UNIVERSITÉ
PARIS DESCARTES
UFR DE MATHÉMATIQUES ET INFORMATIQUE

Manuel d'utilisation

Podcast Generator

Sommaire

1.Introduction

L’idée générale du manuel d’utilisation est qu’il doit permettre de garantir une expérience efficace à l’utilisateur au niveau de notre application de création de podcast, **Podcast Generator**.

En effet, cet ouvrage permet de décrire de manière exhaustive les fonctionnalités offerte par cette application et comment les utiliser, chaque option possède une description détaillée, tout cela en donnant des instructions claires et précises sur celle-ci, ceci permet de mettre en lumière au client les différentes fonctionnalités de cette application et comment en tirer parti.

Le manuel d’utilisation constitue une ressource concrète pour guider l’utilisateur au niveau de l’utilisation général de cette application, afin de tirer pleinement partie du potentiel de cette application, pour ainsi permettre de garantir une sécurité́ et une satisfaction globale lors de son utilisation.

1.1. Objectifs et méthodes

L’application **Podcast Generator** est une application de bureau développée avec Python, destinée à être compatible avec les systèmes Windows, Linux et macOS. L'application est certifié de fonctionner sur Windows, possiblement sur MacOS et linux.

1.2. Documents de référence

De nombreux livrables ont été rendus tout au long du projet. Ces derniers ont tous été publiés sur *La Forge* et sont accessibles à tout moment. Certains participent grâce à leurs contenus, à l'élaboration de ce document.

Les documents sont les suivants :

- Spécifications fonctionnelles de l'application Podcast Generator Version 1.0
- Diagrammes UML du projet (cas d'utilisation, classes, séquences)
- Cahier des charges initial du projet
- Cahier de recette initial du projet
- Maquettes UI/UX de l'interface utilisateur
- Planning de développement (Gantt)
- Manuel d'Installation

3.Concepts de base

Génération de texte par IA :

La génération de texte par intelligence artificielle repose sur des modèles de traitement du langage naturel (NLP) entraînés sur de vastes corpus de données. Les grands modèles de langage (LLM) comme GPT analysent et produisent du texte de manière cohérente et contextuelle en s'appuyant sur des algorithmes d'apprentissage profond. Ces modèles permettent de résumer, structurer et formuler du contenu pour l'adapter à différents formats, notamment pédagogiques.

Génération audio et synthèse vocale :

La synthèse vocale (Text-to-Speech, TTS) transforme le texte en parole en utilisant des modèles neuronaux avancés. Certaines de ces technologies modernes génèrent des voix naturelles avec des variations de ton, de rythme et d'émotion, rendant le discours plus fluide et engageant. Des options de personnalisation (choix de voix, vitesse, intonation) permettent d'adapter l'audio aux besoins pédagogiques et aux préférences des auditeurs.

Génération de podcasts :

Un podcast pédagogique suit une structure logique comprenant une introduction, des sections thématiques et des transitions. L'automatisation de la création de podcasts implique la scénarisation du contenu, la génération de dialogues (par exemple, en simulant une discussion entre plusieurs voix), ainsi que l'ajout d'éléments sonores pour améliorer l'expérience d'écoute.

Podcast Generator :
Un logiciel destiné à automatiser la création de podcasts audio à partir de dialogues textuels générés par intelligence artificielle. Le Podcast Generator utilise des modèles de langage pré-entraînés pour produire du contenu cohérent et pertinent.

Modèles de langage (LLM - Large Language Models) :
Des modèles d'intelligence artificielle entraînés sur de grandes quantités de textes permettant de générer du langage naturel réaliste. Ce projet utilise principalement les formats de modèles "GGUF".

Format GGUF :
Un format optimisé pour stocker des modèles de langage utilisés par `llama-cpp-python`. Il permet une performance optimale pour l'inférence locale (utilisation sur votre machine sans nécessiter de connexion internet).

llama-cpp-python :
Une bibliothèque Python servant à exécuter localement des modèles de langage au format GGUF. Elle nécessite une compilation spécifique en fonction de votre système d'exploitation pour garantir des performances optimales.

