Project: Jarvis Dictator

Goal: Build the ultimate AI-powered dictation system (Jarvis-style) where the user can speak naturally, and the app acts as an intelligent assistant for writing and editing.

Core Concept: When the user speaks, their words are transcribed live. The integrated AI analyzes context, offers grammar corrections, suggests better phrasing, executes editing commands, and even speaks back interactively.

Phase 1: Voice Dictation (Speech → Text)

- Capture live speech using the microphone.
- Convert voice to text (using Whisper, Web Speech API, or similar).
- Display text in real-time in an editor.

Phase 2: Intelligent AI Layer

- AI listens for special voice commands like:
- “Go back and change that.” → undo or edit last sentence.
- “Verify the grammar.” → grammar correction.
- “Make it sound more formal / creative / shorter.” → style rewriting.
- Suggest alternative phrasings for clarity or tone.

Phase 3: Conversational Feedback (Voice + Text)

- Jarvis talks back: reads your text or comments using Text-to-Speech (TTS).
- The AI can summarize, clarify, or confirm actions with natural voice.

Phase 4: Integration

- Stack: React (frontend) + Flask (backend)
- Speech-to-Text: Whisper / Web Speech API
- Text Processing: Local/Cloud LLM (e.g., DeepSeek, Granite, Mistral)
- Text-to-Speech: OpenAI TTS / ElevenLabs / pyttsx3

Example Use Flow

1. User: “Start dictation.” → Jarvis: “Listening.” (mic active)
2. User: “The system should handle multiple inputs simultaneously.” → Text appears live.
3. User: “Make it more concise.” → AI rewrites it to: “The system must support concurrent inputs.”
4. User: “Verify grammar.” → AI highlights and fixes issues.
5. User: “Read it back.” → Jarvis narrates the paragraph aloud.

Future Enhancements

- Contextual awareness (knows document type: report, email, essay)
- Real-time collaborative editing with voice commands
- Adaptive tone (formal, persuasive, technical)
- Voice profiles (choose Jarvis, Ava, etc.)

This file anchors the direction of development for the Jarvis Dictator project — an AI-driven, voice-first writing assistant designed to combine dictation, natural language understanding, and intelligent editing in one seamless experience.

Persona & Discord "Second Friend" Editor

Vision: The AI behaves like a calm, helpful friend sitting in a Discord voice channel, talking with you as an **editor-companion**. It listens as you dictate, offers better wording, fixes grammar when asked, and performs document edits on command. Tone is conversational, precise, and respectful (avoid em dashes per preference; use commas or semicolons instead).

Core Behaviors - **Conversational Editing:** Natural dialogue like “replace the last sentence with…”, “tighten that paragraph”, “make it more formal”, “undo that change”, “summarize this section”, “insert a bullet list of three key points”. - **Live Dictation:** Continuous speech capture, low-latency transcription, immediate insertion to the active document with minimal lag. - **Read-Back & Confirmation:** “Read that back”, “what changed”, “why is this better”, optional confirmations for destructive actions. - **Grammar & Style Coach:** On-demand grammar checks and tone transformations (formal, concise, persuasive, friendly, technical). - **Memory & Context:** Remembers current section, recent changes, your style choices, and project context (e.g., MDPI/IEEE style). Supports “return to where we were”. - **Non-intrusive:** Offers suggestions succinctly; never blocks your flow.

Discord Integration (MVP) - **Join Voice Channel:** Slash command `/join` or keyword “Jarvis join my channel”. - **Capture Audio:** Receive Opus frames, transcode, stream to STT (Whisper local or API). - **Transcription Pipeline:** VAD → chunking → Whisper → real-time text insertions. - **Command Understanding:** Lightweight command parser on top of LLM intent classification. Keywords like *replace, undo, rewrite, verify grammar, summarize, split, merge, bullet, heading, comment, accept, reject*. - **Speak Back:** TTS engine streams audio to Discord voice (ElevenLabs, OpenAI TTS, Edge-TTS). Short, clear responses. “Would you like me to apply that?” only when necessary. - **Text Channel Bridge:** Mirror transcripts and edits to a text channel for history and bookmarking.

Tech Choices - **Bot Framework:** `discord.py` or `discord.js` with voice support. - **Audio I/O:** Discord voice gateway, Opus encode/decode, jitter buffer, low-latency playback. - **STT:** Whisper (local GPU) or API; VAD with Silero/WebRTC for chunking. - **LLM:** DeepSeek, Mistral, or Granite via local inference or API with short-term conversation memory (windowed). RAG hooks for style guides and your past writing. - **TTS:** ElevenLabs or OpenAI TTS for natural voice; fallback to Edge-TTS. - **State:** In-memory session with rolling transcript; operational transform or CRDT if multi-client editing later.

Voice Command Examples - “Replace the last sentence with, *the system must support concurrent inputs*.” - “Make paragraph two concise; keep the numbers.” - “Verify grammar for the last two sentences.” - “Undo the previous change; read the original back.” - “Summarize the current section in three bullets.” - “Change the tone to formal.”

Latent Constraints & Preferences - Avoid em dashes; prefer commas or semicolons. - Respect document style (Times New Roman for exports, MDPI or IEEE rules when relevant). - Keep responses brief in voice; richer detail in the text channel on request.

MVP Task List 1) Create Discord bot, permissions, slash commands `/join`, `/leave`, `/save`. 2) Voice join, receive audio, basic VAD, stream to STT, print transcript to console and text channel. 3) TTS playback for “listening/ack/apply/undo/read-back”. 4) Intent parser: detect *dictate vs command*; support **replace**, **undo**, **verify grammar**, **read back**. 5) In-memory document buffer with change history, anchor by sentences.

Next - Add editor personas, customizable voices. - Add project-aware RAG (style guides, your past docs). - Add collaborative co-editing with conflict-safe ops.

Deployment Vision: Cross-Platform Writing Extension

Initial Phase: Browser Extension - Build as a Chrome/Edge extension first, enabling voice dictation and AI editing directly inside text areas, editors, or chat boxes on any website. - Use Whisper (Web Speech API fallback) for real-time speech-to-text. - Inject a small floating toolbar (🎙️, , ,) that appears near any text box, allowing voice commands, grammar check, rewrite, and read-back. - Maintain session context and undo stack per site.

Supported Targets (Phase 1–2) - Google Docs, Notion, Gmail, Reddit, AO3, and ChatGPT input fields. - Works in textareas, contenteditable elements, and iframes. - Uses background service worker for persistent STT and LLM inference (via backend API).

Phase 3: Native Everywhere - Extend the same behavior system-wide: - Microsoft Word / Office - Desktop editors (VS Code, Obsidian, Notepad++) - Creative sites like AO3, Medium, etc. - Messaging apps (Discord, Slack, etc.) - Implement via OS-level hooks or universal accessibility API integration.

Architecture Summary - **Frontend:** Browser extension → voice capture, DOM manipulation, AI overlay. - **Backend:** Flask/FastAPI microservice with STT (Whisper), LLM (DeepSeek/Mistral/Granite), and TTS pipeline. - **Future:** Electron or native app for cross-OS presence.

Goal: One persistent “Jarvis Dictator” overlay that can listen, understand, and edit *anywhere you write* — a cross-platform, voice-driven co-editor that feels alive.