Rapport détaillé du projet

UTILISATION DES SVM QUANTIQUES (QSVM) POUR LA CLASSIFICATION DES GENRES MUSICAUX FOLKLORIQUES CAMEROUNAIS...

Du son folklorique aux qubits : préserver la culture par l'informatique quantique

Groupe 11

N°	Nom et prénom	Matricule	Option (Niveau)
01	Bounyamine Ousmanou	24ENSPM445	Génie Logiciel (5)
02	DADI GASTON ARSENE DARYLE	22E0500EP	Génie Logiciel (4)
03	KEMTSOP TETSAGHO JOEL	24ENSPM515	Sécurité et Cryptographie (4)
04	MEFIRA MOHAMADOU	24ENSPM0451	Data Science (4)
05	SOUWA MAHAMAT ABBA	24ENSPM0525	RTE (4)

Code source du projet
https://github.com/bounyamine/projet_qsvm_folklorique

Examinateur : **Dr. NOUNAMO DABOU P.**

Dernière mise à jour : December 6, 2025

Résumé

Ce rapport présente la conception et l'implémentation d'un pipeline complet pour la classification binaire de musique folklorique camerounaise (Gurna vs non-Gurna), en combinant traitement du signal audio, extraction de caractéristiques, SVM classique et QSVM avec encodage angulaire. Le système va du son brut jusqu'à une API REST déployable, et explore l'apport potentiel des noyaux quantiques pour ce type de données.

Contents

Chapter 1

Introduction

La musique folklorique joue un rôle central dans la transmission de la mémoire collective et de l'identité culturelle. Au Cameroun, les répertoires traditionnels comme le Gurna constituent un patrimoine immatériel précieux mais souvent peu documenté et menacé par la standardisation des productions musicales. Dans ce contexte, les outils d'intelligence artificielle offrent des moyens nouveaux pour analyser, classer et valoriser ces musiques.

Parallèlement, l'essor de l'informatique quantique ouvre la voie à de nouveaux modèles d'apprentissage automatique, susceptibles d'exploiter des espaces de représentation de très grande dimension via des noyaux (*kernels*) difficilement simulables classiquement. Les modèles de type *Quantum Support Vector Machine* (QSVM) en sont un exemple emblématique.

Ce projet vise à combiner ces deux dimensions – préservation culturelle et IA quantique – en développant un pipeline complet de classification audio folklorique camerounaise (Gurna vs non-Gurna) reposant à la fois sur un SVM classique et un QSVM basé sur un encodage angulaire des caractéristiques audio.

Les contributions principales de ce travail sont les suivantes :

- la conception d'un pipeline de bout en bout pour la classification binaire de fichiers audio folkloriques, depuis les signaux bruts jusqu'aux prédictions de classes ;
- l'implémentation d'un module quantique pour calculer un noyau QSVM à partir d'un encodage angulaire des *features* audio ;
- la mise en place d'une comparaison expérimentale entre un SVM à noyau RBF classique et un QSVM ;
- l'exposition du modèle via une API REST FastAPI et un environnement de déploiement par conteneurs Docker.

Le chapitre 2 présente la vue d'ensemble de l'architecture et la tâche de classification. Le chapitre 3 décrit le prétraitement audio. Le chapitre 4 détaille l'extraction des caractéristiques. Le chapitre 5 est consacré au coeur quantique et au QSVM. Le chapitre 6 décrit le pipeline d'entraînement, de prédiction et d'évaluation. Le chapitre 8 discute l'API et le déploiement.

Enfin, le chapitre 9 propose une discussion critique et des perspectives, avant de conclure en chapitre 10.

Chapter 2

Vue d'ensemble du système

2.1 Tâche de classification et jeu de données

Le problème étudié est une classification binaire de segments audio en deux classes :

- classe 0 : musique folklorique de type Gurna ;
- classe 1 : musique non-Gurna (autres répertoires).

Les signaux audio sont fournis sous forme de fichiers `wav` ou `mp3` organisés par répertoire de classe, par exemple :

- `data/raw_audio/gurna/*.wav`
- `data/raw_audio/non_gurna/*.wav`

Chaque fichier contient un extrait musical, éventuellement bruité, de durée variable. Le pipeline transforme ces signaux bruts en segments normalisés, puis en vecteurs de caractéristiques utilisables par des modèles d'apprentissage supervisé.

2.2 Architecture logicielle générale

Le projet est structuré en plusieurs modules principaux :

- `src/data_pipeline` : prétraitement audio (chargement, suppression des silences, segmentation, normalisation, écriture des segments prétraités) ;
- `src/feature_extraction` : extraction de caractéristiques audio (MFCC, chroma, caractéristiques spectrales, tempo) puis réduction de dimension (StandardScaler + PCA) ;
- `src/quantum_module` : définition du noyau quantique via encodage angulaire et calcul de matrices de Gram ;
- `src/models` : implémentation du QSVM (`QuantumSVM`) comme wrapper compatible scikit-learn autour de `SVC(kernel="precomputed")` ;

- `src/pipeline` : orchestration globale du pipeline via la classe `AudioQSVMpipeline` (train / predict / evaluate) ;
- `src/evaluation` : calcul des métriques (accuracy, précision, rappel, F1, ROC-AUC) et génération de visualisations (matrices de confusion, courbes ROC) ;
- `api/` : API FastAPI exposant des endpoints de prédiction, information modèle et réentraînement ;
- `config/*.yaml` : fichiers de configuration (chemins, paramètres audio, paramètres quantiques).