Environnement virtuel Python (venv) :
Un mécanisme permettant d'isoler les dépendances Python du projet afin d'éviter les conflits avec les autres applications Python installées sur votre système.

Launcher (script de lancement) :
Des fichiers exécutables (scripts batch pour Windows, shell pour macOS ou Linux) facilitant le démarrage rapide de l'application en configurant automatiquement l'environnement nécessaire.

4.Mise en œuvre

Utilisation de l'application

LANCEMENT DE L'APPLICATION

Etape 1 : Récupération des modèles

Le projet Podcast Generator nécessite l'utilisation de modèles de langage au format GGUF. Les modèles recommandés sont les suivants :

- Mistral-Nemo-Instruct-2407-Q8_0 (pour le français et l'anglais)
  Lien :
  https://huggingface.co/Niansuh/Mistral-Nemo-Instruct-2407-Q8_0-GGUF/tree/main

- Qwen1.5-7B-Chat-Q8_0 (pour le japonais et le chinois)
  Lien : https://huggingface.co/Qwen/Qwen1.5-7B-Chat-GGUF/tree/main

Une fois les modèles téléchargés, il faut les placer dans le dossier :
Podcast_Generator_1.0/Models/

Plusieurs modèles peuvent être placés dans ce dossier. L'utilisateur devra ensuite fournir le chemin exact du modèle qu'il souhaite utiliser lors du lancement du serveur.

Téléchargement d'autres modèles :

Il est recommandé d'utiliser le site Hugging Face pour récupérer les modèles au format GGUF :

https://huggingface.co/docs/hub/gguf

Etape 2 : Lancement du programme

Le programme se lance via des fichiers batch (Windows) ou command (macOS). Le comportement diffère légèrement selon le système d'exploitation.

Sous Windows :

L'utilisateur doit lancer le fichier suivant :

- launcher_windows.bat

Ce script effectue automatiquement toutes les étapes suivantes :

Pour le premier terminal:

- L'utilisateur doit fournir le chemin du modèle .gguf à charger.
- Le serveur va se lancer.
- Il faudra attendre que le server soit prêt pour utiliser les options du second terminal, qui est l'application en elle-même.

Pour le second terminal:

- Vérification de la présence de Python3.11.
- Vérification de la présence de llama-cpp-python dans le dossier.
- Il va y avoir la création d'un environnement virtuel Python
- Installation des dépendances nécessaires (requirements.txt)(Nécessite une connexion internet et peut prendre du temps.)[~15 minutes pour le premier lancement.]
- Ouverture du menu principal.

Après le premier lancement, l'exécution du second terminal sera plus rapide car les dépendances seront déjà installées.

Sous macOS :

Le processus nécessite deux étapes distinctes.

1. L'utilisateur doit exécuter une seule fois le fichier suivant pour autoriser les permissions :
   1. install_mac.command

   Pour cela, ouvrir un terminal dans le dossier du projet et exécuter :
   chmod +x install_mac.command
   ./install_mac.command

2. Ensuite, à chaque lancement de l'application :

   a. Ouvrir un terminal et exécuter :
   ./launcher_server_mac.command

   - L'utilisateur doit fournir le chemin du modèle .gguf à charger.
   - Le serveur va se lancer.
   - Il faudra attendre que le server soit prêt pour utiliser les options du second terminal, qui est l'application en elle-même.

   b. Ouvrir un deuxième terminal et exécuter :
   ./launcher_mac.command

   - Vérification de la présence de Python3.11.
   - Vérification de la présence de llama-cpp-python dans le dossier.
   - Il va y avoir la création d'un environnement virtuel Python
   - Installation des dépendances nécessaires (requirements.txt)(Nécessite une connexion internet et peut prendre du temps.)[~15 minutes pour le premier lancement.]
   - Ouverture du menu principal.

Sinon utiliser launcher_mac.command pour lancer les 2 terminaux en même temps.

