Zero-to-Companion AI: Simple, Budget-Friendly Plan (Windows)

This blueprint gets you a talking assistant with memory, reminders, and room to grow—**with no coding experience**. Start fully free/offline, then swap components as you compare LLMs and voices.

Phase 0 — What You'll Build in 60–90 Minutes

Goal: A desktop voice assistant that: - Listens when you hold a hotkey - Transcribes your speech (accent-friendly) - Replies with natural speech - Remembers facts and todos locally - Can set reminders and checklists

Tech choices (all free/local): - **STT (speech-to-text):** *Faster-Whisper* (great with accents) - **LLM (brain):** *Ollama* running **Llama 3.1 8B** (good quality on most PCs) — you can later swap to any API model - **TTS (voice):** *Piper* (surprisingly natural, multiple voices) - **Memory:** SQLite + Chroma (embeddings) for long-term recall - **Scheduler/Reminders:** APScheduler + Windows notifications - **UI:** Simple tray app + system hotkey (no browser required)

Phase 1 — Install & Configure (Step-by-Step)

1) Prereqs (Windows)

- Install **Python 3.11+**
- Install **Git**
- (Optional but recommended) Install **Visual Studio Build Tools** (for wheels that might compile)

2) Local models

- **Ollama** (desktop LLM runner); after install, run: `ollama pull llama3.1:8b`
- **Piper** (TTS); download a voice model (e.g., `en_GB-northern_english_male` or `en_US-amy`)

3) Project scaffold

```
companion-ai/
  app/.env
  app/main.py
  app/agent/
    __init__.py
    stt.py        # Faster-Whisper
    tts.py        # Piper wrapper
    brain.py      # LLM abstraction (Ollama or API)
    memory.py     # SQLite + Chroma
    actions.py    # reminders, notes, calendar
    router.py     # intent + tool routing
  app/ui/
```

```
    tray.py       # system tray + hotkey
  data/
    memory.db
    vectordb/
    audio_tmp/
  requirements.txt
  README.md
```

4) Python deps (place in `requirements.txt`)

```
faster-whisper
sounddevice
numpy
pydub
simpleaudio
ollama
chromadb
sentence-transformers
apscheduler
pystray
pillow
keyboard
python-dotenv
rich
```

5) Environment (`app/.env`)

```
# LLM backend: "ollama" or "api"
LLM_BACKEND=ollama
OLLAMA_MODEL=llama3.1:8b
# If testing APIs later, drop keys here
OPENAI_API_KEY=
ANTHROPIC_API_KEY=
GOOGLE_API_KEY=
# Piper voice path
PIPER_VOICE=./voices/en_US-amy-medium.onnx
# Language hint for STT (helps with accents)
STT_LANGUAGE=en
```

Phase 2 — Minimal Working Code (copy/paste order)

> Paste these files in the specified paths. Then run `python app/main.py`.

app/agent/stt.py

```python
from faster_whisper import WhisperModel

class STT:
    def __init__(self, language="en"):
        # medium or large for best accent handling if your GPU/CPU allows
        self.model = WhisperModel("medium", compute_type="int8")
        self.language = language

    def transcribe(self, wav_path: str) -> str:
        segments, _ = self.model.transcribe(wav_path, language=self.language)
        return " ".join([s.text.strip() for s in segments]).strip()
```

app/agent/tts.py

```python
import subprocess, wave, contextlib
from pathlib import Path

class TTS:
    def __init__(self, voice_path: str):
        self.voice_path = voice_path

    def synth(self, text: str, out_path: str):
        out = Path(out_path)
        out.parent.mkdir(parents=True, exist_ok=True)
        # assumes piper.exe is on PATH; otherwise provide full path
        subprocess.run([
            "piper", "--model", self.voice_path, "--output_file", str(out)
        ], input=text.encode("utf-8"), check=True)
        return str(out)
```

