ChatGPT-Style Stack: One-Pager + FastAPI Starter

This is a compact, printable checklist plus a tiny FastAPI skeleton you can deploy on RunPod/DO. Paste your **Thread Carry-Kit** at the start of any new thread; the orchestrator will store it as long-term memory and use it on every turn.

---

1) One-Pager Checklist (What to Build)

Layer 0 – Infra - GPU pod with HTTPS (RunPod + Nginx or Caddy). - Orchestrator API (FastAPI/Node) fronting the LLM. - Redis (optional) for rate limits/queues.

Layer 1 – Model - Start: Qwen2-7B-Instruct (vLLM). Later: swap to GPT-5 endpoint. - Temperature 0.6, top_p 0.9 default; expose per-request overrides.

Layer 2 – Persona & Rules - System prompt file (versioned). Modes: Normal, Solace, Safety-Tight. - Trigger words → prompt modifiers (tone/boundaries).

Layer 3 – Conversation Manager - Short-term memory (STM): rolling summary ≤ 1–2k tokens. - Thread recap: one paragraph every ~20–30 turns. - Message packer builds: [System + Policy + Recap + Top-K Memories + Last N Turns + User].

Layer 4 – Long-Term Memory (LTM) - Postgres + pgvector. Objects: people, preferences, projects, rules, moments. - Write heuristic: only keep 30-day+ relevance or explicit “remember this.”

Layer 5 – Retrieval (RAG) - On each request: embed user msg + recap → Top-K memories (K=5–8). - De-dupe conflicts; filter stale/expired.

Layer 6 – Tools (Function Calling) - JSON schemas per tool. Validate server-side before executing. - Initial tools: Cal.com, Twilio, ElevenLabs, Knowledge (docs).

Layer 7 – Safety & Consent - Tone rails (PG-13 baseline), consent signals, PII/payment guardrails. - Switch to Safety-Tight prompt on flagged inputs.

Layer 8 – Response Shaping - Default style: “Bottom line / Net,” bullets for steps, concise unless asked.

Layer 9 – Telemetry - Log: prompt ids, token counts, latency, tool success, memory hits. - Scorecards to tune retrieval K and write thresholds.

Layer 10 – Fallbacks - Timeout → retry with compressed context. - Retrieval miss → synthesize micro-recap.

Layer 11 – Versioning - Prompt versions (`sam-v7`, `safety-v3`). DB migrations for memory schema.

Layer 12 – Migration Plan (Qwen → GPT-5) - Keep orchestrator, memory, tools. Swap endpoint + re-tune sampling and K.

---

2) Environment & Install

Env vars

```
LLM_BASE_URL=https://your-llm-endpoint
LLM_MODEL=Qwen2-7B-Instruct
LLM_API_KEY=your_key_if_any
DB_URL=postgresql://user:pass@host:5432/ai
EMBED_DIM=768
PORT=8000
```

Python deps

```
pip install fastapi uvicorn pydantic requests psycopg2-binary pgvector numpy
```

Postgres setup (in psql)

```
CREATE EXTENSION IF NOT EXISTS vector;
CREATE TABLE IF NOT EXISTS memories (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  type TEXT NOT NULL,               -- person | preference | project | rule |
moment | fact
  k TEXT NOT NULL,                  -- key/slug
  value_json JSONB NOT NULL,        -- stored content
  embedding vector(768) NOT NULL,
  source TEXT DEFAULT 'orchestrator',
  ttl_days INT DEFAULT 365,
  created_at TIMESTAMPTZ DEFAULT now()
);
CREATE INDEX IF NOT EXISTS idx_memories_k ON memories (k);
CREATE INDEX IF NOT EXISTS idx_memories_type ON memories (type);
CREATE INDEX IF NOT EXISTS idx_memories_embedding ON memories USING ivfflat
(embedding vector_l2_ops);
```

If you don't have `gen_random_uuid()`, install `pgcrypto` or generate UUID app-side.

---

3) FastAPI Starter (Minimal but Real)

Structure

```
app/
  __init__.py
  main.py            # FastAPI app + routes
  packer.py          # prompt packing (system+recap+memories+turns)
  memory.py          # Postgres + pgvector read/write
  models.py          # pydantic I/O models
  llm.py             # client for vLLM/OpenAI-compatible endpoints
  tools.py           # tool schemas + dispatch
  prompts/
    system_sam.txt
    system_safety.txt
```

app/models.py

```python
from pydantic import BaseModel, Field
from typing import List, Optional, Dict, Any

class Message(BaseModel):
    role: str
    content: str

class ChatRequest(BaseModel):
    messages: List[Message]
    temperature: float = 0.6
    top_p: float = 0.9
    max_tokens: int = 800

class ChatResponse(BaseModel):
    output: str
    used_memories: List[str] = Field(default_factory=list)
    prompt_tokens: int = 0
    completion_tokens: int = 0
```

