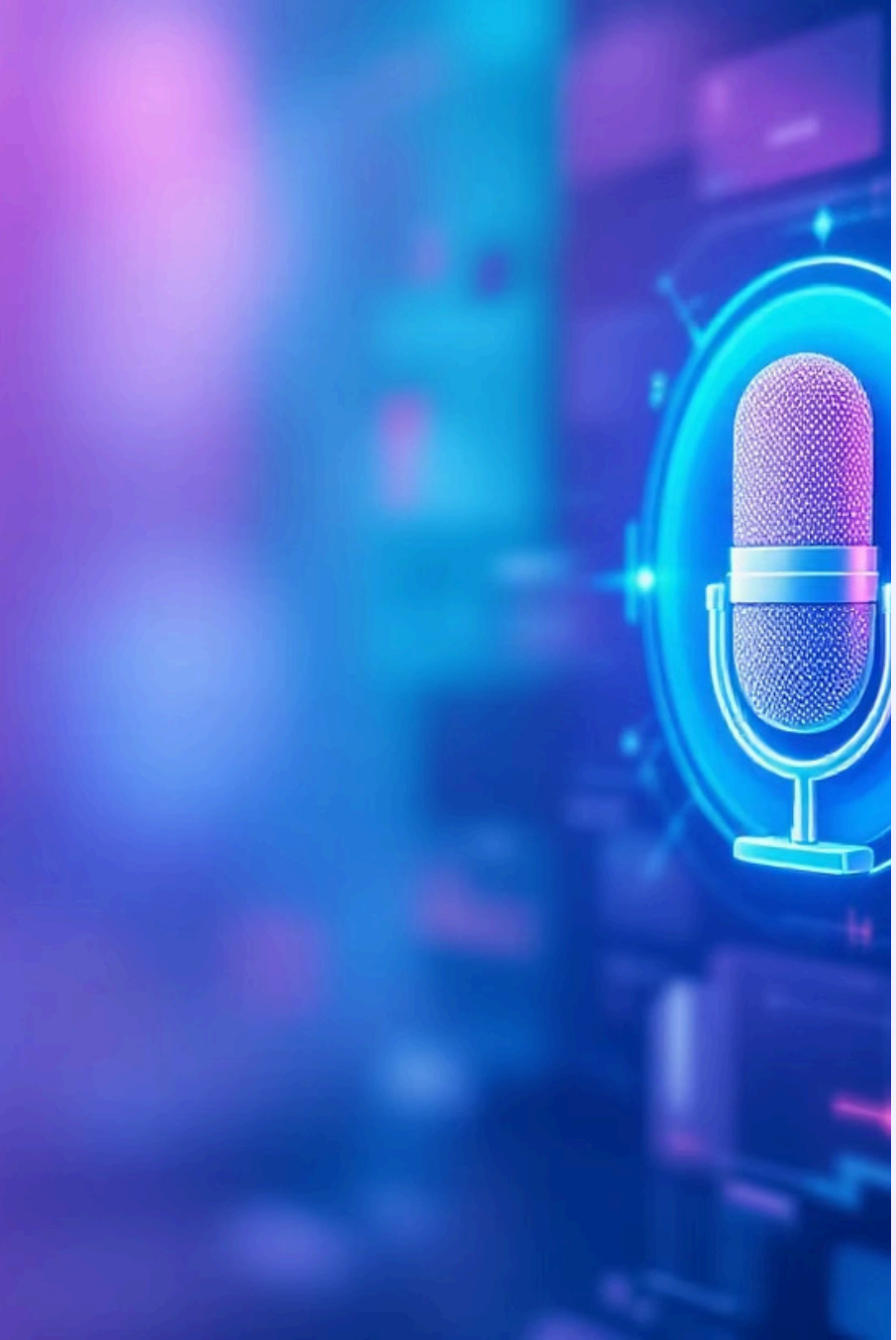

شرح مشروع المساعد الصوتي الشخصي (Hands-Free)

يهدف هذا المشروع إلى بناء مساعد صوتي شخصي متكامل باللغة العربية، قادر على الاستماع الدائم لكلمة تفعيل محددة، ثم معالجة الأوامر الصوتية المعقدة وتنفيذ مهام متنوعة. يعتمد المشروع على مزيج من تقنيات التعرف على الكلام، فهم اللغة الطبيعية (NLU)، وتشغيل الصوت، بالإضافة إلى دمج نماذج الذكاء الاصطناعي التوليدية لتقديم استجابات ذكية.