La commande d'entrée principale est le script `main.py`, qui permet de lancer le pipeline en trois modes :

- `mode = train` : prétraitement des données, extraction des features, entraînement des modèles SVM et QSVM ;
- `mode = evaluate` : évaluation des modèles entraînés ;
- `mode = predict` : prédiction sur un fichier audio donné.

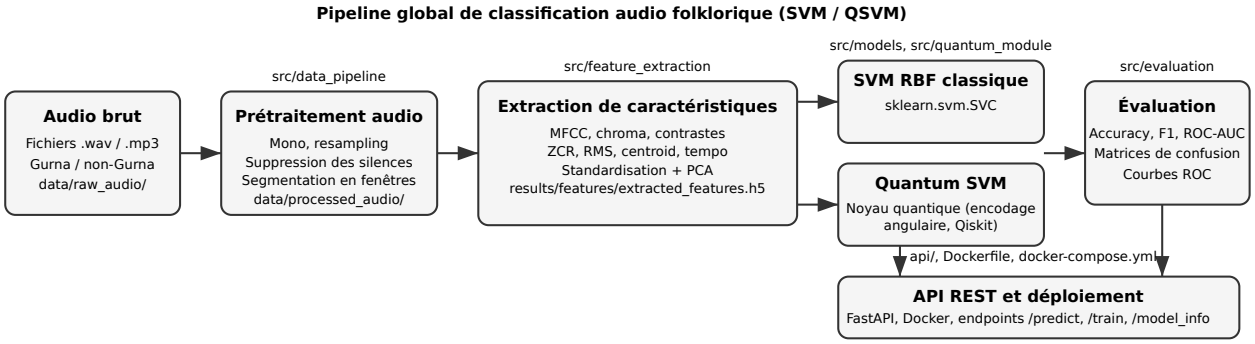

Figure 2.1: Schéma global du pipeline de classification audio folklorique (du son brut à l'API).

Chapter 3

Prétraitement audio

Le prétraitement audio est implémenté dans le module `src/data_pipeline/core.py` via la classe `AudioPreprocessor` et la structure de paramètres `PreprocessParams`.

3.1 Paramètres de prétraitement

Les principaux paramètres de prétraitement, chargés via `config/audio_params.yaml`, sont :

- la fréquence d'échantillonnage cible `sample_rate` (par défaut 22 050 Hz) ;
- la durée de segment `segment_duration_seconds` (par exemple 8 secondes) ;
- le seuil de silence `silence_top_db` pour la suppression des silences ;
- le niveau RMS cible `target_rms` pour la normalisation du volume.

Ces paramètres sont encapsulés dans la dataclasse `PreprocessParams`, ce qui rend le prétraitement modulable et configurable.

3.2 Suppression des silences et segmentation

Pour chaque fichier audio d'entrée, le pipeline effectue les étapes suivantes :

1. **Chargement et mise au format** : le signal est chargé en mono à la fréquence `sample_rate` choisie, ce qui garantit une représentation homogène pour tous les fichiers.
2. **Normalisation RMS** : le niveau d'énergie du signal (RMS) est normalisé vers une valeur cible `target_rms`, ce qui réduit l'impact des variations de volume entre enregistrements.
3. **Suppression des silences** : à l'aide de `librosa.effects.split`, le signal est découpé en intervalles non silencieux en fonction du seuil `silence_top_db`. Les segments silencieux sont éliminés.

4. **Segmentation en fenêtres de durée fixe** : les zones non silencieuses sont concaténées, puis découpées en segments de longueur fixe (par exemple 8 secondes). Les segments trop courts (moins de la moitié de la durée attendue) sont ignorés afin de conserver des entrées informatives.

Chaque segment est ensuite sauvegardé dans `data/processed_audio/<label>/` sous forme de fichier `wav`, ce qui permet de séparer clairement la phase de prétraitement de la suite du pipeline.