Remarque importante :

L’interface utilisateur nécessite que le serveur soit lancé et opérationnel. L’utilisateur doit donc impérativement attendre que le serveur ait fini de charger le modèle avant d’utiliser les menus. Dans le doute, attendre que le terminal affiche que le serveur est prêt à recevoir des requêtes avant d’ouvrir le second terminal.

LANCEMENT DE L’APPLICATION en ligne de commande

Si l’environnement est déjà activé vous pouvez lancer le projet avec la commande:

‘python -m llama_cpp.server --model "%MODEL_PATH%" --n_ctx 4096 --n_gpu_layers 100 --port 11434’

en remplacant %MODEL_PATH% par le chemin du model.

Puis sur un autre terminal:

‘python3.11 Podcast_Generator/mainTerminalUI.py’

5. Liste des commandes

L'interface de Podcast Generator se présente sous la forme d'un menu interactif en ligne de commande. Deux modes sont proposés à l'utilisateur : le **mode automatique (pipeline complet)** et le **mode manuel**, chacun disposant de plusieurs sous-menus spécialisés.

1. Mode Automatique (pipeline complet)

Ce mode exécute toutes les étapes de génération d'un podcast à partir d'un fichier source, sans intervention intermédiaire. Il est recommandé pour les utilisateurs souhaitant obtenir un résultat complet rapidement.

Préparez le chemin du fichier à exploiter et donnez-le.

Étapes exécutées automatiquement :

- **1. Extraction du texte source** :
  L'utilisateur fournit un chemin vers un fichier supporté (`.pdf`, `.docx`, `.txt`, `.md`, `.tex`, ou image avec OCR).
  Le texte est extrait via le module `SourceImporter`.

- **2. Résumé automatique avec découpage sémantique (RAG)** :
  Le texte est découpé en blocs, résumés individuellement, puis un résumé global est généré. Cela permet une meilleure structure et compréhension du contenu.

- **3. Extraction des mots-clés et des thèmes et création du résumé** :
  À partir du résumé, le système identifie les concepts principaux sous forme de mots-clés (mots isolés) et de thèmes (titres de sections).
  A noter que le résumé global est le premier bloc du fichier, mais les résumés partiels sont accessibles.

- **4. Sauvegarde des fichiers intermédiaires** :
  Tous les résultats sont stockés dans un dossier `Result/`, incluant `summary.json`, `keywords.json` et `themes.json`.

- **5. Génération du script narratif** :
  Le système génère un script structuré avec une intro, 4 parties thématiques, et une conclusion (OUTRO). Le style narratif s'adapte à la langue choisie.

- **6. Génération automatique des dialogues** :
  À partir du script, le système transforme chaque partie en dialogue réaliste entre plusieurs personnages, avec attribution automatique de noms et de tons vocaux.

- **7. Synthèse vocale complète** :
  Chaque réplique est transformée en audio, puis toutes les parties sont fusionnées. Le podcast final est généré au format `.wav` dans un dossier dédié.

---

2. Mode Manuel

Ce mode permet de contrôler individuellement chaque étape de la génération du podcast. Il offre un accès granulaire à toutes les fonctions internes du système.

2.1. Tone Manager

Permet de gérer les tons vocaux utilisés par la synthèse vocale XTTS (intonation, vitesse, émotion). Ce sous-menu propose :

- Lister tous les tons : affiche tous les tons disponibles (ex: “calme”, “sérieux”, “enthousiaste”), définis dans `tone_presets.json`.
- Ajouter ou modifier un ton : permet de définir un nouveau ton ou de modifier un ton existant en précisant :
    - nom du ton
    - vitesse (float entre 0.5 et 2.0)
    - émotion (neutral, happy, sad, angry)
- Supprimer un ton : supprime un ton du fichier `tone_presets.json`.
- Préécouter un ton : génère un exemple audio d’un ton donné sur un texte par défaut.
- Trouver le ton le plus proche : utilise les embeddings sémantiques pour trouver le ton existant le plus proche d’un mot-clé donné (ex: "tragique" renvoie "dramatique").

