**Ce qu’on attend de nous à la fin du projet** ce que nous soyons capable de démontrer notre **capacité à concevoir un système complet de reconnaissance de la parole** dans une langue peu dotée (ressource faible), en nous appuyant sur des **modèles de Deep Learning séquentiels** (type RNN, LSTM ou GRU), à partir de données audios brutes issue du dataset Yembatons. Par exemple Si **l’utilisateur prononce** « *aleko’o* », le système doit **écouter l’audio**, l’**analyser**, et **prédire le texte correspondant** : "aleko’o". C’est du speech to text A la fin nous devons avoir :  
**1. Un système qui prend en entrée un fichier audio (ex. quelqu’un qui dit « bonjour » en yemba) et qui produit le mot écrit correspondant.**

**2. Une explication claire de ton processus : comment tu as préparé les données, choisi le modèle, entraîné le système, évalué les résultats.**

**3. Un modèle entraîné et, si possible, une interface ou API pour interagir avec ton système (web ou CLI).**

**4. Une documentation complète qui explique la logique de ton projet, tes choix, et comment le reproduire.**

**5. Des réflexions sur les limites, les biais et les perspectives d’amélioration (ex. étendre à d’autres dialectes, améliorer avec plus de données, etc.).**

**Etapes**

**1. Compréhension du domaine et des données**

* **Lire sur la reconnaissance automatique de la parole (ASR).**
* **Comprendre la structure du dataset Yembatones :**
  + **Est-ce des paires *audio + transcription* ?**
  + **Combien de locuteurs ? Combien d’heures ?**
  + **Format des fichiers (WAV, MP3, etc.) ?**
* **Identifier les unités cibles : reconnaissance de mots entiers, de phonèmes, ou de caractères ?**

**2. Prétraitement des données (data preprocessing)**

* **Nettoyage :**

**Pretraitement des lignes 153-156 par ajout de nouvelle valeurs manquantes sur les colones syllabe 1, ton 1, syllabe 2, ton 2**

* + **Supprimer les données corrompues, bruitées ou mal transcrites.**
* **Extraction de caractéristiques audio :**
  + **Convertir les fichiers audio en spectrogrammes ou MFCC (Mel Frequency Cepstral Coefficients).**
* **Tokenisation du texte :**
  + **Construction du vocabulaire (ex : liste de mots ou caractères).**
  + **Encodage des cibles (via LabelEncoder, CTC, etc.).**
* **Séparation du dataset :**
  + **Train / Validation / Test (80/10/10 par exemple).**

**3. Conception et choix du modèle**

* **Choix des architectures :**
  + **RNN simple (baseline), LSTM, GRU.**
  + **Modèle séquence-à-séquence (seq2seq) ou CTC (Connectionist Temporal Classification) si pas d’alignement audio/texte.**
* **Choix du framework :**
  + **PyTorch ou TensorFlow (je te guiderai).**
* **Architecture typique :**
  + **CNN (extraction features) + BiLSTM + CTC Loss.**

**4. Entraînement du modèle**

* **Choix des hyperparamètres :**
  + **Learning rate, nombre d’époques, batch size, etc.**
* **Suivi de l’apprentissage :**
  + **Courbes de perte (loss), métriques de validation.**
  + **Early stopping, régularisation, dropout.**

**5. Évaluation des performances**

* **Utiliser des métriques adaptées à l’ASR :**
  + **WER (Word Error Rate).**
  + **CER (Character Error Rate).**
* **Analyser les erreurs :**
  + **Quels types de mots sont mal reconnus ?**
  + **Problèmes de prononciation, longueur, etc.**

**6. Interface utilisateur (facultatif mais valorisant)**

* **Créer une petite application ou interface CLI :**
  + **Upload d’un audio → affichage de la transcription.**
  + **Utilisation de Gradio, Flask, ou une interface simple.**

**7. Documentation du projet**

* **Rédiger :**
  + **Un rapport clair (markdown, PDF, ou notebook).**
  + **Un README si tu publies sur GitHub.**
* **Inclure :**
  + **Description du dataset.**
  + **Architecture du modèle.**
  + **Explication des choix.**
  + **Résultats et évaluation.**
  + **Limites et pistes d’amélioration.**

**8. Mise en production (optionnel/bonus)**

* **Sauvegarde du modèle (.pt ou .h5).**
* **Chargement pour inférence rapide.**
* **Déploiement avec une API (Flask/FastAPI) ou un site web léger.**
* **Possibilité de partage du modèle via Hugging Face ou autres plateformes.**

**🧪 Ce que nous apprendras en chemin**

**✅ Prétraitement audio et NLP  
✅ Construction de modèles séquentiels  
✅ Traitement de langues à faible ressource  
✅ Évaluation avec WER/CER  
✅ Intégration et déploiement d’un projet réel**

Possibilite d’ameloration

**✅ 1. GRU (Gated Recurrent Unit) + CTC**

**Pourquoi ?**

* Moins de paramètres que les LSTM → **meilleure généralisation avec peu de données**
* Moins sensible à l’overfitting que BiLSTM sur petits corpus

**Architecture type :**

text

Copier le code

Input (Mel spectrogram) → GRU (1 ou 2 couches, bidirectionnelles) → Linear → LogSoftmax → CTC

**✅ 2. Stacked GRU + Attention (sans CTC)**

**Pourquoi ?**

* L’attention peut mieux capturer les relations tonales ou syllabiques longues
* Fonctionne bien avec des séquences de longueurs variables
* Permet une **décodage séquence-à-séquence**, utile pour capturer des relations complexes entre tonalité + phonèmes

**Architecture type :**

text

Copier le code

Encoder (2–3 couches GRU bidirectionnelles)

→ Attention Mechanism

→ Decoder GRU (mono-directionnel)

→ Linear + Softmax

→ CrossEntropy Loss

⚠️ Ici, tu abandonnes le CTC pour une approche **encoder-decoder avec attention**, donc tu dois aligner entrées/sorties.

