ce qui s'est manifesté par des erreurs de manque de mémoire ou Out of memory en anglais (OOM) lors de l'allocation de tenseurs.

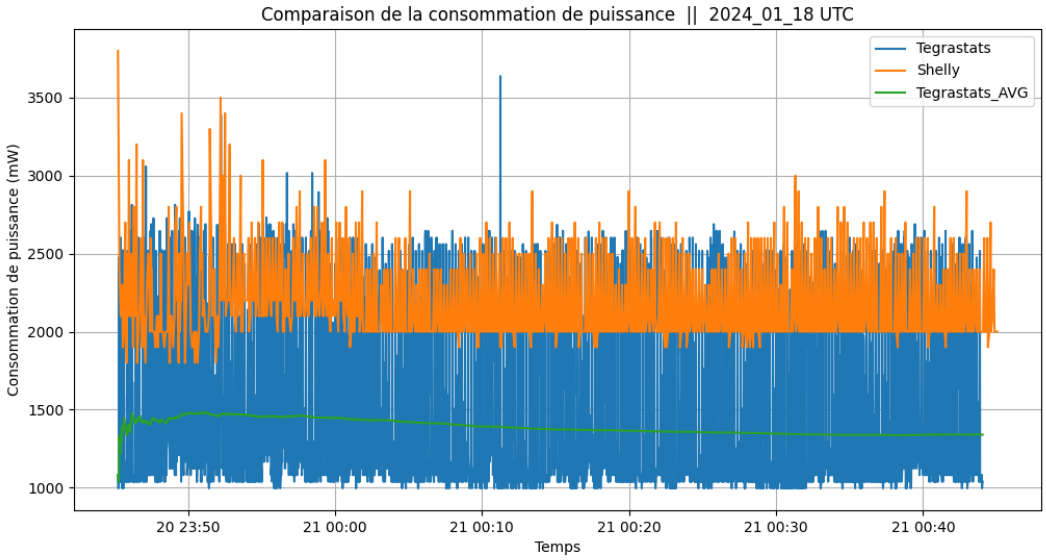
Face à cette contrainte, nous avons dû adapter notre approche pour optimiser l'utilisation des ressources disponibles. Après plusieurs itérations, nous avons finalement opté pour une taille de batch réduite à 40, une longueur de fenêtre de série temporelle ajustée à 50, et une dimension de l'espace latent Z fixée à 3. Ces ajustements ont permis au modèle d'être entraîné efficacement sur la Jetson Nano sans dépasser les limites de sa mémoire et de ses capacités de traitement.

De plus, nous avons implémenté une modification dans la boucle d'entraînement pour récupérer et enregistrer la valeur de la perte (loss) à chaque étape. Cette adaptation nous a non seulement aidés à surveiller l'efficacité de l'entraînement en temps réel, mais a également fourni des insights précieux sur la performance du modèle tout au long du processus d'entraînement.

IV. Résultats

1. Comparaison des différents modes de fonctionnement

Consommation en IDLE



Fi gure 5 évolution de la puissance en IDLE

L'analyse de la figure 5 révèle que l'outil tegrastats, spécifique aux appareils NVIDIA Jetson, capture des variations fines dans la consommation d'énergie du dispositif, illustrées par la série de données bleue avec des fluctuations significatives. Cela indique que tegrastats est sensible aux changements rapides et mineurs de consommation, reflétant probablement les variations instantanées de charge du processeur et du GPU.

En revanche, la prise connectée Shelly, représentée par la série de données orange, montre un pas de mesure plus grand, avec des variations moins fréquentes et plus arrondies. Les données de Shelly semblent indiquer un palier minimal autour de 2W, ce qui pourrait suggérer une consommation d'énergie de base ou un seuil de détection de la prise connectée.

La ligne verte, Tegrastats_AVG, qui représente la moyenne des données tegrastats, se situe majoritairement en dessous de 1,5W, dénotant que, malgré les pics temporaires de consommation enregistrés par tegrastats, la consommation d'énergie moyenne reste relativement faible. Cette moyenne peut être considérée comme une meilleure représentation de la consommation énergétique typique du dispositif sur la période observée.

En somme, l'image montre que, bien que la prise connectée Shelly puisse être utile pour surveiller la consommation d'énergie globale sur des intervalles plus longs et avec moins de précision, l'outil tegrastats fournit une vue plus détaillée et instantanée de la performance énergétique, capturant des nuances que

Shelly pourrait ne pas détecter. L'utilisation combinée des deux méthodes offre un aperçu complet, avec tegrastats illustrant la consommation énergétique instantanée et détaillée et Shelly fournissant une évaluation de la consommation sur de plus longues périodes.