2.2. Envoyer une requête au modèle

Cette option permet de saisir manuellement un prompt libre à envoyer au LLM configuré. Cela peut être utile pour :

- tester la compréhension d'un modèle
- explorer des générations personnalisées
- vérifier les capacités de formulation

Le prompt est envoyé en backend **server**.

2.3. Convertir un fichier audio (wav/mp3)

Utilitaires audio pour conversion de formats. Le système propose :

- Convertir un `.wav` en `.mp3` : réduit la taille des fichiers pour l'export ou la diffusion.
- Convertir un `.mp3` en `.wav` : nécessaire si l'on veut utiliser un fichier externe pour XTTS (qui requiert du `.wav`).

Les fichiers sont enregistrés dans le même dossier que l'original avec l'extension modifiée.

2.4. Gestion des voix

Ce menu liste les voix disponibles pour la synthèse. Les catégories suivantes sont disponibles :

- Voix intégrées (XTTS) : voix natives du modèle XTTS (ex: "fr_1", "en_1"...)
- Voix personnalisées masculines : voix situées dans `VoicesPreview/Male/`
- Voix personnalisées féminines : voix situées dans `VoicesPreview/Female/`
- Voix personnalisées toutes : combinaison des deux dossiers précédents
- Voix brutes (racine `Voices/`) : fichiers `.wav` placés directement dans le dossier `Voices/` (hors Male/Female) et qui permettent d'être utilisé comme voix dans le podcast.
  Un fichier est fourni pour prononcer avec la voix que vous voulez une phrase qui couvre une bonne partie des spectres vocaux nécessaires.
- Toutes les voix disponibles : regroupe toutes les sources de voix utilisables

Chaque voix listée peut ensuite être affectée à un personnage dans le menu `create_discussion`.

5. Génération par étapes (menu avancé)

Menu spécialisé permettant de lancer individuellement chaque étape de création du podcast :

1. Extraire du texte (fichier ou URL)

Description :
Permet d'extraire du texte brut depuis un fichier (.pdf, .txt, .docx, .md, .tex) ou une page web.
Les images sont supportées mais très sensibles au bruit.
Pour certaines pages web il est recommandé de faire control+P pour sauvegarder la page web comme PDF pour une meilleure extraction.

Entrée :

- Chemin vers un fichier local (ex. `monfichier.pdf`)
- ou une URL web (ex. `https://...`)

Sortie :

- Texte affiché à l'écran (non sauvegardé)

Remarques :

- Les images et PDF scannés sont traités avec OCR automatique.
- Si l'extraction échoue, un message `[ERREUR]` s'affiche.

---

2. Générer un résumé à partir d'un fichier

Description :
Résume un texte extrait à partir d'un fichier, en le découpant en blocs et en générant un résumé final synthétique.

Entrée :

- fichier (.pdf, .txt, .docx, .md, .tex) ou une page web.
- Les images son supporté mais très sensible au bruit.

Sortie :

- Fichier `summary.json` sauvegardé dans un dossier `Result/RSM-YYYYMMDD-HHMM/`

Contenu du fichier :

- Index 0 : résumé global
- Index 1+ : résumés partiels par bloc

---

3. Extraire les mots-clés à partir d'un résumé

Description :
Extrait automatiquement les mots-clés les plus pertinents à partir d'un résumé JSON.

Entrée :

- Chemin vers un `summary.json` (généré à l'étape 2 ou 5)

Sortie :

- Fichier `keywords.json` dans le même dossier

Contenu :
Liste de mots ou concepts courts, nettoyés et normalisés.

---

4. Extraire les thèmes à partir d'un résumé

Description :
Identique à l'option 3, mais extrait les **thèmes globaux** plutôt que les mots-clés.