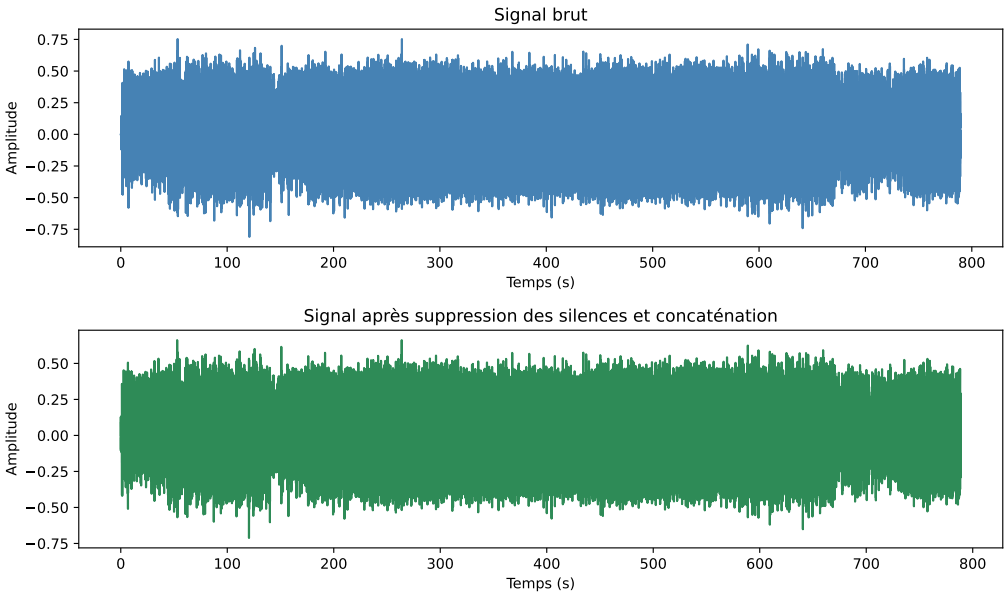

Figure 3.1: Illustration du prétraitement : signal brut, extraction des zones non silencieuses, puis segmentation en fenêtres normalisées.

Chapter 4

Extraction de caractéristiques audio

L'extraction des caractéristiques est assurée par la classe `FeatureExtractor` dans `src/feature_extraction/core.py`, associée à la dataclasse `FeatureParams`.

4.1 Caractéristiques extraites

Pour chaque segment prétraité, un ensemble riche de caractéristiques audio est extrait à l'aide de la bibliothèque `librosa` :

- **MFCC** (Mel-Frequency Cepstral Coefficients), de dimension `n_mfcc` (par défaut 20) ;
- **Chroma** (répartition de l'énergie par classe de hauteur) ;
- **Spectral contrast** (contraste entre pics et vallées du spectre) ;
- **Tonnetz** (représentation tonale) ;
- **ZCR** (Zero Crossing Rate) ;
- **RMS** (Root Mean Square) ;
- **Spectral centroid**, **bandwidth**, **rolloff** (caractéristiques de forme du spectre) ;
- **Tempo** (estimation globale du tempo).

Chaque caractéristique est initialement représentée sous forme de matrice temps–fréquence (par exemple dimension $\times$ nombre de trames).

4.2 Agrégation temporelle et vecteur de features

Les matrices de caractéristiques sont ensuite agrégées en un vecteur de dimension fixe de la manière suivante :

- pour chaque caractéristique matricielle (MFCC, chroma, etc.), on calcule la moyenne et l'écart-type sur l'axe temporel ;

- pour le tempo (scalaire), on conserve directement sa valeur ;
- tous ces vecteurs sont concaténés pour former un vecteur final en une dimension (par segment).

Cette étape produit une représentation compacte mais informative des propriétés spectrales, tonales et rythmiques des segments Gurna et non-Gurna.

4.3 Standardisation et réduction de dimension (PCA)

L'ensemble des vecteurs de caractéristiques est ensuite normalisé et projeté dans un espace de dimension plus faible, adapté aux modèles SVM et QSVM :

1. **Standardisation** : les features sont centrées-réduites (soustraction de la moyenne et division par l'écart-type). Les paramètres du scaler (moyenne, écart-type) sont sauvegardés.
2. **PCA (Principal Component Analysis)** : une PCA est appliquée pour réduire la dimension à un nombre de composantes spécifié dans `config/quantum_params.yaml` (par exemple 8). Les composantes principales et la moyenne PCA sont également sauvegardées.

Les données transformées, accompagnées des labels et des identifiants de fichiers, sont stockées dans un fichier HDF5, typiquement `results/features/extracted_features.h5`. Ce fichier contient également les statistiques de scaling et de PCA nécessaires pour reproduire la même transformation en prédiction.