Ahmed Alzain ^_^

بنية المشروع ومحتوياته الرئيسية

gui.py

المحتوى: يحتوي على الكلاسات الخاصة بالواجهة الرسومية (مثل ChatMessageWidget, TypingIndicatorWidget, SettingsDialog)، بالإضافة إلى الكلاس الرئيسي MainWindow الذي يدير التفاعلات المنطقية مع المستخدم ومكونات الخلفية.

الدور: يعرض المحادثات، يوفر إعدادات للمستخدم، ويستقبل الأوامر من خيوط الصوت ويوجهها للمعالجة.

main.py

المحتوى: يقوم بتهيئة نظام التسجيل (logging) لتوثيق الأحداث والأخطاء، يحمّل متغيرات البيئة من ملف .env (مثل مفاتيح API). يُنشئ كائن التطبيق الرئيسي لـ PySide6 ويُطلق الواجهة الرسومية MainWindow.

الدور: هو المسؤول عن بدء تشغيل كل شيء في المشروع.

plugins/

المحتوى: كل ملف .py داخل هذا المجلد (باستثناء الملفات الداخلية مثل __init__.py) يمثل إضافة بوظيفة محددة.

الدور: يوفر بنية معيارية لتنفيذ مهام محددة مثل التحكم بالصوت، فتح التطبيقات، والبحث على الإنترنت.

audio_manager.py

المحتوى: يتضمن كلاسات QThread (خيوط) منفصلة لكل وظيفة صوتية: WakeWordThread، AudioPlaybackThread، وConversationThread.

الدور: يجعل المساعد يستمع لكلمة التفعيل، ويسجل أوامر المستخدم، ويقوم بتشغيل الردود الصوتية دون تجميد الواجهة.

train_nlu.py

المحتوى: يحتوي على دوال لتحميل البيانات من training_data.json وتدريب نموذج spaCy.

الدور: يقوم بإنشاء أو تحديث نموذج nlu_model_ar الذي يستخدمه المساعد لفهم الأوامر.

training_data.json

المحتوى: يحدد "النوايا" (Intents) التي يمكن للمساعد فهمها (مثل GET_TIME, GET_WEATHER, OPEN_APPLICATION). لكل نية، يوجد مجموعة من "النصوص" التدريبية (العبارات التي قد يقولها المستخدم). كما يحدد "الكيانات" (Entities) التي يجب استخراجها من هذه النصوص (مثل APP_NAME, LOCATION, QUERY, VOLUME_LEVEL, VOLUME_TARGET).

الدور: أساس تدريب نموذج spaCy لفهم الأوامر المعقدة.

user_data.json

المحتوى: يخزن بيانات مثل اسم المستخدم، المدينة الافتراضية للطقس، ومؤشر الميكروفون المفضل.

الدور: يحفظ التفضيلات بين جلسات الاستخدام لتجربة شخصية.

nlu_model_ar/

المحتوى: يتضمن الملفات الناتجة عن عملية التدريب بواسطة train_nlu.py.

الدور: يُحمّل بواسطة gui.py لاستخدامه في تصنيف نوايا المستخدم واستخراج الكيانات من كلامه.

requirements.txt

المحتوى: يسرد جميع مكتبات Python الخارجية التي يحتاجها المشروع (مثل pyaudio, pvporcupine, SpeechRecognition, PySide6, spacy, google-generativeai, gtts, requests, python-dotenv, comtypes, pycaw).

الدور: يسهل تثبيت جميع التبعيات اللازمة لتشغيل المشروع.

.env

المحتوى: مفاتيح API لخدمات مثل Picovoice, Google Gemini, و WeatherAPI.

الدور: يوفر طريقة آمنة لإدارة بيانات الاعتماد دون تضمينها مباشرة في الكود.

app.log

المحتوى: يسجل رسائل المعلومات والأخطاء التي تحدث أثناء تشغيل التطبيق.

الدور: مفيد جدًا لتتبع المشكلات وتشخيص الأخطاء.

الكلاسات والدوال الرئيسية وآلية عملها الموسعة

1 main.py

الدور: يبدأ تشغيل التطبيق.