ACTIF sans limitation de step et limitation à 20 Epoch

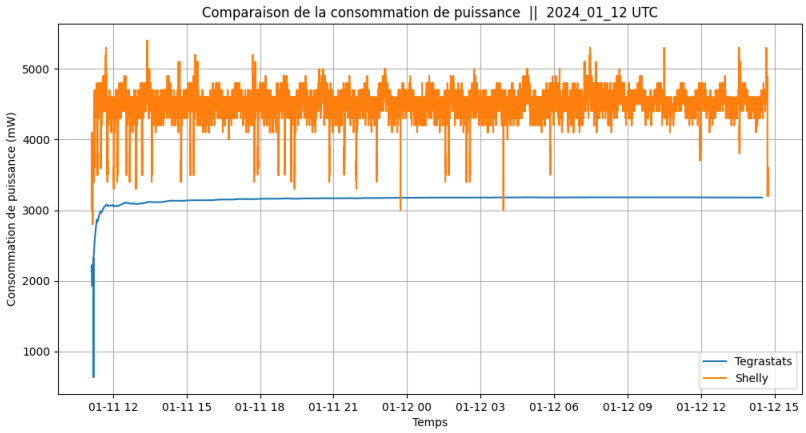


Figure 6 évolution de la puissance à 537600000 Hz

Dans la figure 6 les processeurs sont actifs à une fréquence maximale de 1,1 MHz et le GPU fonctionne à une fréquence de 0.5GHz soit 60% de la valeur max, La consommation d'énergie augmente initialement, puis se stabilise a 5W pour Shelly mais tegrastat montre une consommation moyenne de 3W, ce qui reflète une période de montée en charge où l'appareil commence à exécuter des tâches exigeantes, comme l'entraînement d'un modèle d'IA, avant de se stabiliser.

Sachant une epoch correspond à 1000 steps et 1 heure, à fréquence max il aurait fallu 20 h donc 33h pour 0.5Ghz, mais on arrêté après une journée.

ACTIF avec les fréquences max et une limitation 100 steps

Sur la figure 7, les données de Tegrastats et Shelly montrent une consommation d'énergie variable avec des pics significatifs qui coïncident avec les périodes d'utilisation intensive du GPU, comme le démontre la figure 8 où l'utilisation du GPU atteint régulièrement 100%. Ces observations sont en corrélation avec les fréquences élevées rapportées pour GPU, indiquant que le système est soumis à une charge de travail importante en raison de l'entraînement du modèle d'intelligence artificielle, qui exige une utilisation maximale des ressources de calcul. La courbe de moyenne sur le graphique de consommation d'énergie (Tegrastats_AVG) offre une vue d'ensemble des tendances sur la période observée, soulignant la capacité

du système à maintenir un niveau de performance élevé tout en gérant les fluctuations de la demande énergétique

Le premier palier en dessous de 5W correspond au processing des données, le second correspond à l'entrainement du model, on observe ensuite une chute qui pourrait correspondre à la fin de traitement d'un batch de données et le traitement du batch suivant correspond à la remontée de la courbe. Enfin le dernier palier correspond à la phase de test qui dure environs 20 minutes.

