Local Buddy: Voice + Vision + Audio on macOS (Roadmap & Starter Tasks)

```
GOAL
Build a local macOS “buddy” app with voice in/out, optional vision, and on-device LLM. Start small with Disc

PHASE 0 — BASELINE (DONE OR IN-PROGRESS)
- Git installed and linked to GitHub
- Tailscale configured
- Docker installed (optional but useful)
- Brew available

PHASE 1 — START SMALL: DISCORD BOT (2–3 SESSIONS)
1) Repo setup
   - Create repo: discordbot1.0 (done)
   - Add .env with DISCORD_TOKEN
   - npm init -y

2) Minimal bot
   - Dependencies:
       npm i discord.js dotenv
   - Scripts:
       npm pkg set type="module" scripts.start="node src/bot.js"
   - Health check command: “/health” → “ok”
   - Run locally: npm start
   - Containerize later with Dockerfile (optional)

3) Operational habits
   - One ticket per change
   - git add -A && git commit -m "feat: X" && git push
   - Keep README with run instructions

PHASE 2 — LOCAL CLI “BUDDY” (VOICE IN/OUT) (3–5 SESSIONS)
Concept: A terminal app that listens to your mic, transcribes, sends to a local LLM, speaks the reply, and ca

1) Choose runtime
   - Python (simplest integration) with virtualenv
   - or Node.js if you prefer JS; Python is recommended for Whisper integrations

2) Local LLM backend
   - Install Ollama and pull one model:
       curl -fsSL https://ollama.com/install.sh | sh
       ollama pull llama3:8b-instruct
     (Optionally: deepseek-coder:7b for code tasks)

3) Speech-to-text (offline)
   - Option A: faster-whisper (GPU-accelerated on Apple Silicon)
       python -m venv .venv && source .venv/bin/activate
       pip install faster-whisper sounddevice numpy
     Notes: needs mic permission in macOS
   - Option B: Vosk (lower resource)
       pip install vosk sounddevice

4) Text-to-speech (macOS native)
   - Use “say” command for a first version:
       say "Hello from your buddy"
   - Python wrapper:
       import subprocess; subprocess.run(["say", text])

5) Wire it together (Python sketch)
   - mic → faster-whisper → text
   - text → Ollama (HTTP API at http://localhost:11434)
   - response → say
   - Command safety: only run whitelisted maintenance tasks

6) Safe maintenance commands (whitelist)
   - system_info: sw_vers; uname -a
   - brew_outdated: brew outdated
   - disk_free: df -h /
   - memory: vm_stat; memory_pressure
   - Never run destructive actions without explicit confirmation
7) File layout
```

```
   local-buddy/
     src/
       main.py          # orchestrates loop
       stt.py           # faster-whisper integration
       tts.py           # macOS say wrapper
       llm.py           # Ollama client
       skills.py        # whitelisted commands
     .env               # config (optional)
     README.md

PHASE 3 — VISION (OPTIONAL, 2–4 SESSIONS)
1) Screen capture permissions (System Settings → Privacy & Security → Screen Recording)
2) OCR for reading the screen or images
   - tesseract + pytesseract (brew install tesseract; pip install pytesseract)
3) Basic pipeline
   - capture screenshot → OCR → summarize via LLM
4) Camera input (optional)
   - OpenCV for webcam frames; ask user permission first
5) Safety
   - Only process local images/screen on demand
   - No outbound image data without explicit permission

PHASE 4 — TURN CLI INTO A MENUBAR APP (2–4 SESSIONS)
1) SwiftUI wrapper (preferred) or Electron
2) Hotkey to start/stop listening
3) Status icon in menu bar
4) Persist settings (model choice, TTS voice, mic device)
5) Log viewer

PHASE 5 — DOCKERIZATION & RUNPOD (OPTIONAL)
1) Containerize Discord bot (already planned)
2) For the local buddy, keep it on Mac for mic/tts; use RunPod only for heavy LLM if needed (tunnel via Tails
3) If using remote model:
   - Mac: stt/tts + thin client
   - RunPod: LLM server behind Tailscale
   - Configure client to talk to http://100.x.x.x:11434 or your custom port

DAY-BY-DAY STARTER TASKS (CONSERVATIVE PACING)
Day 1 (done): Git + Tailscale baseline
Day 2: Join RunPod to Tailscale; clone discordbot repo to RunPod; run npm install
Day 3: Add /health slash command to Discord bot; push; optional Dockerfile
Day 4: Start local buddy project (Python): create venv; install faster-whisper, sounddevice
Day 5: Add Ollama backend; test prompt roundtrip
Day 6: Add macOS TTS ("say"); full mic→LLM→speech loop
Day 7: Add 3 safe maintenance commands; require confirmation for any command that modifies the system

COMMAND BLOCKS (COPY WHEN READY)
A) Python venv and faster-whisper
python3 -m venv .venv
source .venv/bin/activate
pip install faster-whisper sounddevice numpy requests python-dotenv

B) Minimal Ollama client (HTTP POST)
import requests, os
def ask_ollama(prompt, model="llama3:8b-instruct"):
    r = requests.post("http://localhost:11434/api/generate", json={"model": model, "prompt": prompt, "stream
    r.raise_for_status()
    return r.json().get("response","")
print(ask_ollama("Say hello in one short sentence."))

C) macOS TTS helper
import subprocess
def speak(text: str):
    subprocess.run(["say", text])

D) Simple skill whitelist idea
ALLOWED = {
  "system_info": ["sw_vers"],
  "disk_free": ["df", "-h", "/"],
  "brew_outdated": ["brew", "outdated"]
}
# only run commands from ALLOWED; require confirmation for anything else

RISK GUARDRAILS
- All commands are local and explicit; never execute arbitrary shell from LLM output
- Require user confirmation ("Are you sure?") for anything that writes to disk, installs, or deletes
- Log any command run and its output to ./logs/
```

- Keep models local (Ollama) unless you intentionally connect to RunPod via Tailscale

WHAT TO BRING TO NEXT SESSION
- Confirmation that RunPod joined Tailscale
- discordbot1.0 cloned on RunPod and npm install completed
- Choice: start “local-buddy” in Python or stay on Discord bot for one more feature

TEACHER SUMMARY
We are leveling up in layers: Discord bot → local CLI buddy (voice) → optional vision → optional menubar a