آلية العمل:

1. load_dotenv(): يحمّل مفاتيح API والمتغيرات الأخرى من ملف .env، مما يتيح للكود الوصول إليها كمتغيرات بيئة.
2. logging.basicConfig(): يُهيئ نظام التسجيل، موجهًا الرسائل إلى ملف app.log وإلى وحدة التحكم (console). هذا ضروري لتتبع سلوك التطبيق وتصحيح الأخطاء.
3. QApplication(sys.argv): يُنشئ كائن تطبيق PySide6، وهو ضروري لتشغيل أي تطبيق يعتمد على Qt.
4. MainWindow(): يُنشئ مثيل (instance) لكلاس MainWindow للمساعد الذي يمثل واجهة المستخدم الرئيسية.
5. window.show(): يعرض نافذة التطبيق للمستخدم.
6. sys.exit(app.exec()): يبدأ حلقة أحداث التطبيق (event loop)، مما يسمح للواجهة الرسومية بالاستجابة لتفاعلات المستخدم وإدارة الخيوط.

2 audio_manager.py

AudioPlaybackThread(QThread)

- **السمات:** _is_playing, _stop_requested, current_audio_file
- **الدوال:** run(), play_audio(file_path), stop_playback()
- **الإشارات:** finished

WakeWordThread(QThread)

- **السمات:** access_key, porcupine, pa, audio_stream
- **الدوال:** run(), stop()
- **الإشارات:** wake_word_detected, stop_speaking_signal, wake_word_error_signal

ConversationThread(QThread)

- **السمات:** _is_in_conversation, microphone_index
- **الدوال:** run(), stop()
- **الإشارات:** conversation_signal, status_signal, conversation_finished_signal, command_to_execute_signal, stop_speaking_signal

3 gui.py

MainWindow(QMainWindow)

السمات الرئيسية: user_name, user_city, microphone_index, commands, nlu_model, audio_playback_thread, wake_word_thread, conversation_thread, gemini_chat_session, follow_up_plugin

الدوال الرئيسية:

- __init__(): تهيئة الواجهة، تحميل الإعدادات والإضافات
- _set_preferred_browser(): تحديد مسار متصفح Google Chrome
- open_url_in_thread(url): فتح روابط في خيط منفصل
- _open_application(app_name): فتح تطبيقات النظام
- open_settings_dialog(): فتح نافذة الإعدادات
- listen_for_single_response(timeout): الاستماع لرد صوتي واحد
- run_first_time_setup(): إعداد التطبيق لأول مرة
- check_for_first_run(): التحقق من وجود ملف الإعدادات
- load_plugins(): تحميل الإضافات من مجلد plugins
- stop_current_speech(): إيقاف الكلام الجاري
- execute_command(command): معالجة الأوامر الصوتية
- start_wake_word_engine(): بدء محرك كلمة التفعيل
- handle_wake_word_error(error_message): معالجة أخطاء كلمة التفعيل
- start_conversation_mode(): بدء وضع المحادثة
- add_message_to_conversation(speaker, text, source): إضافة رسائل للمحادثة
- update_status(text, color_hex): تحديث حالة التطبيق
- closeEvent(event): معالجة إغلاق التطبيق
- speak(text, source): تحويل النص إلى كلام

بنية الإضافات - plugins/

كل إضافة في هذا المجلد تحتوي على كلاس Plugin مع الدوال التالية:

- **get_keywords():** تُرجع قائمة بالكلمات المفتاحية أو العبارات التي ترتبط بوظيفة هذه الإضافة.
- **execute(main_window, command, speak_func, query_text=None):** تنفذ وظيفة الإضافة الفعلية.
- **process_follow_up(main_window, command, speak_func):** دالة اختيارية لمعالجة استجابات المستخدم عندما تحتاج الإضافة إلى معلومات إضافية.

آلية العمل - execute_command(command)

1. **أوامر التحكم:** تتحقق أولاً من الأوامر المباشرة مثل "stop_conversation" لإنهاء وضع المحادثة.
2. **أوامر المتابعة:** إذا كان هناك self.follow_up_plugin نشطًا، يتم تمرير الأمر مباشرة إلى دالة process_follow_up لهذا المكون.
3. **فهم اللغة الطبيعية (NLU):** يتم تمرير الأمر إلى self.nlu_model (نموذج spaCy) لتحديد "النية" والكيانات المرتبطة بها.
4. **معالجة Gemini AI:** إذا لم يتم تصنيف الأمر بواسطة NLU، يتم تمرير الأمر إلى handle_gemini_chat().

train_nlu.py - train_spacy(data, iterations, model_name)

1. إنشاء نموذج spaCy فارغ للغة العربية
2. إضافة مكونات "مصنف النص" و"مستخرج الكيانات المسماة"
3. إضافة النوايا والكيانات المعرفة في بيانات التدريب
4. بدء عملية تدريب النموذج
5. تحديث النموذج لكل مثال تدريبي مع تطبيق التسرب
6. حفظ النموذج في المجلد المحدد (nlu_model_ar)

train_nlu.py - load_data(file_path)

يقرأ ملف training_data.json ويقوم بتنسيق البيانات ليتوافق مع متطلبات spaCy، حيث يحوّل كل عبارة إلى قاموس يحتوي على النص، الكيانات المستخرجة، والنوايا المرتبطة بها.