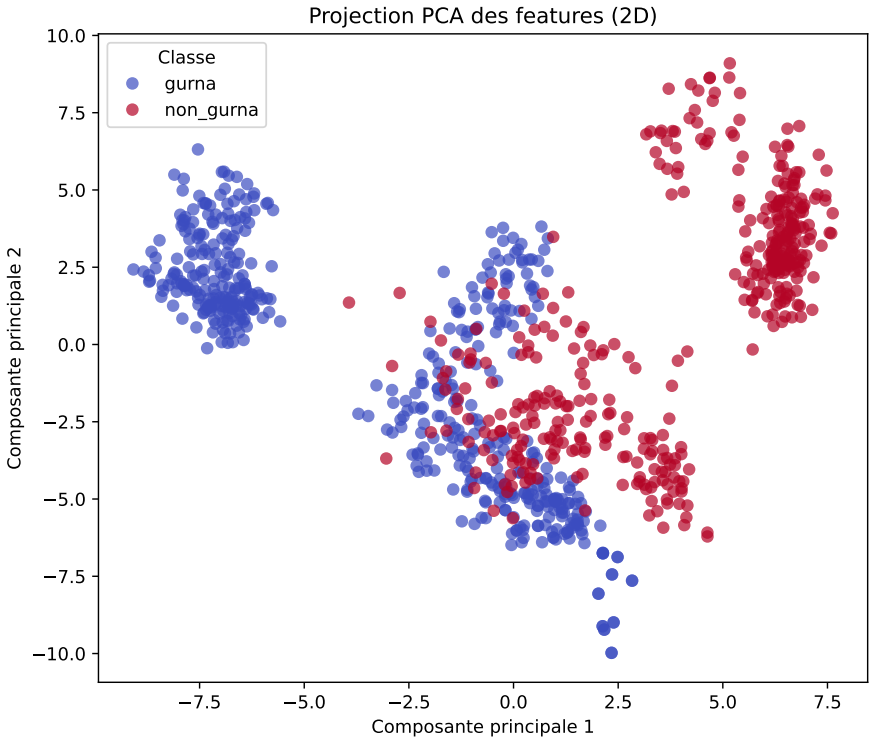

Figure 4.1: Visualisation des features projetés par PCA en 2D, illustrant la séparation entre segments Gurna et non-Gurna.

Chapter 5

Coeur quantique et Quantum SVM

5.1 Encodage angulaire des features

Le module `src/quantum_module/core.py` implémente un noyau quantique basé sur un encodage angulaire des features dans un circuit quantique.

Pour chaque vecteur de features réduit de dimension d, on considère un nombre de qubits n_{qubits} (paramètre configurable). Si d est inférieur à n_{qubits}, les features sont complétées par des zéros ; si d est supérieur, elles sont tronquées. Les valeurs sont ensuite normalisées dans l'intervalle $[-1, 1]$ et transformées en angles via une fonction de type $2 \cdot \arcsin(\cdot)$.

Sur chaque qubit i, on applique une rotation $RY(\theta_i)$, où θ_i est l'angle obtenu pour la i-ème composante du vecteur. Le circuit quantique correspondant prépare ainsi un état $|\varphi(x)\rangle$ qui encode les caractéristiques du segment audio.

Figure 5.1: Schéma conceptuel du circuit d'encodage angulaire : chaque qubit reçoit une rotation RY proportionnelle à une composante du vecteur de features.

5.2 Définition du noyau quantique

Le noyau quantique entre deux vecteurs de features x_1 et x_2 est défini comme :

$$K_q(x_1, x_2) = |\langle \varphi(x_1) \mid \varphi(x_2) \rangle|^2. \tag{5.1}$$

Cette quantité est estimée expérimentalement via un circuit quantique composé de deux étapes d'encodage successives, suivies d'une mesure, et en observant la probabilité de mesurer l'état $|0 \dots 0\rangle$. Plus les états $|\varphi(x_1)\rangle$ et $|\varphi(x_2)\rangle$ sont proches, plus cette probabilité est élevée.

À partir de cette définition, le module construit des matrices de Gram quantiques :

- K_{train} (entraînement–entraînement) ;
- K_{test} (test–entraînement) lorsque nécessaire.

5.3 Implémentation du QSVM

Le module `src/models/quantum_svm.py` fournit la classe `QuantumSVM`, qui encapsule :

- une configuration `QuantumSVMConfig` (nombre de qubits, nombre de tirs `shots`, nom du backend Qiskit, paramètre de régularisation C) ;
- un classifieur interne `SVC(kernel="precomputed", probability=true)` de scikit-learn ;
- une copie des données d'entraînement dans l'espace projeté.

L'entraînement (`fit`) se déroule en deux étapes :

1. calcul de la matrice de Gram quantique K_{train} sur les données d'entraînement ;
2. apprentissage du SVM avec ce noyau pré-calculé.

Pour la prédiction (`predict`, `predict_proba`), le modèle recalcule la matrice de Gram entre les points de test et les points d'entraînement, puis utilise le SVM interne pour produire les décisions ou probabilités.

Les modèles QSVM sont sauvegardés sur disque via `joblib` (fichier `models/qsvm_model.joblib`), contenant à la fois la configuration, le classifieur entraîné et les features d'entraînement.

