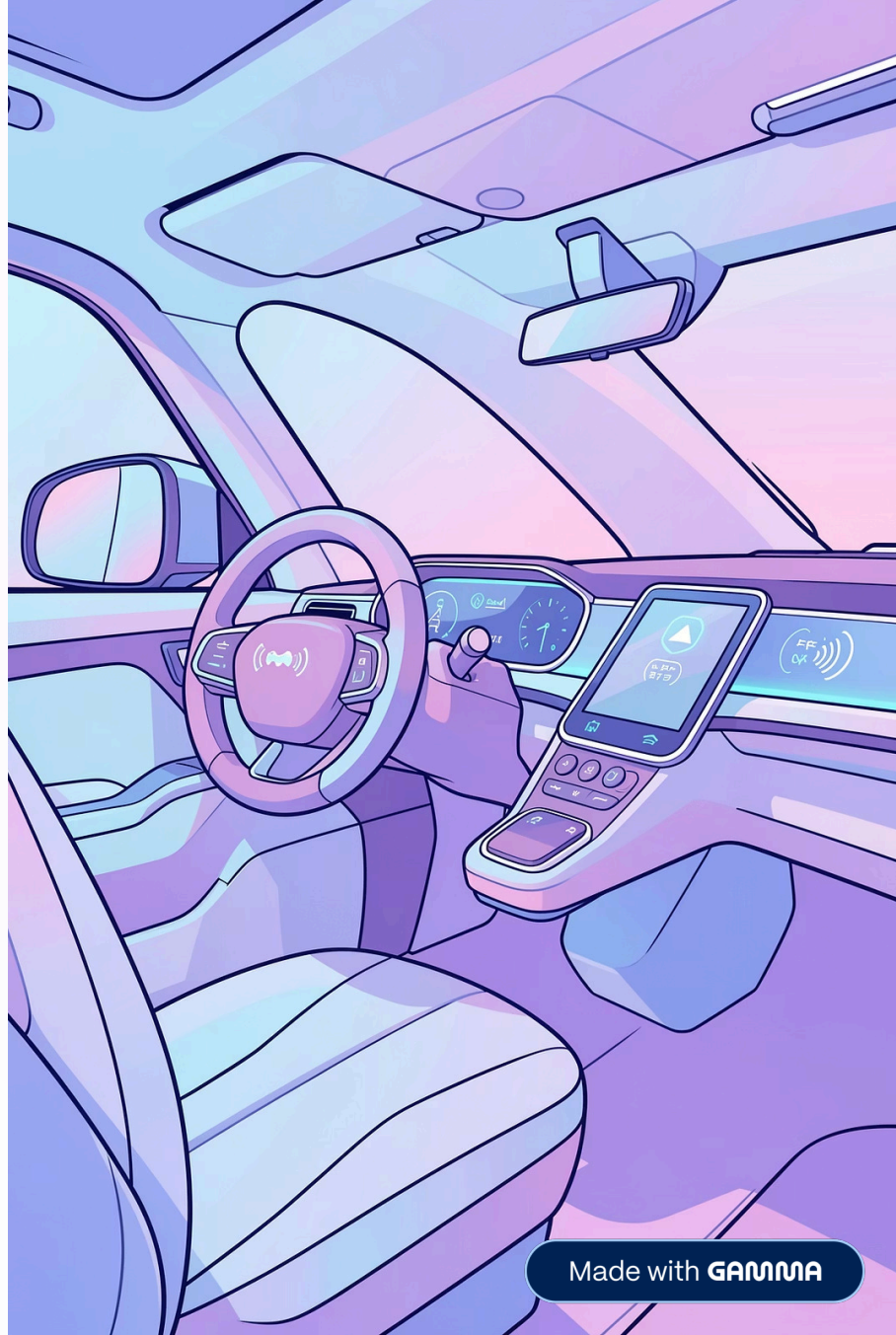
Apprentissage Vocal pour Véhicules Autonomes

Un pipeline complet pour la reconnaissance des commandes vocales en français, optimisé pour les systèmes embarqués.



Objectif du Projet

Commandes Vocales Françaises

Développer un système pour contrôler un véhicule autonome (avancer, reculer, gauche, droite, arrêter).

Optimisation Embarquée

Système conçu pour un déploiement efficace sur des plateformes embarquées.

Déploiement sur Jetson

Déploiement final sur Jetson (TensorRT) pour l'inférence en temps réel.

Création d'une base de données audio personnalisée

Sélection des commandes

- Commandes courtes, adaptées à l'embarqué : avance, recule, gauche, droite, stop.

Enregistrement personnalisé

- Enregistrements manuels (micro PC), Français, 16 kHz, .wav (mono, float32).
- Durée cible : 1 seconde, ≥ 20 prises par classe.

Organisation du jeu de données

Structure standard basée sur les classes pour la compatibilité PyTorch et l'extensibilité.

