

# Rapport — Adapter-Based Fine-Tuning (Whisper-small) — ASR Fellowship Challenge

## 1. Informations personnelles

- Nom : NONO NGANSOP NAHOMIE MADELEINE
- Email / Contact : nahomie.nono@facscience-uy1.cm

## 2. Description de l'expérience

- Modèle de base : openai/whisper-small
- Dataset : DigitalUmuganda/ASR\_Fellowship\_Challenge\_Dataset (Afrivoice\_Kinyarwanda health subset)

## 3. Architecture des adaptateurs

Les adaptateurs sont des modules légers insérés dans un modèle pré-entraîné afin d'ajuster le modèle à un nouveau domaine ou langage **sans modifier les poids originaux**. Dans notre cas, nous utilisons Whisper-small comme modèle de base et insérons les adaptateurs de manière stratégique dans l'encodeur et le décodeur.

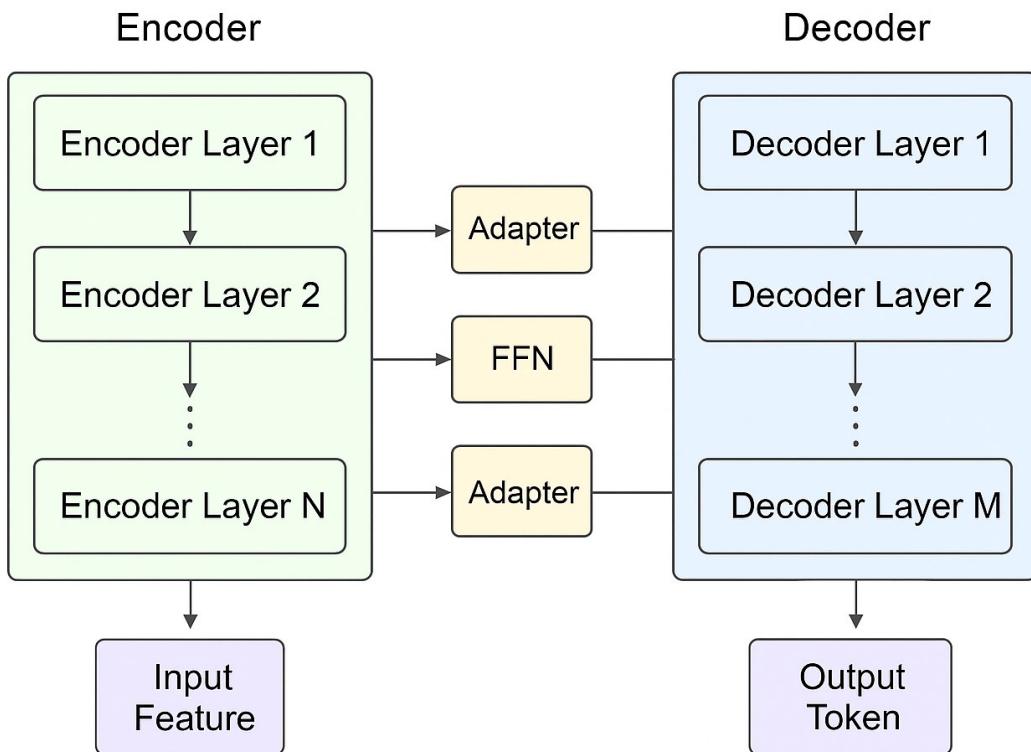
### 3.1 Structure du module adaptateur

- **Topologie bottleneck (linéaire)** :
  1. **Down-projection** : réduit la dimension  $d_{model}$  à un espace latent de dimension réduite  $r$  (bottleneck).
  2. **Activation non-linéaire** : ReLU.
  3. **Up-projection** : remonte à la dimension originale  $d_{model}$ . L'adaptateur commence avec une contribution nulle.
- **Résidu** : sortie de l'adaptateur ajoutée à la sortie de la couche originale pour maintenir le flux d'informations du modèle de base.

### 3.2 Emplacement dans Whisper

- **Encodeur** : après chaque Feed-Forward Network (FFN) de chaque couche, on insère un adaptateur.
- **Décodeur** : de même, après chaque FFN de chaque couche.
- Les couches d'attention multi-fête **ne sont pas modifiées**. L'apprentissage se concentre uniquement sur les adaptateurs.

### 3.3 Schéma conceptuel



### 3.4 Avantages

- Paramètres entraînables réduits** : seul un petit sous-ensemble du modèle est optimisé (~1-5% des paramètres totaux).
- Préservation des connaissances pré-entraînées** : gel des poids de Whisper-small empêche l'oubli catastrophique.
- Flexibilité** : facile d'insérer des adaptateurs supplémentaires pour différents domaines ou langues sans modifier la base.

---

## 4. Stratégie d'entraînement

- Geler le modèle de base** : tous les poids de Whisper sont non entraînables (`requires_grad = False`).
- Activer uniquement les adaptateurs** :
  - Les poids des modules adaptateurs (`w_down`, `w_up`, `b_down`, `b_up`) sont entraînables.

- Cela concentre l'optimisation sur l'adaptation à la langue Kinyarwanda sans toucher aux connaissances générales de Whisper.

### 3. Préparation des données :

- Les fichiers audio .webm sont extraits du tarball et resamplés à 16 kHz (Whisper requirement).
- Les features sont extraites avec `WhisperProcessor.feature_extractor`.
- Les transcriptions (`text`) sont tokenisées pour produire les labels.

### 4. Optimisation :

- Optimiseur : AdamW
- Scheduler linéaire avec warmup
- Loss : CrossEntropy sur les tokens
- Batch size : 8, epochs : 3, learning rate : 3e-4

5. Sauvegarde : après l'entraînement, seuls les poids des adaptateurs sont enregistrés.

## 5. Résultats attendus

- WER modèle base : ...
- WER modèle affiné : ...
- Différence WER : ...
- Nombre de paramètres entraînables : ...

## 6. Reproductibilité & Instructions (pas à pas)

1. Installer les dépendances: `pip install -r requirements.txt`
2. Télécharger le dataset (si besoin) et exécuter l'entraînement: `python src/train.py --model_name openai/whisper-small --batch_size 8 --num_epochs 3 --adapter_dir ./adapters`
3. Générer transcriptions: `python src/evaluate.py --model_name openai/whisper-small --adapter_path None --split test --out base_transcriptions.txt` `python src/evaluate.py --model_name openai/whisper-small --adapter_path ./adapters/adapter_weights.pth --split test --out finetuned_transcriptions.txt`

## 7. Conclusion & prochaines étapes

- Suggestions: data augmentation, LoRA, PEFT integration, validation split, scheduler tuning.