app/agent/brain.py

```python
import os

class Brain:
    def __init__(self, backend="ollama", model="llama3.1:8b"):
        self.backend = backend
        self.model = model
        if backend == "ollama":
            import ollama
            self.client = ollama
        elif backend == "api":
            # example: OpenAI, Anthropic, or Google — add later
            raise NotImplementedError("API mode not set up yet")

    def chat(self, messages):
        system = messages[0]["content"] if messages and messages[0]
["role"]=="system" else ""
        user = messages[-1]["content"]
        if self.backend == "ollama":
            res = self.client.chat(model=self.model, messages=[
                {"role": "system", "content": system},
                {"role": "user", "content": user}
            ])
            return res["message"]["content"]
```

app/agent/memory.py

```
import sqlite3, json, time, os
from sentence_transformers import SentenceTransformer
import chromadb

class Memory:
    def __init__(self, db_path="data/memory.db", vecdir="data/vectordb"):
        os.makedirs(os.path.dirname(db_path), exist_ok=True)
        self.conn = sqlite3.connect(db_path)
        self._mktables()
        self.embed = SentenceTransformer("all-MiniLM-L6-v2")
        self.client = chromadb.PersistentClient(path=vecdir)
        self.col = self.client.get_or_create_collection("mem")

    def _mktables(self):
        c = self.conn.cursor()
        c.execute("CREATE TABLE IF NOT EXISTS facts (id INTEGER PRIMARY KEY,
key TEXT, value TEXT, ts INTEGER)")
        c.execute("CREATE TABLE IF NOT EXISTS todos (id INTEGER PRIMARY KEY,
text TEXT, done INTEGER, ts INTEGER)")
        self.conn.commit()

    def save_fact(self, key, value):
        ts=int(time.time())
        self.conn.execute("INSERT INTO facts(key,value,ts) VALUES(?,?,?)",
(key,value,ts))
        self.conn.commit()
        emb=self.embed.encode([f"{key}: {value}"]).tolist()[0]
        self.col.add(documents=[f"{key}: {value}"], embeddings=[emb],
ids=[f"fact-{ts}"])

    def search(self, query, k=5):
        emb=self.embed.encode([query]).tolist()[0]
        res=self.col.query(query_embeddings=[emb], n_results=k)
        return res.get("documents", [[]])[0]

    def add_todo(self, text):
        ts=int(time.time())
        self.conn.execute("INSERT INTO todos(text,done,ts) VALUES(?,?,?)",
(text,0,ts))
        self.conn.commit()

    def list_todos(self):
        return list(self.conn.execute("SELECT id,text,done FROM todos ORDER
BY id DESC"))
```

`app/agent/actions.py`

```
from apscheduler.schedulers.background import BackgroundScheduler
import datetime as dt
import win10toast

toaster = win10toast.ToastNotifier()

class Actions:
    def __init__(self):
        self.sched = BackgroundScheduler()
        self.sched.start()

    def remind_in(self, minutes: int, text: str):
        run_at = dt.datetime.now() + dt.timedelta(minutes=minutes)
        self.sched.add_job(lambda: toaster.show_toast("Reminder", text,
duration=10), 'date', run_date=run_at)
        return f"Okay, I'll remind you in {minutes} minutes."
```

`app/agent/router.py`

```
import re

class Router:
    def __init__(self, memory, actions):
        self.memory = memory
        self.actions = actions

    def route(self, text: str):
        # very simple intent routing
        if m:=re.match(r"remember (.+?) is (.+)", text, re.I):
            self.memory.save_fact(m.group(1), m.group(2))
            return "Noted. I'll remember that."
        if m:=re.match(r"remind me in (\d+) minutes? (.+)", text, re.I):
            return self.actions.remind_in(int(m.group(1)), m.group(2))
        if re.search(r"what do you remember|recall", text, re.I):
            docs=self.memory.search("user profile", k=3)
            return "Here's what I recall: \n- " + "\n- ".join(docs)
        if re.search(r"todos?", text, re.I):
            items=self.memory.list_todos()
            return "Your todos:\n" + "\n".join([f"[{('x' if d else ' ')}]
{t}" for _,t,d in items])
        return None  # let the LLM handle it
```