Entrée :

- Chemin vers un `summary.json` (généré à l'étape 2 ou 5)

Sortie :

- Fichier `themes.json` dans le même dossier

Contenu :

- Liste de titres de parties (ex. : "Origine de l'IA", "Défis éthiques", etc.)

---

5. Résumé + Mots-clés + Thèmes (enchaînés)

Description :
Effectue les étapes 2, 3 et 4 à la suite.
Génère et sauvegarde automatiquement :

- `summary.json`
- `keywords.json`
- `themes.json`

Entrée :

- Fichier texte source

Sortie :

- Dossier complet prêt pour l'étape suivante

---

6. Générer un script de podcast

Description :
Construit un script narratif à partir du dossier JSON contenant le résumé, les mots-clés et les thèmes.

Entrée :

- Chemin vers dossier contenant ***IMPÉRATIVEMENT***:
    - `summary.json`
    - `keywords.json`
    - `themes.json`

Sortie :

- `script_fr.json` (ou _en ou autre, en fonction de la langue choisie)

Structure :

- `intro`
- `parts`: liste de 4 parties avec titre et contenu
- `outro`

---

7. Générer un dialogue à partir d'un script

Description :
Transforme un script JSON en un dialogue avec plusieurs participants, chacun parlant avec un ton distinct.

Entrée :

- `script_fr.json` (ou _en ou autre, en fonction de la langue choisie)
- Nom d'auteur (inclus dans les métadonnées)

Sortie :

- Fichier texte `DialogueRaw__<titre>__<date>__<heure>__<lang>.txt` (prêt à être transformé en audio)

---

8. Générer une discussion complète depuis un fichier dialogue

Description :
Prend un fichier dialogue `.txt` balisé (`[# Nom : ton : ID]`) et génère tous les fichiers `.wav` dans le même dossier.

Étapes :

- Demande les voix à associer à chaque personnage
- Génère chaque réplique individuellement
- Ajoute l'introduction audio (titre)

Entrée :

- Chemin vers un fichier dialogue formaté
- Noms et voix (ex: José fr_1 Claire Hanekawa.wav)

Sortie :

- Dossier contenant tous les `.wav` nommés par index

---

9. Générer une phrase simple (create_sentence)

Description :
Synthétise vocalement une phrase courte avec un ton et une voix définis.

Entrée :

- Nom de la voix (XTTS ou fichier .wav)
- Nom du ton (doit exister dans `tone_presets.json`)
- Texte à synthétiser

Sortie :

- Fichier `.wav` nommé `sentence.wav` dans `output/`

---

10. Générer un podcast complet (create_discussion)

Description :
Permet d'associer manuellement les voix aux personnages puis de générer tous les fichiers `.wav` correspondants à un fichier dialogue.

Sous-options :

- a. Associer une voix à un personnage
- b. Remplir plusieurs associations d'un coup
- c. Extraire les noms de personnages d'un fichier
- d. Remplacer un nom dans un dialogue
- e. Lancer `create_discussion`
- f. Retour au menu précédent

---

11. Fusionner les fichiers vocaux en 1 (muxgenerateddiscussion)

Description :
Fusionne tous les fichiers `.wav` d'un dossier en un seul fichier audio.
Utilise `muxgenerateddiscussion()`.
Remarque: l'exécution est quasi instantanée.

Entrée :

- Dossier contenant les fichiers `.wav`

Sortie :

- Fichier final nommé `FINAL - <nom_du_dossier>.wav` dans un sous-dossier `Mix/`

---

12. Générer le podcast vocal (à partir d'un fichier dialogue)

Description :
Pipeline audio automatisé qui prend un fichier dialogue `.txt` et génère tous les fichiers vocaux **puis les assemble**.

Entrée :

- Fichier `.txt` de dialogue

Sortie :

- Dossier avec tous les `.wav` + fichier final `.wav` dans `/Mix`

---

13. Changer de langue

Description :
Permet de changer la langue de travail dans le menu par étapes.
Les langues supportées sont :

- `fr` (français)
- `en` (anglais)
- `ja` (japonais)
- `zh-cn` (chinois simplifié)
- `zh-tw` (chinois traditionnel)

---

14. Retour

Description :
Retourne au menu principal.