دورة الاستخدام من البداية للنهاية

الإعداد (Preparation)

- **المطور/المستخدم:** يقوم بتثبيت Python والمتطلبات من requirements.txt. يُنشئ ملف .env ويضع مفاتيح API اللازمة فيه.
- **المطور/المستخدم:** يُشغّل python train_nlu.py لتدريب نموذج فهم اللغة الطبيعية، مما يُنشئ مجلد nlu_model_ar.

بدء التطبيق (Application Startup)

- **المستخدم:** يُشغّل python main.py.
- **النظام:** main.py يقوم بتهيئة نظام التسجيل وتحميل متغيرات البيئة.
- **النظام:** MainWindow تبدأ بالتحميل، وتقوم بتحميل الإضافات والنموذج المدرب.
- **النظام:** إذا كانت هذه هي المرة الأولى، تبدأ عملية الإعداد الأولي التفاعلي.
- **المساعد:** يبدأ WakeWordThread في الاستماع باستمرار لكلمة التفعيل "أليكسا".

تفعيل المساعد (Wake Word Detection)

- **المستخدم:** يقول "أليكسا".
- **النظام:** WakeWordThread تكتشف كلمة التفعيل وتُطلق إشارة wake_word_detected.
- **المساعد:** يقوم MainWindow باستلام الإشارة، ويُطلق ConversationThread.
- **المساعد:** يُشغّل صوت ترحيب ويُحدّث الواجهة إلى "تفضل بأمرك...".

وضع المحادثة وتلقي الأمر

- **المستخدم:** يصدر أمرًا صوتيًا (مثلاً: "افتح لي الرسام").
- **النظام:** ConversationThread تستخدم speech_recognition لتسجيل الصوت وتحويله إلى نص.
- **النظام:** يُعرض الأمر في سجل المحادثة بالواجهة.
- **النظام:** إذا قال المستخدم كلمات مثل "توقف"، تُنهي ConversationThread نفسها.

معالجة الأمر (Command Processing)

- **النظام:** ConversationThread تُرسل الأمر النصي إلى دالة execute_command.
- **النظام:** يتم تحليل الأمر باستخدام نموذج NLU لتحديد النية واستخراج الكيانات.
- **النظام:** بناءً على النية والكيانات، يتم توجيه الأمر إلى الإضافة الصحيحة أو تنفيذه مباشرةً.

الاستجابة الصوتية والمرئية

- **النظام:** بعد معالجة الأمر، تُصدر الإضافة أو دالة المعالجة نص الرد.
- **المساعد:** دالة speak(text, source) تستخدم gTTS لتحويل النص إلى ملف صوتي.
- **النظام:** يُرسل مسار الملف الصوتي إلى AudioPlaybackThread للتشغيل.
- **النظام:** تُضاف رسالة المساعد إلى واجهة المحادثة، وتُحدّث الحالة.

تكرار دورة المحادثة / العودة لوضع كلمة التفعيل

- **النظام:** بعد كل أمر، تُحدث ConversationThread الواجهة إلى "في انتظار أمرك التالي...".
- **النظام:** يستمر ConversationThread في الاستماع لأوامر جديدة.
- **المستخدم:** يمكنه قول أمر آخر أو كلمات مثل "توقف".
- **النظام:** إذا توقفت المحادثة، يعود المساعد تلقائيًا إلى وضع الاستماع لكلمة التفعيل.

أمثلة على الأوامر المدعومة

- "افتح لي الرسام" - فتح تطبيق الرسام
- "ما هو الطقس في دمشق؟" - الاستعلام عن الطقس
- "ابحث عن أغنية هادئة على يوتيوب" - البحث في يوتيوب
- "من هو مخترع الهاتف؟" - استعلام معرفي عام
- "ارفع الصوت" - التحكم بمستوى الصوت
- "كم الساعة الآن؟" - الاستعلام عن الوقت

اختبار وظائف المساعد الصوتي المتعلقة بمهام النظام، والمتصفح، ويوتيوب، و Gemini.