app/ui/tray.py

```
import io, sounddevice as sd, wave, tempfile
from pydub import AudioSegment

class Recorder:
    def __init__(self, samplerate=16000, channels=1):
        self.sr=samplerate; self.ch=channels

    def record_wav(self, seconds=10):
        audio = sd.rec(int(seconds*self.sr), samplerate=self.sr,
channels=self.ch, dtype='int16')
        sd.wait()
        # write WAV to temp file
        fd = tempfile.NamedTemporaryFile(suffix='.wav', delete=False)
        with wave.open(fd.name, 'wb') as wf:
            wf.setnchannels(self.ch); wf.setsampwidth(2);
wf.setframerate(self.sr)
            wf.writeframes(audio.tobytes())
        return fd.name
```

app/main.py

```
import os
from dotenv import load_dotenv
from agent.stt import STT
from agent.tts import TTS
from agent.brain import Brain
from agent.memory import Memory
from agent.actions import Actions
from agent.router import Router
from ui.tray import Recorder
from rich import print

load_dotenv()

stt = STT(language=os.getenv("STT_LANGUAGE","en"))
tts = TTS(voice_path=os.getenv("PIPER_VOICE"))
brain = Brain(backend=os.getenv("LLM_BACKEND","ollama"),
model=os.getenv("OLLAMA_MODEL","llama3.1:8b"))
mem = Memory(); acts = Actions(); router = Router(mem, acts)
rec = Recorder()

SYSTEM_PROMPT = (
    "You are Kyle, a warm, concise personal assistant and gaming companion. "
    "You keep answers short, ask clarifying questions when needed, and can
suggest todos/reminders."
)

print("[bold green]Hold 'Enter' in the console to talk. Release to stop (10s
max). Ctrl+C to exit.[/bold green]")

while True:
    try:
        input("\nPress Enter to record...")
        wav_path = rec.record_wav(seconds=10)
        user_text = stt.transcribe(wav_path)
        print(f"[cyan]You:[/cyan] {user_text}")
        # Try tool routing first
        routed = router.route(user_text)
        if routed is None:
            reply = brain.chat([
                {"role":"system","content": SYSTEM_PROMPT},
                {"role":"user","content": user_text}
            ])
        else:
            reply = routed
        print(f"[magenta]Kyle:[/magenta] {reply}")
        outwav = tts.synth(reply, "data/audio_tmp/response.wav")
        os.system(f'start /min wmplayer "{outwav}"')  # simplest playback on
Windows
    except KeyboardInterrupt:
        print("\nBye!"); break
```

Phase 3 — Voice & Personality (Non-robotic, free)

- **Piper voices:** Test several; some sound much more natural. Add light **prosody control** by chunking sentences and inserting small pauses (split on punctuation).
- **Memory of style:** Save user preferences like “speak casually,” “short replies,” “South African English phrasing,” etc., as facts.
- **Fillers (light touch):** Occasionally insert natural fillers ("yeah, totally" / "gotcha")—keep under 5% of sentences to avoid sounding fake.

Phase 4 — Long-Term Memory (Practical Setup)

1) **Daily journal file**: Append a structured JSON entry per day (meetings, tasks finished, notable chat).
2) **Memory buckets**: profile (name, preferences), projects (DevReady, Leafy Bites), routines (gym, guitar), games (New World).
3) **Auto-summaries**: After every 50 messages, summarize and embed the summary to reduce vector size. 4) **Recall rules**: On each user turn, search memory with the last user text; append top 3 snippets to the LLM context.