**✅ 3. Hierarchical RNNs (pour syllabes + tons séparés)**

**Pourquoi ?**

* Tu peux exploiter la **structure hiérarchique** de la langue tonale : phonème → syllabe → ton

**Exemple :**

* Premier GRU pour encoder les **caractéristiques phonétiques**
* Deuxième GRU pour **prédire les tons associés**
* En sortie : concaténation ou double tête de prédiction (phonème + ton)

**✅ 4. CTC avec embeddings tonals explicites**

Si tu restes en CTC, tu peux **injecter la tonalité** comme **information explicite** :

* Ajouter une entrée supplémentaire par frame (canal tonal)
* Ou utiliser un feature fusion de spectrogramme + vecteur de ton détecté

**✅ 5. RNN-T (Recurrent Neural Network Transducer)**

* Variante améliorée de CTC
* Combine un **encodeur (ex : LSTM)**, un **préfixe prédicteur (type language model)**, et un **joint network**
* Meilleure performance que CTC classique
* Prise en charge du streaming (utile pour déploiement)

**🧪 Recommandation pour ton cas concret :**

| **Objectif** | **Architecture conseillée** |
| --- | --- |
| Petit corpus, données tonales | GRU bidirectionnel + CTC |
| Capturer tons/phonèmes séparément | Hiérarchique GRU |
| Exploiter alignement implicite | GRU Encoder-Decoder + Attention |
| Approche plus avancée | RNN-T |
| Restreint à CTC mais plus robuste | CNN + GRU + CTC (au lieu de BiLSTM) |

Piste d’amelioration

**🔍 ÉTAPE 1 : DIAGNOSTIC INITIAL ET NETTOYAGE DU DATASET**

**Objectif :** Éliminer les biais ou artefacts qui nuisent à l’apprentissage

**✅ Actions :**

* Supprimer tous les enregistrements où encoded est vide ou où plus de 70 % des tokens sont ∅|∅
* Vérifier la distribution des classes (syllabe, ton) et les déséquilibres
* Rééquilibrer (via suréchantillonnage ou pondération) si nécessaire
* Normaliser les chemins de fichiers (os.path.normpath()) pour éviter les erreurs

**🔧 Résultat attendu :**

Dataset propre, bien distribué, prêt pour un apprentissage optimal

**🧠 ÉTAPE 2 : OPTIMISATION DU MODÈLE**

**Objectif :** Explorer différentes configurations du GRU+attention

**✅ Expérimentations proposées :**

* Ajouter bidirectional=True au GRU de l’encodeur
* Augmenter hidden\_dim (ex. 256 → 384 → 512)
* Jouer sur le dropout (ex. 0.1, 0.3, 0.5)
* Essayer un MultiHeadAttention PyTorch
* Tester une architecture **encoder bidirectionnel + decoder unidirectionnel** avec attention classique

**🔧 Résultat attendu :**

Meilleure capacité de représentation, compréhension du contexte syllabique-tonal

**🎧 ÉTAPE 3 : AMÉLIORATION DU PRÉTRAITEMENT AUDIO**

**Objectif :** Extraire des features plus informatives

**✅ Améliorations proposées :**

* Ajout de torchaudio.transforms.FrequencyMasking et TimeMasking pendant l’entraînement
* Ajout de delta et delta-delta mels
* Test avec MFCC + pitch en complément
* Padding centré ou ajusté à la longueur réelle

**🔧 Résultat attendu :**

Melspectrogrammes plus robustes à la variabilité des locuteurs et bruits

**🧪 ÉTAPE 4 : PERTE, DÉCODAGE ET MÉTRIQUES**

**Objectif :** Mieux apprendre les séquences et éviter les erreurs courantes

**✅ Actions :**

* Ajouter du label smoothing pour réduire la surestimation
* Penser à une loss mixte (CTC + Seq2Seq) ou remplacer par CTC pur
* Comparer le **greedy decoding** à un **beam search** (k=3 ou 5)
* Évaluer séparément les erreurs de tonalité et de syllabes

**🔧 Résultat attendu :**

Décodage plus stable, prédictions plus précises et plus proches des séquences cibles

**🔄 ÉTAPE 5 : AUGMENTATION DES DONNÉES**

**Objectif :** Enrichir les données audio et texte pour éviter l’overfitting

**✅ Méthodes :**

* Ajouter du bruit blanc, du shift temporel, pitch shift
* Stretch audio (time stretch)
* Génération semi-synthétique de nouveaux échantillons à partir de syllabes existantes
* Mixup audio entre locuteurs proches

**🔧 Résultat attendu :**

Meilleure généralisation, moins de surapprentissage aux locuteurs

**📊 ÉTAPE 6 : SUIVI ET VISUALISATION DES ERREURS**

**Objectif :** Identifier les faiblesses restantes du modèle

**✅ À faire :**

* Générer un rapport .csv des erreurs par type : erreur de ton, de syllabe, d’insertion/suppression
* Visualiser les erreurs avec Spectrogrammes + annotation prédite vs attendue
* Isoler les exemples ambigus pour reannotation ou exclusion