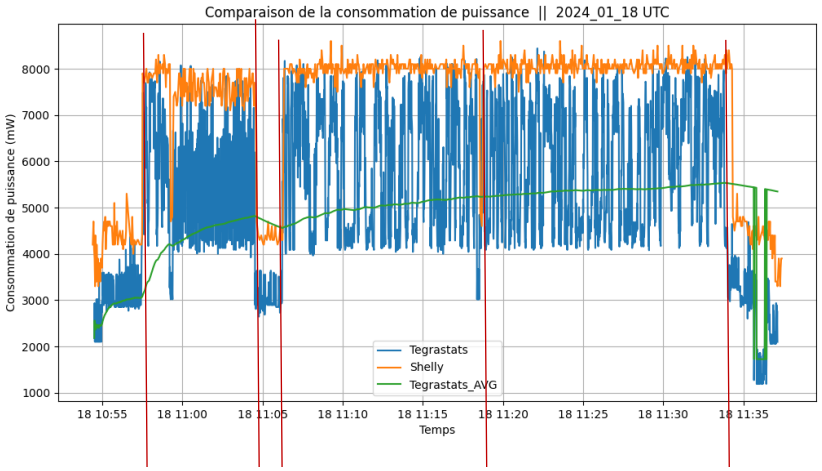


Figure 7 évolution de la puissance à 921600000 Hz

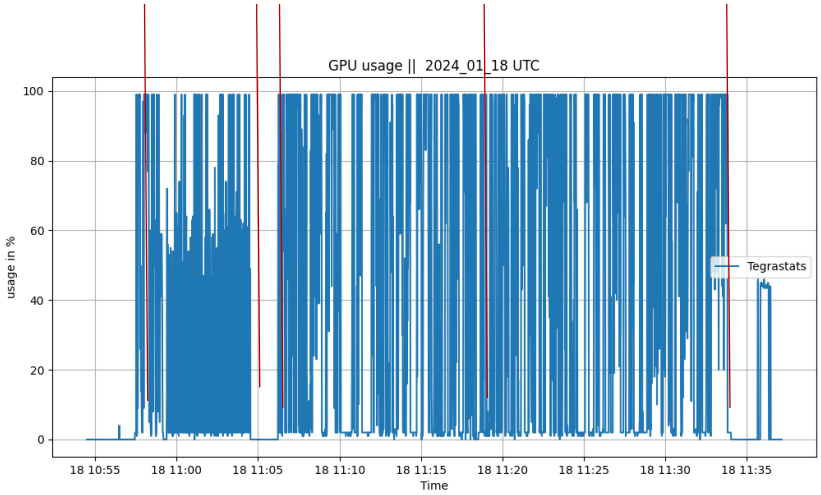


Figure 8 Utilisation du GPU

En conclusion, entre le mode IDLE et le fonctionnement max on a 5.5W-1.5W=4W (Puissance moyenne) nécessaire pour faire l'entrainement et l'Energie consommée vas être fonction du temps E=4W*t , il est donc important de minimiser les temps d'entrainement

2. Résultats d’entrainement

- **100 steps**

La figure 9 montre une tendance décroissante, indiquant une amélioration et un apprentissage du modèle, comme en témoigne la réduction constante de la perte. Cette phase est caractéristique d'un processus d'optimisation réussi où le modèle affine progressivement ses paramètres pour mieux s'ajuster aux données.

Cependant, vers la fin de l'entraînement, il y a un pic de perte exceptionnellement élevé qui pourrait suggérer un événement atypique dans le processus d'entraînement. Ce pic peut être attribué à divers facteurs, tels qu'une anomalie dans les données d'entraînement, un choix inapproprié du taux d'apprentissage qui aurait pu causer une divergence temporaire, ou un problème transitoire dans l'optimisation.

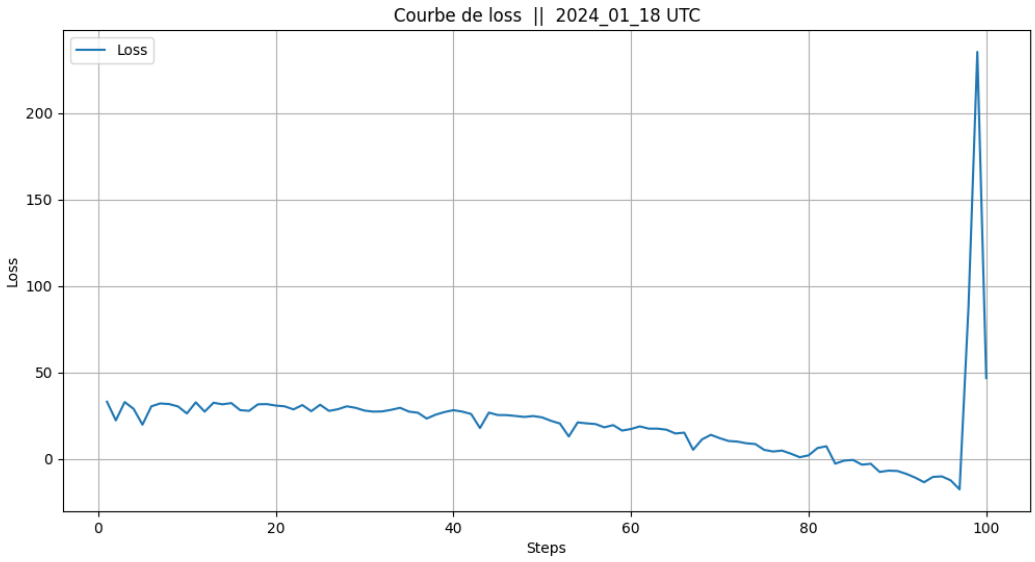


Figure 9 Courbe de Loss à 100 Steps

Un score F1 proche de 1 indique une performance élevée du modèle, avec un bon équilibre entre précision et rappel. Nous avons pour 100 steps un score F1 de 0.86 qui est généralement considéré comme très bon dans de nombreux contextes d'apprentissage automatique, surtout dans des tâches complexes comme la détection d'anomalies où les données peuvent être très déséquilibrées ou bruitées. Dans l’article source le score F1 était de 0.9 pour 5000 steps, on peut donc limiter le nombre de steps pour économiser de l’Energie, mais avoir quand même de bonne performances.