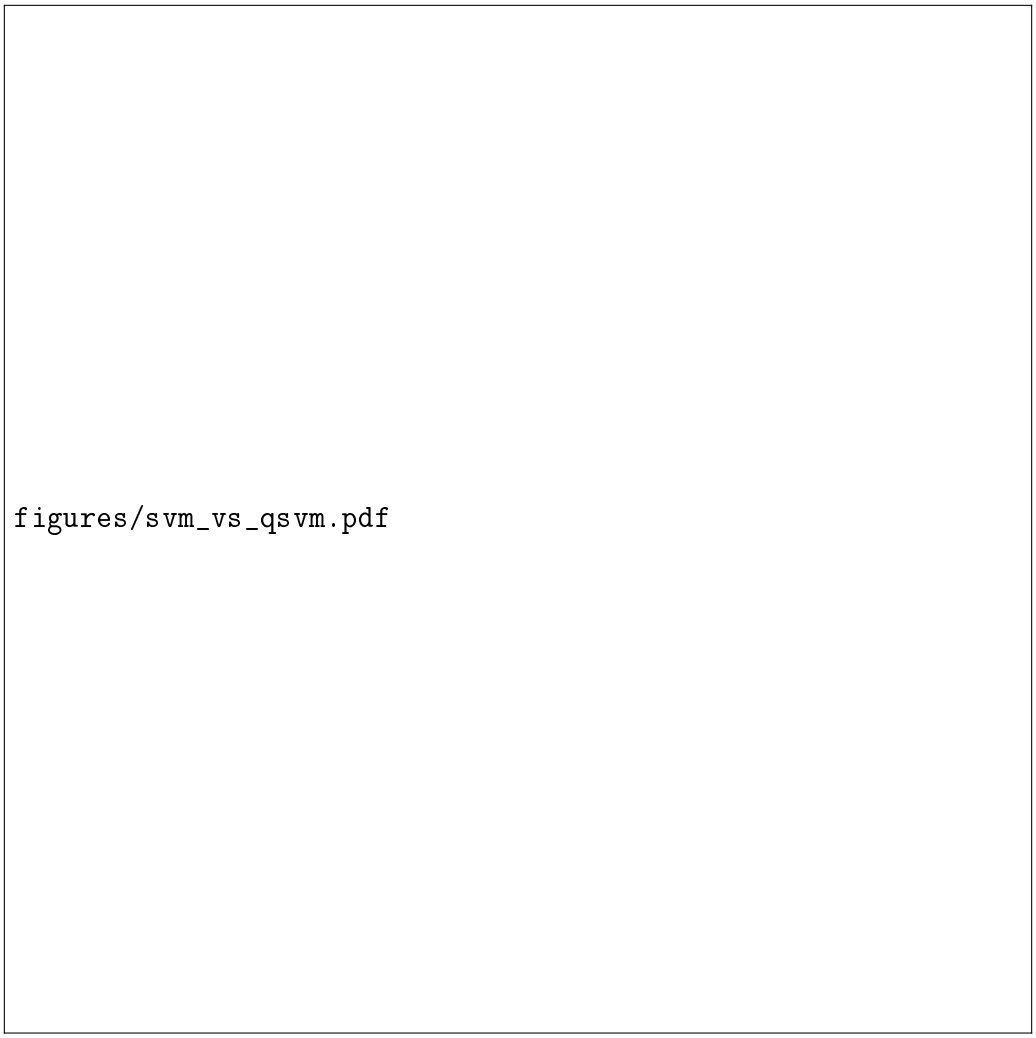

Figure 5.2: Comparaison conceptuelle entre un SVM à noyau RBF classique et un QSVM à noyau quantique basé sur encodage angulaire.

Chapter 6

Pipeline d'entraînement, de prédiction et d'évaluation

L'orchestration de bout en bout est assurée par la classe `AudioQSVMpipeline` dans `src/pipeline/core.py`, configurée via les fichiers YAML du répertoire `config/`.

6.1 Chargement de la configuration

La classe `PipelineConfig` regroupe :

- `paths` (chemins définis dans `config/paths.yaml`) : répertoires de données brutes, données prétraitées, modèles, résultats, etc. ;
- `audio` (paramètres audio, `config/audio_params.yaml`) ;
- `quantum` (paramètres quantiques et PCA, `config/quantum_params.yaml`).

Les chemins importants (données brutes, données prétraitées, fichier HDF5 de features, répertoire des modèles) sont dérivés automatiquement à partir de cette configuration.

6.2 Entraînement

La méthode `train()` suit la logique suivante :

1. **Construction du dataset de features** : si le fichier HDF5 de features n'existe pas, le pipeline lance le prétraitement audio via `AudioPreprocessor.run_full_pipeline()` puis exécute l'extraction de caractéristiques et la PCA via `FeatureExtractor.build_and_save_data` ;
2. **Séparation des données** : les vecteurs de features sont séparés en ensembles entraînement / validation avec `train_test_split` en respectant la stratification des classes ;

3. **Entraînement du SVM RBF classique** : un modèle `SVC(kernel="rbf", probability=true)` est entraîné sur les données réduites. La performance sur l'ensemble de validation est mesurée (accuracy) et le modèle est sauvegardé sous `models/svm_rbf_model.joblib` ;

4. **Entraînement du QSVM** : si l'environnement quantique (Qiskit, backend) est disponible, un sous-échantillon du jeu d'entraînement (par exemple 50 exemples) est sélectionné pour rendre l'entraînement du QSVM praticable, puis une matrice de Gram quantique est calculée sur ce sous-ensemble et un SVM à noyau pré-calculé est entraîné. Le modèle QSVM est sauvegardé dans `models/qsvm_model.joblib`.

