Building an AI-powered virtual assistant that surpasses Siri's capabilities while avoiding scope creep is an ambitious project. Let’s break this down into actionable steps, tools, and a prioritized scope to keep you focused.

**1. Core Architecture Overview**

Here’s a high-level design for your assistant:

Copy

Voice/Text Input → Task Parser (LLM) → Task Execution Engine → Feedback Loop

**Key Components:**

* **Input Layer**: Voice/text detection.
* **Orchestration Layer**: LLM to parse intent and generate structured commands.
* **Execution Layer**: Python + macOS Automator/AppleScript + n8n workflows.
* **Security Layer**: User confirmation for critical actions.
* **Feedback Layer**: Voice/text responses and GUI notifications.

**2. Tools & Technologies**

**A. Input Handling**

* **Voice-to-Text (STT)**:
  + **Local**: Use Apple’s AVFoundation (via PyObjC) for low-latency macOS integration.
  + **Offline Model**: [Whisper.cpp](https://github.com/ggerganov/whisper.cpp) (optimized for Apple Silicon).
  + **Python Library**: SpeechRecognition with whisper backend for prototyping.
* **Text Input**: Build a hotkey-triggered CLI or GUI (e.g., [Tauri](https://tauri.app/) for lightweight Rust-based UI).

**B. Task Execution**

* **System-Level Automation**:
  + **AppleScript**: For Finder/file operations (e.g., renaming files).
  + **Automator**: Pre-built workflows triggered via Python.
  + **Python**: Use subprocess, os, shutil, and watchdog for file operations.
  + **Swift/Objective-C**: For deeper macOS integration (e.g., Calendar/Contacts access).
* **Workflow Automation**:
  + **n8n**: For multi-step workflows (e.g., "Email me the latest project files" → Compress → Send via Gmail).
  + **Shortcuts**: Integrate macOS Shortcuts for predefined tasks.

**C. LLM Integration**

* **Local LLM** (Recommended):
  + **Model**: Llama 3 70B (4-bit quantized) or Mistral 7B for faster inference on M2.
  + **Inference Engine**: [llama.cpp](https://github.com/ggerganov/llama.cpp) (optimized for Apple Silicon) or [Ollama](https://ollama.ai/" \t "_blank).
  + **Fine-Tuning**: Use a dataset of command→action pairs (e.g., 1,000 examples of "Move file X to folder Y" mapped to AppleScript).
* **Cloud LLM** (Fallback):
  + Use OpenAI API with function-calling for structured outputs (if local models underperform).

**D. Orchestration Layer**

* **Task Parsing**:
  + Use the LLM to convert natural language to structured JSON:

json

Copy

{

"action": "move\_file",

"params": {"source": "~/Downloads/report.pdf", "destination": "~/Documents"}

}

* **Task Router**: A Python dispatcher that maps actions to scripts/workflows.

**3. Scope & Features**

**Phase 1: MVP (2-4 Weeks)**

* **Input**:
  + Voice commands via Whisper.cpp.
  + Text input via hotkey-triggered CLI.
* **Core Actions**:
  + File operations (create/read/move/copy) via AppleScript/Python.
  + Launch apps/URLs.
  + Predefined n8n workflows (e.g., "Backup my project").
* **LLM**:
  + Local Llama 3/Mistral for intent parsing.
  + Fine-tune on 500 synthetic command examples.
* **Feedback**:
  + Basic voice responses with pyttsx3 or macOS say command.
  + System notifications with pync.

**Phase 2: Advanced Features (2-3 Months)**

* **Context Awareness**:
  + Track user context (e.g., "Open the file I edited yesterday").
  + Use vector databases (LanceDB/Chroma) to cache recent actions.
* **Multi-Step Workflows**:
  + "Summarize the PDF I just downloaded and email it to John" → Combine pdftotext, LLM summarization, and n8n.
* **GUI**:
  + Build a Spotlight-like overlay with Tauri.
  + Add confirmation dialogs for destructive actions.
* **Plugins**:
  + Support Python plugins for custom actions (e.g., "Analyze this CSV with Pandas").

**Phase 3: Polish (1 Month)**

* **Security**:
  + Implement user permissions (e.g., "Allow app to delete files?").
  + Sandbox critical operations.
* **Optimization**:
  + Precompile AppleScripts for speed.
  + Cache frequent LLM responses.
* **Cross-Platform**:
  + Wrap logic in Docker for future Windows/Linux support.

**4. Code Skeleton**

python

Copy

# Example dispatcher.py

from llm import parse\_command

from workflows import move\_file, send\_email

def execute\_task(command: str):

structured\_action = parse\_command(command) # LLM call

if structured\_action["action"] == "move\_file":

move\_file(\*\*structured\_action["params"])

elif structured\_action["action"] == "email":

send\_email(\*\*structured\_action["params"])

# Example LLM prompt

LLM\_PROMPT = """

Convert this command to JSON: "Move the latest screenshot to my Desktop"

Output: {"action": "move\_file", "params": {"source": "~/Downloads/\*.png", "destination": "~/Desktop"}

"""

**5. Avoiding Scope Creep**

* **Strict Prioritization**: Focus on Phase 1 MVP first. Reject features like "control smart home devices" until core is stable.
* **Tooling**:
  + Use poetry for Python dependency management.
  + Track progress with GitHub Projects.
* **Testing**:
  + Unit tests for critical actions (e.g., test\_move\_file).
  + Use Mac’s launchd to run the assistant as a background service.

**6. Key Challenges & Solutions**

* **LLM Latency**: Quantize models (GGUF format) and use Metal GPU acceleration.
* **Security**: Never execute commands like rm -rf without user confirmation.
* **File Context**: Use fzf or mdfind for fuzzy file searches.

**7. Final Recommendations**

* Start with a local LLM (Mistral 7B) for simplicity.
* Use Python as the glue language but offload performance-critical tasks to Swift or compiled binaries.
* For inspiration, study open-source projects like [MacGPT](https://github.com/msfrisbie/mac-gpt" \t "_blank) or [Alfred](https://www.alfredapp.com/).

This project is doable with your experience! Prioritize iterating on the MVP and refine the LLM’s intent parsing first. Let me know if you want deeper dives into specific components.