Phase 5 — Reminders & Integrations

- **Local reminders:** already covered via APScheduler + Windows toast. Add `remind me tomorrow at 9am` by parsing with `dateparser`.
- **Google Calendar (read-only first):** query upcoming events; speak the next 3 items.
- **Notion (notes & tasks):** create/read pages; mirror todos there.
- **Clipboard & screenshots:** hotkeys to capture and OCR with Tesseract for quick notes.

  Keep each integration behind a feature flag in `.env` so you can toggle on/off without breaking the app.

Phase 6 — Gameplay Helper (New World)

Start simple (avoid anything that violates ToS): - **Voice notes:** “Note: I’m at Depths dungeon; need Screaming Scales x3.” - **Checklists:** “Add task: craft 20 Asmodeum, remind me tomorrow.” - **Timers:** “Alert me in 8 minutes for food buff.” - **Overlay (later):** A small always-on-top window showing your active checklist and timers. - **OCR (optional):** Read specific resource counters from the screen to log progress; use only passive reads.

Phase 7 — Model Bake-Off (Pick your LLM cheaply)

Run identical tests across: - **Local**: Llama 3.1 8B (Ollama), Qwen2.5-7B, Mistral-7B-Instruct - **API (free/ cheap trials)**: small “mini” models from major vendors

Metric sheet (score 1–5 each): - Conversation naturalness - Accent understanding (via STT, held constant) - Task accuracy (reminders, notes) - Long-term recall (uses retrieved memory correctly) - Latency

Prompt pack for testing: - Persona: “You are a friendly South African assistant named Kyle. Keep replies under 70 words.” - Tasks: - “Remember my gym days are Mon/Wed/Fri.” - “Tomorrow at 7:30am remind me to pack gym clothes.” - “Create a New World checklist for my Asmodeum routine.” - “What did I ask you to remember about gym?”

Record scores in a CSV; choose the best cost/quality.

Phase 8 — Auto-Improver Loop (Assistant improves itself)

- **Log everything** (errors, slow responses, misunderstood intents).
- **Weekly self-review prompt:** Summarize logs; ask model to propose 3 actionable improvements. Save to `IMPROVEMENTS.md`.
- **One improvement per week** rule to keep scope small.

Phase 9 — Upgrades When Ready

- **Streaming voice** (near realtime): replace WAV flow with VAD + chunking; read while generating.
- **Better TTS**: switch to XTTS v2 (local) or a paid API if needed.
- **Better STT**: switch model size or cloud STT for tougher accents.
- **Web UI**: lightweight React panel with chat history, memory browser, todo manager.
- **Multi-agent skills**: a “Planner” role that decides whether to call tools or ask follow-ups.

Safety & Privacy

- All local by default (no cloud keys).
- A visible indicator when recording.
- A “delete last 24h” and “wipe memory” command.
- Keep logs in `data/logs/` with a simple retention window (e.g., 14 days).

Daily Use Cheatsheet

- **“Remember X is Y.”** → stores a fact
- **“Remind me in 12 minutes to check the oven.”** → toast reminder
- **“Add todo: renew car license.”** → todo list
- **“What do you remember about gym?”** → memory recall
- **“New World session checklist.”** → creates tasks + timers

---

Troubleshooting (common)

- **Audio device errors:** set input/output device explicitly in `sounddevice.default.device`.
- **Slow replies:** try `llama3.1:8b-q4` in Ollama, or reduce STT model size to `small`.
- **Piper not found:** add the folder to PATH or reference full path in `tts.py`.

Next Steps for You (today)

1) Install Python, Git, Ollama, Piper; pull `llama3.1:8b` and one Piper voice. 2) Create the folder structure and paste the files above. 3) Add paths in `.env` and run `python app/main.py`. 4) Say: “Remember my gym days are Mon/Wed/Fri.” then “What are my gym days?” 5) Try the Model Bake-Off when you’re comfortable.

You now have a private, non-robotic voice companion that runs locally and grows with you. Add one feature at a time and let Kyle help plan the next step. 🚀