En l'absence de données audio (répertoire vide), un dataset synthétique est généré pour maintenir un pipeline fonctionnel, ce qui permet de tester les différentes briques sans jeu de données réel.

6.3 Prédiction

La méthode `predict(audio_path)` prend en entrée le chemin d'un fichier audio et retourne un dictionnaire contenant :

- le label et les probabilités prédites par le SVM RBF ;
- le label et les probabilités prédites par le QSVM (si un modèle QSVM a été entraîné) ;
- le nombre de segments exploités pour la prédiction.

Le flux de prédiction est le suivant :

1. vérification de l'existence du dataset de features et du modèle SVM ;
2. prétraitement du fichier audio (suppression du silence, segmentation, normalisation) via `AudioPreprocessor.process_file()` avec un label spécial (_predict) ;
3. extraction des features pour chaque segment, en réutilisant les mêmes paramètres que pour l'entraînement ;
4. application du scaler et de la PCA à partir des statistiques sauvegardées dans le fichier HDF5 ;
5. prédiction pour chaque segment, puis agrégation via la moyenne des probabilités de classe sur l'ensemble des segments.

6.4 Évaluation

La méthode `evaluate()` évalue les modèles sur un sous-ensemble des données disponibles :

- si un fichier de features audio est présent, les données sont séparées en train / test et :

- le SVM RBF est évalué sur l'ensemble de test ;
- si le modèle QSVM est disponible, il est également évalué sur ce même ensemble ;

- sinon, un dataset synthétique est utilisé pour évaluer au moins le SVM classique.

Les performances obtenues (par exemple `svm_rbf_accuracy`, `qsvm_accuracy`) sont retournées sous forme de dictionnaire et peuvent être exploitées ensuite par le module `src/evaluation` pour produire des figures plus détaillées.

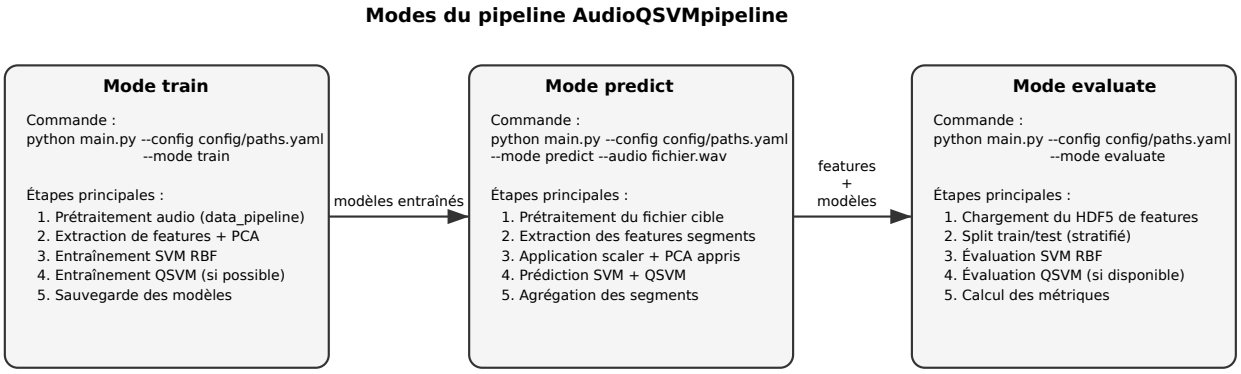

Figure 6.1: Diagramme de flux du pipeline `AudioQSVMpipeline` montrant les différentes étapes pour les modes `train`, `predict` et `evaluate`.

Chapter 7

Évaluation et visualisation des résultats

Le module `src/evaluation/core.py` propose la classe `ModelEvaluator` qui facilite l'analyse détaillée des performances des modèles.

À partir des labels vrais y_{true}, des prédictions y_{pred} et, si disponibles, des probabilités de classe y_{proba}, le module :

- calcule les métriques standard de classification binaire : accuracy, précision, rappel, F1, ROC-AUC ;
- génère et sauvegarde la matrice de confusion (dans `results/evaluations/confusion_matrices/`) ;
- génère et sauvegarde la courbe ROC (dans `results/evaluations/roc_curves/`).

Ces visualisations permettent notamment :

- d'identifier les types d'erreurs les plus fréquentes (par exemple des Gurna confondus avec des non-Gurna) ;
- de comparer la capacité discriminante des modèles en termes de compromis entre taux de vrais positifs et taux de faux positifs.