Augmentation et Prétraitement des Données

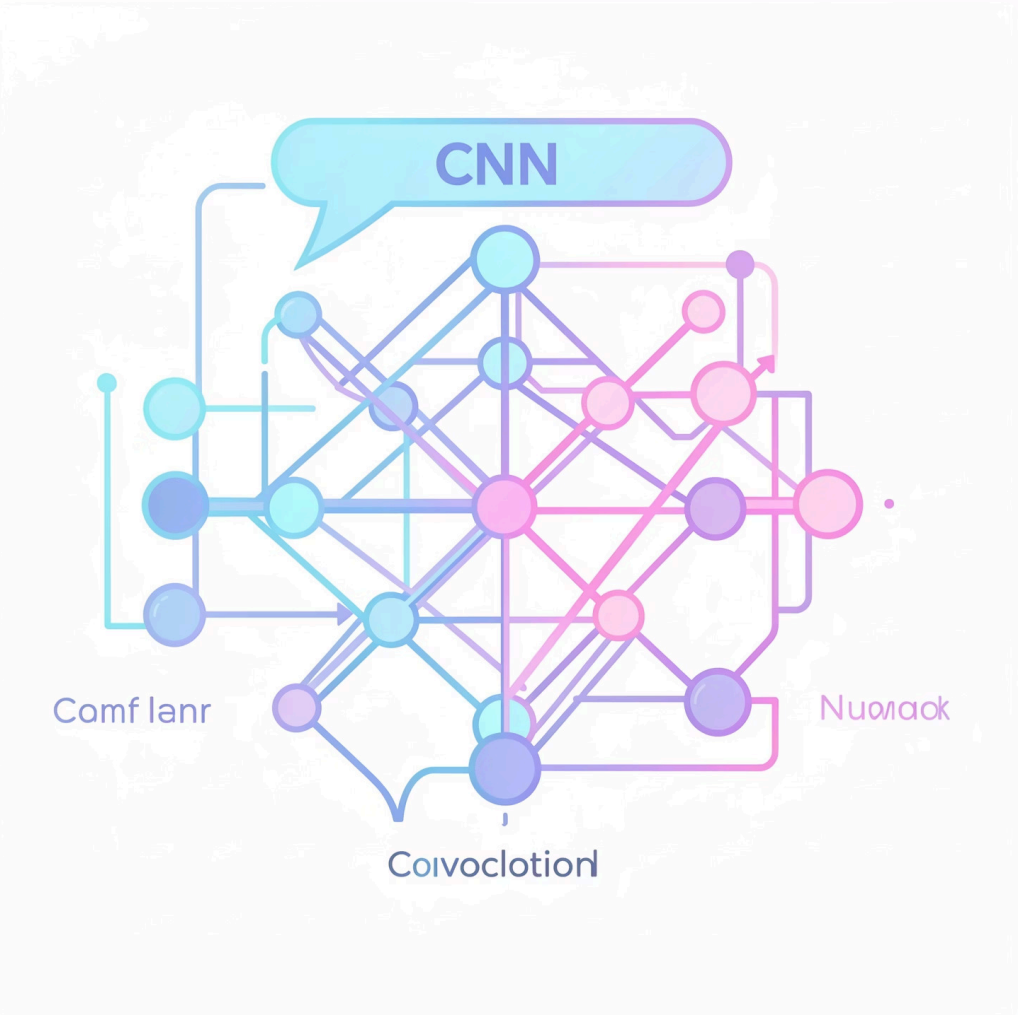
Augmentation des Données

Augmentez la robustesse du modèle avec de légères variations de volume, de hauteur (±2-3 %) et d'étirement temporel. Bruit minimal pour préserver l'authenticité. Multiplie la taille de l'ensemble de données.

Prétraitement Audio

- Normalisation : Convertir en mono, compléter/couper à 1 seconde (16000 échantillons).
- Extraction de Caractéristiques : MelSpectrogram (n_mels=64, sample_rate=16000), converti en décibels.
- Format d'Entrée du Modèle : (batch_size, 1, 64, T) où T ≈ 80–85.

Modèle de Deep Learning & Entraînement



Architecture Choisie

CNN simple et efficace : 2 blocs Conv2D + ReLU + MaxPool, Flatten dynamique, Fully Connected, Dropout. Léger et compatible TensorRT.

Entraînement sur Google Colab

GPU CUDA, PyTorch + torchaudio, jeu de données sur Google Drive. Fonction de perte : CrossEntropy, Optimiseur : Adam, Taille de lot : 16, Époques : 100–500.

Sauvegarde Persistante

L'état du modèle et de l'optimiseur est sauvegardé sur Google Drive à chaque époque, permettant la reprise de l'entraînement.

Optimisation pour les Systèmes Embarqués

Quantification (PTQ)

Réduit la mémoire, la latence et la consommation d'énergie. Quantification Post-Entraînement (PTQ) avec étalonnage sur des lots réels, optimisant les poids et les activations (INT8 possible).

Export PyTorch → ONNX

Exportation contrôlée avec entrée fictive (1, 1, 64, T), Opset compatible TensorRT, fichier ONNX unique. Validé avec onnxruntime.

Conversion ONNX → TensorRT

Utilisation de JetPack et TensorRT (trtexec --onnx=... --saveEngine=... --fp16) pour générer un fichier .engine ultra-optimisé pour un chargement instantané et une inférence en temps réel sur Jetson.

Inférence et Améliorations Futures

Inférence en Temps Réel

Sur Jetson ou PC, utilisant une entrée microphone en temps réel ou des fichiers .wav. Le pipeline inclut la capture audio, la détection de silence, un prétraitement identique et l'inférence TensorRT pour la reconnaissance de commandes.

Points Forts Clés

- Pipeline embarqué complet
- Jeu de données personnalisé
- Faible latence
- Compatible Jetson

Domaines d'Amélioration

- Plus de locuteurs diversifiés
- Intégration du bruit ambiant réel de la voiture
- Détection de mot-clé d'activation
- Fusion avec des données de caméra/LiDAR