يمكنك اتباع الخطوات التالية بعد التأكد من أنك قمت بتعديل ملف training_data.json وإعادة تدريب النموذج بنجاح:

قبل البدء: تأكد من تشغيل تطبيق المساعد الصوتي الرئيسي (main.py) حتى يكون جاهزًا للاستماع للأوامر.

اختبار مهام النظام (التحكم بالصوت وفتح التطبيقات)

1. التحكم بالصوت:

رفع الصوت: قل "ارفع الصوت" أو "علي الصوت".

يجب أن: يزيد مستوى الصوت، ويقول المساعد "تم رفع الصوت".

خفض الصوت: قل "اخفض الصوت" أو "وطي الصوت".

يجب أن: ينخفض مستوى الصوت، ويقول المساعد "تم خفض الصوت".

كتم الصوت: قل "كتم الصوت".

يجب أن: يتم كتم الصوت، ويقول المساعد "تم كتم الصوت".

إلغاء كتم الصوت: قل "إلغاء كتم الصوت" أو "شغل الصوت".

يجب أن: يعود الصوت، ويقول المساعد "تم تشغيل الصوت".

ضبط مستوى معين: قل "اضبط الصوت على 50" (أو أي رقم بين 0 و 100).

يجب أن: يتم ضبط الصوت على المستوى المحدد، ويقول المساعد "تم ضبط الصوت على 50 بالمئة".

معرفة مستوى الصوت الحالي: قل "صوت" فقط.

يجب أن: يخبرك المساعد بمستوى الصوت الحالي.

2. فتح التطبيقات:

فتح الرسام: قل "افتح الرسام" أو "شغل الرسام".

يجب أن: يفتح تطبيق الرسام.

فتح الآلة الحاسبة: قل "افتح الآلة الحاسبة" أو "شغل الآلة الحاسبة".

يجب أن: يفتح تطبيق الآلة الحاسبة.

فتح المفكرة: قل "افتح المفكرة" أو "شغل المفكرة".

يجب أن: يفتح تطبيق المفكرة.

فتح جوجل كروم: قل "افتح جوجل كروم" أو "شغل جوجل كروم".

يجب أن: يفتح متصفح جوجل كروم.

فتح المنبه: قل "افتح المنبه" أو "شغل المنبه".

يجب أن: يفتح تطبيق المنبهات أو الساعة.

فتح فيجوال ستوديو كود: قل "افتح فيجوال" أو "افتح فيجوال ستوديو كود".

يجب أن: يفتح برنامج Visual Studio Code.

3. اختبار وظائف المتصفح فتح المتصفح الافتراضي:

قل "افتح المتصفح".

يجب أن: يفتح المتصفح الافتراضي على صفحة جوجل الرئيسية.

البحث في المتصفح عن شيء محدد: قل "ابحث بالمتصفح عن آخر أخبار اليوم" أو "ابحث عن أسعار الذهب".

يجب أن: يفتح المتصفح ويجري بحثًا عن العبارة المطلوبة على جوجل.

4. اختبار وظائف يوتيوب فتح يوتيوب:

قل "افتح يوتيوب" أو "شغل يوتيوب".

يجب أن: يفتح موقع يوتيوب في المتصفح.

البحث في يوتيوب عن فيديو معين: قل "ابحث في يوتيوب عن اغنيه هادئه" أو "شغل اغنيه هادئه على يوتيوب".

يجب أن: يفتح يوتيوب ويجري بحثًا عن الفيديو المطلوب.

5. اختبار وظائف Gemini (لأسئلة المعرفة العامة) أسئلة المعرفة العامة:

اطرح أسئلة لا تتطلب فتح تطبيق أو بحثًا مباشرًا في المتصفح أو يوتيوب، مثل:

"ما هي عاصمه الصين؟"

"من هو مخترع الهاتف؟"

"كم عدد الكواكب في النظام الشمسي؟"

"متى تأسست شركة مايكروسوفت؟"

"ما هو لون السماء؟"

"من هو مؤسس الدوله؟"

الاستجابة المتوقعة:

يجب أن: يقوم المساعد الصوتي بمعالجة السؤال باستخدام نموذج Gemini (يجب أن ترى "أفكر..." ثم إجابة من "Gemini" في واجهة المستخدم).