- 5000 Steps

Les figure 11 et 10 capturent des aspects clés du fonctionnement d'un système embarqué durant un processus d'entraînement intensif d'un modèle d'IA. Le premier graphique montre une consommation d'énergie avec une moyenne stable, indiquant un fonctionnement régulier du système, et des pics qui coïncident avec des activités de calcul intensives pendant 3.5 heures. La figure 11 révèle une tendance générale à la baisse de la perte d'entraînement sur 5000 étapes, signe d'une amélioration continue du modèle, malgré la présence de pics de perte sporadiques reflétant des instabilités ponctuelles, on a supprimé les valeurs trop aberrantes pour pouvoir voir la forme de la courbe (on avait des valeurs à -90000000 ; on a limité à -300). Ces informations sont essentielles pour évaluer l'efficacité de l'entraînement et l'optimisation énergétique du système, deux considérations critiques pour la performance et la durabilité des applications d'IA embarquées.

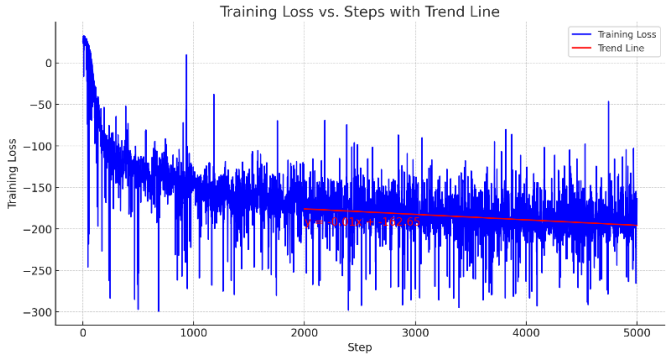


Figure 11 Courbe de Loss

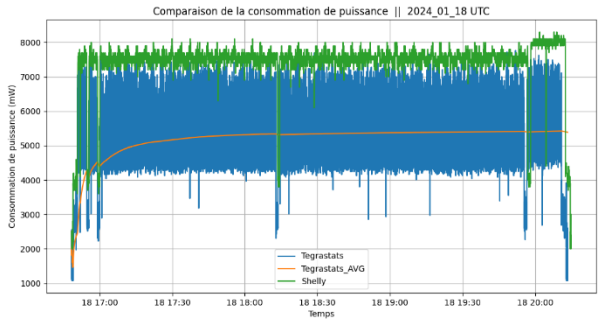


Figure 10 évolution de la puissance pour 5000 Steps

En comparaison avec la figure 12 obtenue dans l'article source on se rend compte qu'on a les même tendance et une stabilisation à -180, nous obtenons un score F1 de 0.888 ce qui est un bon score et proche des 0.9 de l'article.

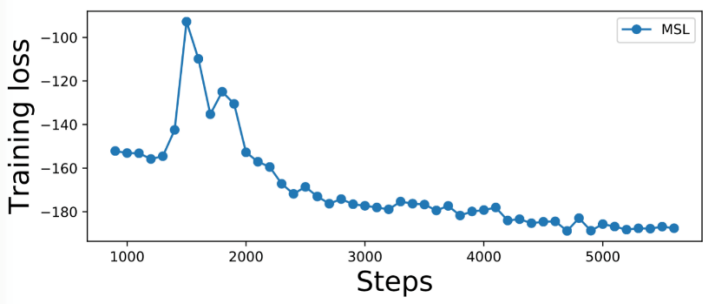


Figure 12 Courbe de loss de l'article source

- **1000 steps**

Pour une limite de 1000 steps (illustré par la figure 13) une stabilisation de la loss à -150, un entrainement en 1 heure (illustré par la figure 14) et un score de F1 à 0.88, ce qui nous amène à nous dire que 1000 steps est un bon équilibre pour ce modèle.