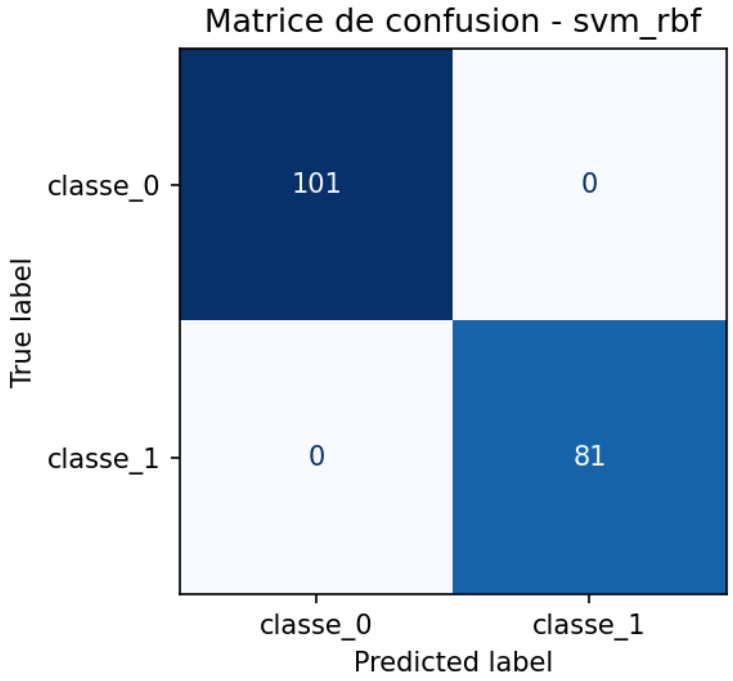

Figure 7.1: Matrice de confusion pour le SVM RBF (exemple).

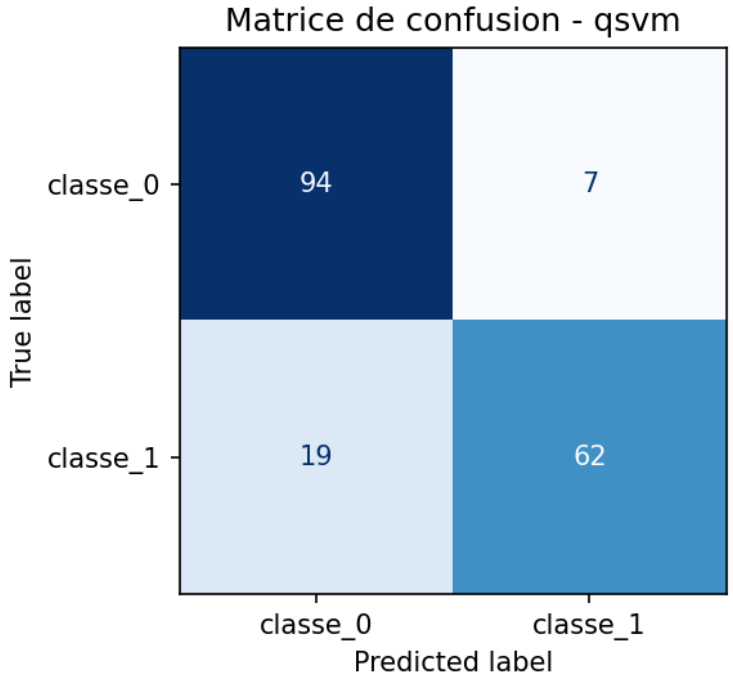

Figure 7.2: Matrice de confusion pour le QSVM (exemple).

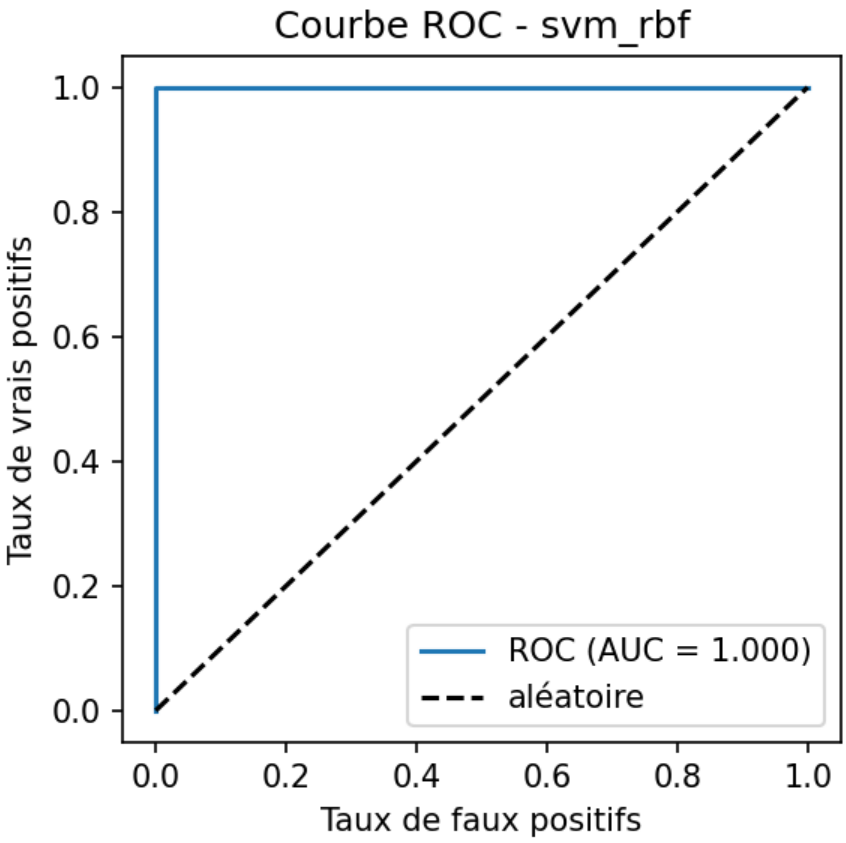

Figure 7.3: Courbes ROC superposées du SVM RBF et du QSVM (exemple).