6. Messages d’erreur

Voici les principaux messages d’erreur que l’utilisateur peut rencontrer lors de l’utilisation de Podcast Generator, ainsi que leurs significations et actions recommandées.

[ERREUR] Texte introuvable ou vide.
→ Le fichier source n’a pas pu être lu ou était vide.
Action : vérifier le chemin du fichier, son contenu, et son format.

[ERREUR] Échec du résumé : …
→ Un problème est survenu lors de la génération du résumé.
Action : vérifier que le serveur est lancé et que le modèle répond. Relancer l’opération.

[ERREUR] Fichier introuvable : …
→ Le chemin spécifié vers un fichier est incorrect.
Action : vérifier l’orthographe et l’emplacement du fichier.

[ERREUR] Impossible d’extraire le texte.
→ L’extraction de texte depuis le fichier source a échoué (par exemple, image floue ou fichier corrompu).
Action : utiliser un fichier plus lisible ou convertir au format texte/PDF clair.

[ERREUR] Échec de la génération audio.
→ Problème lors de la synthèse vocale d’un dialogue ou d’un fichier.
Action : vérifier que la voix demandée est bien disponible et que le texte n’est pas trop long (>273 caractères sans découpage).

[Exception] …
→ Erreur Python ou système inattendu.

Action : lire le message d'exception pour comprendre l'origine (fichier manquant, module absent, paramètre invalide, etc.)

[ERREUR] Une erreur est survenue : …
→ Message générique pour toute exception non gérée.
Action : relancer l'étape ou redémarrer l'application.

7.Annexes

Annexe A – Structure des dossiers du projet

```
Podcast_Generator_1.0/
├── Models/                → Contient les fichiers .gguf des modèles LLM
├── Voices/                → Contient les voix personnalisées
│   ├── Male/
│   ├── Female/
│   └── [autres voix]
├── Result/                → Dossiers générés automatiquement
│   ├── RSM-YYYYMMDD-HHMM/     → Contient summary.json, keywords.json, themes.json
│   └── [autres dossiers]
├── Output/                → Contient les fichiers audio générés
│   ├── Mix/               → Podcast final fusionné (.wav)
│   └── [fichiers .wav]
├── Podcast_Generator/     → Code source de l'application
│   └── [...modules Python]
├── launcher_windows.bat   → Lance le projet complet sous Windows
├── launcher_mac.command   → Lance le projet complet sous macOS
├── launcher_server_mac.command → Lance uniquement le serveur LLM sous macOS
└── install_mac.command    → Rend les scripts macOS exécutables
```

Annexe B – Extrait de fichier dialogue (format attendu)

{
"titre": "L'impact de l'IA en médecine",
"auteur": "Claire Dubois",
"date": "2025-04-27"
}

[# Paul : sérieux : 1] L'intelligence artificielle transforme déjà notre rapport au diagnostic médical.

[# Lucie : calme : 2] Oui, notamment dans la détection précoce des maladies rares.

[# Paul : neutre : 3] Mais cela pose aussi des questions éthiques profondes.

Annexe C – Commande ligne alternative

1. **Lancer le serveur :**

```
python -m llama_cpp.server --model "Models/mon_modele.gguf" --n_ctx 4096
--n_gpu_layers 100 --port 11434
```

2. **Dans un autre terminal :**

```
python3.11 Podcast_Generator/mainTerminalUI.py
```

8.Glossaire

IA (Intelligence Artificielle)
Ensemble de technologies permettant à des systèmes informatiques de simuler certaines capacités humaines comme l'apprentissage, la compréhension du langage, la planification ou encore la génération de contenu textuel ou vocal.

LLM (Large Language Model)
Modèle de langage de grande taille, entraîné sur des corpus massifs de textes. Utilisé pour générer des textes cohérents, résumer des documents, extraire des mots-clés, ou formuler des dialogues. Exemples utilisés : Mistral, Qwen.