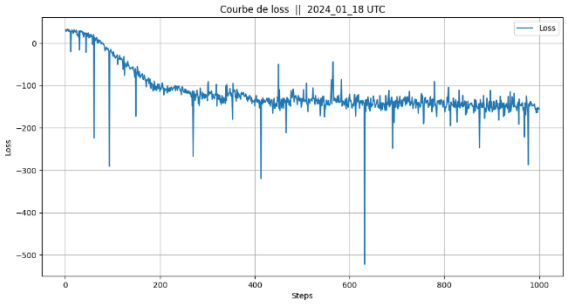


Figure 13 Courbe de loss pour 1000 steps

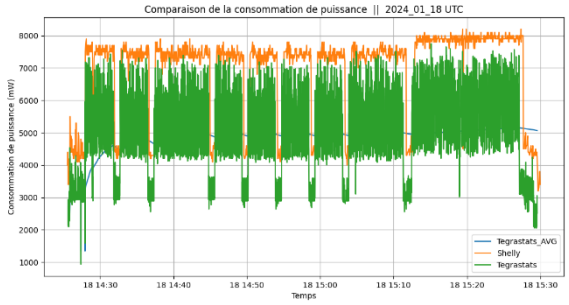


Figure 14 évolution de la puissance pour 1000 Steps

V. Synthèse

Pour initialiser l'environnement de la Jetson Nano en vue de l'entraînement d'un modèle d'IA, les commandes suivantes doivent être saisies dans le terminal après l'installation préalable de Git :

git clone https://github.com/Melvyn212/Edge-AI-For-SH y && ./Edge-AI-For-SHM/ansible/set_up_jnano.sh

./Edge-AI-For-SHM/DockerRun/DockerRun.sh

La première ligne de commande clone le dépôt nécessaire et lance un script de configuration Ansible pour préparer la Jetson Nano. La seconde exécute un script qui démarre l'environnement d'entraînement dans un conteneur Docker, installant toutes les dépendances requises et configurant le système pour le processus d'entraînement.

Problèmes rencontrés

Au cours du développement de notre projet, plusieurs défis significatifs ont été rencontrés, impactant le calendrier prévu et nécessitant des adaptations stratégiques de notre méthode de travail.

Accès aux Ressources Matérielles : L'un des principaux obstacles a été l'accès intermittent aux dispositifs NVIDIA Jetson Nano. Des interruptions fréquentes dans la connexion SSH et des dysfonctionnements récurrents du serveur 'Khamul' ont considérablement entravé notre capacité à travailler de manière

continue et efficace. Ces problèmes de connectivité ont non seulement retardé le développement et l'entraînement des modèles mais ont aussi limité notre capacité à expérimenter et à itérer rapidement.

Défis Techniques avec le Modèle d'IA : Le modèle d'intelligence artificielle choisi pour ce projet présentait ses propres défis techniques en raison de son âge et de sa dépendance à une version antérieure de TensorFlow (TensorFlow 1). L'ancienneté du modèle a entraîné des complications dans l'intégration avec les environnements de développement modernes, qui sont principalement optimisés pour TensorFlow 2. Des plateformes de développement couramment utilisées comme Google Colab ou SaturnCloud n'étaient pas compatibles avec notre Framework, limitant nos options pour l'entraînement du modèle.

Conclusion

Ce projet a été un voyage intensif à travers les défis et les opportunités présentés par l'Intelligence Artificielle Embarquée. Malgré les obstacles techniques et logistiques, tels que l'accès limité aux ressources matérielles et les limitations de compatibilité liées à l'utilisation de TensorFlow 1, nous avons démontré notre capacité à nous adapter et à innover. En implémentant directement nos solutions sur les Jetson Nano, nous avons non seulement surmonté les contraintes des plateformes de développement mais aussi optimisé nos modèles pour une efficacité énergétique accrue.

Les résultats obtenus illustrent l'efficacité de nos modèles d'IA, avec des scores F1 approchant ceux des recherches actuelles malgré une limitation du nombre de steps, ce qui souligne la pertinence de nos méthodes pour des applications pratiques en Intelligence Artificielle Embarquée. La prochaine étape de ce projet est la mise en place d'une architecture pour un apprentissage fédéré, que l'on n'a malheureusement pas eu le temps de faire. Les enseignements tirés de ce projet guideront nos futures initiatives dans le domaine de l'IA embarquée et contribueront à l'avancement de cette technologie.

Bibliographie

[1] S. YA, «Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network,» ACM, Anchorage AK USA, 2019.