Chapter 8

API REST et déploiement

8.1 API FastAPI

Le module `api/app.py` définit une application FastAPI exposant plusieurs endpoints principaux :

- `POST /predict` : reçoit un fichier audio en entrée et retourne les prédictions du SVM et, si disponible, du QSVM (labels et probabilités) ;
- `GET /model_info` : fournit des informations sur l'état des modèles (entraînés ou non) et quelques métriques de performance ;
- `POST /train` : permet de relancer un entraînement des modèles à partir des données disponibles (potentiellement protégé par une clé API).

Un endpoint `GET /health` est également disponible pour vérifier simplement que l'API est en fonctionnement.

L'API est documentée automatiquement via l'interface Swagger accessible sur `/docs` lorsqu'on lance le serveur FastAPI localement.

8.2 Conteneurisation et déploiement

Le projet inclut un `Dockerfile` et un fichier `docker-compose.yml` permettant de déployer facilement l'API et les modèles dans un environnement conteneurisé. Les volumes montés assurent la persistance des répertoires `data/`, `models/` et `results/` sur la machine hôte.

Cette approche favorise :

- la reproductibilité des expériences ;
- la portabilité du service de classification vers différents environnements (local, serveur distant, cloud).

Chapter 9

Discussion et perspectives

Ce projet illustre la faisabilité d'un pipeline complet de classification de musique folklorique basé sur un QSVM. L'usage combiné de techniques classiques de traitement du signal audio, de modèles d'apprentissage supervisé (SVM RBF) et de noyaux quantiques offre un terrain d'expérimentation intéressant pour évaluer l'apport potentiel de l'informatique quantique dans un cas d'usage réel.

Plusieurs points méritent toutefois une discussion critique :

- **Qualité et taille du dataset** : la performance des modèles dépend fortement du nombre et de la diversité des enregistrements Gurna et non-Gurna. Un dataset trop restreint peut limiter la généralisation et favoriser le sur-apprentissage, en particulier pour le QSVM, qui est plus coûteux à entraîner.
- **Coût de calcul du QSVM** : le calcul de la matrice de Gram quantique, même en simulation, est coûteux en temps de calcul. Dans ce projet, un sous-échantillon des données d'entraînement est utilisé pour rendre l'entraînement du QSVM praticable, ce qui limite potentiellement ses performances maximales.
- **Choix du schéma d'encodage** : l'encodage angulaire utilisé ici est relativement simple. D'autres schémas d'encodage, plus complexes ou adaptés à des propriétés musicales spécifiques, pourraient capturer des structures plus riches dans les données.

Parmi les perspectives envisageables, on peut citer :

- l'augmentation du dataset avec d'autres répertoires folkloriques camerounais ou d'autres régions du monde ;
- l'exploration de feature maps quantiques plus expressives, éventuellement composées de plusieurs couches de portes et d'entanglement ;
- le test du pipeline sur des machines quantiques réelles, au-delà de la simulation, afin d'étudier l'impact du bruit quantique sur les performances ;
- l'intégration de ce système dans des applications de recommandation, d'indexation ou de valorisation de patrimoine musical.

Chapter 10

Conclusion

Ce travail présente un pipeline complet pour la classification binaire de musique folklorique camerounaise, combinant un SVM classique et un QSVM basé sur un noyau quantique issu d'un encodage angulaire des caractéristiques audio. À partir de signaux bruts organisés par classe, le système met en oeuvre un prétraitement sophistiqué (suppression des silences, segmentation, normalisation), une extraction de caractéristiques riches (spectrales, tonales, rythmiques) et une réduction de dimension adaptée aux contraintes quantiques.

L'implémentation du noyau quantique et du QSVM montre comment les circuits quantiques peuvent être intégrés dans un pipeline d'apprentissage automatique existant, avec une interface compatible scikit-learn. La comparaison expérimentale avec un SVM RBF fournit un premier point de repère pour évaluer l'intérêt et les limitations de cette approche quantique sur des données musicales réelles.

Enfin, l'exposition de ce pipeline via une API FastAPI et la conteneurisation Docker rendent le système facilement déployable et réutilisable dans d'autres contextes. Au-delà de ce cas d'étude, ce projet constitue une base solide pour explorer plus largement l'apport des modèles quantiques dans l'analyse de données culturelles et multimédia.

Bibliography

[1] Brian McFee et al., "librosa: Audio and music signal analysis in python", *Proceedings of the 14th python in science conference*, 2015.

[2] Häner, T., Steiger, D. S., Svore, K. M., *Qiskit: An Open-source Framework for Quantum Computing*, 2017.

[3] Cortes, C., Vapnik, V., "Support-vector networks", *Machine learning*, 20(3), 273–297, 1995.