Podcast
Fichier audio structuré, destiné à une écoute différée. Un podcast pédagogique suit généralement un format structuré : introduction, plusieurs parties thématiques, conclusion, le tout potentiellement scénarisé sous forme de dialogue.

Script de podcast
Texte structuré généré automatiquement par l'IA, composé d'une introduction, de quatre parties principales et d'une conclusion. Il constitue la base de génération du dialogue.

Dialogue formaté
Texte où chaque ligne de parole est encadrée d'un tag `[# Nom : ton : ID]`. Ce format est utilisé pour générer l'audio ligne par ligne, en associant chaque réplique à une voix et à un ton.

Synthèse vocale (TTS - Text to Speech)
Procédé consistant à transformer du texte en audio. Le système Podcast Generator utilise le moteur XTTS (coqui.ai) avec personnalisation de la voix et du ton.

Ton (ou tonalité)
Paramètre de synthèse audio permettant de moduler l'expression vocale (émotion, vitesse). Les tons sont définis dans un fichier `tone_presets.json` et utilisés pour enrichir la diversité des voix générées.

Voix personnalisée
Fichier audio `.wav` représentant une voix spécifique utilisée pour synthétiser les dialogues. Ces voix peuvent être créées ou récupérées, puis placées dans les dossiers `VoicesPreview/Male/` ou `VoicesPreview/Female/`.

GGUF
Format optimisé pour l'inférence locale de modèles de langage avec la bibliothèque

`llama-cpp-python`. Il est utilisé pour charger efficacement des modèles comme Mistral ou Qwen sans connexion internet.

llama-cpp-python
Bibliothèque Python permettant d'exécuter localement des modèles au format GGUF. Nécessite une compilation adaptée à l'architecture CPU/GPU de la machine utilisée.

RAG (Retrieval-Augmented Generation)
Méthode de résumé consistant à diviser un document en morceaux, à en extraire les parties les plus pertinentes, puis à générer un résumé synthétique à partir de ces éléments.

Environnement virtuel (venv)
Espace isolé de dépendances Python. Utilisé pour garantir que l'installation du projet Podcast Generator ne perturbe pas d'autres projets Python présents sur la machine.

Prompt
Commande textuelle ou instruction saisie par l'utilisateur et envoyée à un LLM pour obtenir une réponse générée. Utilisé pour tester manuellement le comportement du modèle.

Back-end
Méthode choisie pour interagir avec le modèle de langage. Deux options sont possibles dans l'application :

- `local` : via llama-cpp-python (modèle GGUF exécuté localement)
- `server` : via une API HTTP (OpenAI, LM Studio, etc.)

Résultat intermédiaire
Fichier ou dossier produit à une étape du pipeline (ex. : `summary.json`, `keywords.json`, `themes.json`) et utilisé comme entrée pour les étapes suivantes.

9.Références

Références internes au projet :

- Cahier des charges du projet Podcast Generator
- Cahier de recette du projet Podcast Generator
- Plan de tests
- Manuel d'installation
- Conception technique détaillée
- Diagrammes UML
- Planning de développement
- Spécifications fonctionnelles
- Maquettes UI/UX

Références techniques et bibliographiques :

- Hugging Face – Documentation des modèles GGUF : https://huggingface.co/docs/hub/gguf
- Llama-cpp-python – Moteur d'inférence local GGUF : https://github.com/ggerganov/llama.cpp
- XTTS (Text-to-Speech) – Moteur Coqui : https://github.com/coqui-ai/TTS
- Documentation Python 3.11 : https://docs.python.org/3.11/
- Présentation du format GGUF (llama.cpp) : https://github.com/ggerganov/llama.cpp/blob/master/docs/gguf.md
- Exemple de modèle utilisé : https://huggingface.co/Niansuh/Mistral-Nemo-Instruct-2407-Q8_0-GGUF
  https://huggingface.co/Qwen/Qwen1.5-7B-Chat-GGUF

10. Index