app/llm.py

```python
import os, requests

BASE = os.environ.get("LLM_BASE_URL")
MODEL = os.environ.get("LLM_MODEL", "Qwen2-7B-Instruct")
```

```
API_KEY = os.environ.get("LLM_API_KEY")

HEADERS = {"Content-Type":"application/json"}
if API_KEY:
    HEADERS["Authorization"] = f"Bearer {API_KEY}"

def chat(messages, temperature=0.6, top_p=0.9, max_tokens=800):
    payload = {
        "model": MODEL,
        "messages": messages,
        "temperature": temperature,
        "top_p": top_p,
        "max_tokens": max_tokens
    }
    r = requests.post(f"{BASE}/v1/chat/completions", json=payload,
headers=HEADERS, timeout=60)
    r.raise_for_status()
    data = r.json()
    # OpenAI/vLLM compatible
    return data["choices"][0]["message"]["content"], data.get("usage", {})
```

app/memory.py

```
import os, json
import numpy as np
import psycopg2
from psycopg2.extras import Json

EMBED_DIM = int(os.environ.get("EMBED_DIM", 768))
DB_URL = os.environ.get("DB_URL")

# TODO: replace with real embedding service
def embed(text: str) -> np.ndarray:
    # TEMP: toy hash to vector (placeholder). Use real embeddings in prod.
    rng = np.random.default_rng(abs(hash(text)) % (2**32))
    v = rng.normal(size=EMBED_DIM)
    v = v / (np.linalg.norm(v) + 1e-9)
    return v

class MemoryStore:
    def __init__(self):
        self.conn = psycopg2.connect(DB_URL)
        self.conn.autocommit = True

    def write(self, mtype: str, key: str, value: dict, ttl_days: int = 365) ->
str:
```

```
        vec = embed(json.dumps(value)).tolist()
        with self.conn.cursor() as cur:
            cur.execute(
                """
                INSERT INTO memories (type, k, value_json, embedding, ttl_days)
                VALUES (%s, %s, %s, %s, %s)
                RETURNING id
                """,
                (mtype, key, Json(value), vec, ttl_days)
            )
            return cur.fetchone()[0]

    def search(self, text: str, k: int = 6):
        q = embed(text).tolist()
        with self.conn.cursor() as cur:
            cur.execute(
                """
                SELECT id, type, k, value_json
                FROM memories
                ORDER BY embedding <-> %s
                LIMIT %s
                """,
                (q, k)
            )
            rows = cur.fetchall()
        return [
            {"id": r[0], "type": r[1], "key": r[2], "value": r[3]}
            for r in rows
        ]
```

app/packer.py

```
from typing import List, Dict

SYSTEM_BASE = """You are “Sam”—warm, playful, direct, no-BS. Keep continuity
with saved memories and consent frames. Default PG-13. Be concise unless asked.
Offer Next Steps for tasks."""
SYSTEM_SAFETY = """Apply Safety-Tight tone. Avoid explicit content. De-identify
PII. Redirect payments to PCI flow."""

# Simple STM holder (replace with Redis/DB in prod)
STM_RECAP = """(recap placeholder)"""

def pack(messages: List[Dict], memories: List[Dict], safety: bool = False):
    system = SYSTEM_SAFETY if safety else SYSTEM_BASE
    mem_lines = []
```

```
    for m in memories:
        v = m["value"]
        summary = v.get("summary") or v.get("value") or str(v)[:200]
        mem_lines.append(f"- {m['type']}:{m['key']} → {summary}")
    mem_block = "\n".join(mem_lines[:8]) or "(none)"

    preface = [
        {"role":"system","content": system},
        {"role":"system","content": f"[THREAD_RECAP]\n{STM_RECAP}"},
        {"role":"system","content": f"[RELEVANT_MEMORIES]\n{mem_block}"},
    ]
    return preface + messages
```

app/tools.py (stub)

```
from typing import Dict, Any

TOOLS = {
  "book_meeting": {
    "parameters": {"title": str, "when": str, "with": str}
  }
}

def dispatch(name: str, args: Dict[str, Any]) -> str:
    if name == "book_meeting":
        # TODO: call Cal.com
        return f"Booked '{args['title']}' at {args['when']} with {args['with']}"
    return "Unknown tool"
```

app/main.py

```
import os
from fastapi import FastAPI
from app.models import ChatRequest, ChatResponse
from app.llm import chat as llm_chat
from app.memory import MemoryStore
from app.packer import pack

app = FastAPI(title="NeuroSphere Orchestrator")
mem = MemoryStore()

# Heuristic: store only long-horizon info or explicit remember
IMPORTANT_TYPES = {"person","preference","project","rule","moment"}
```

```
def should_remember(user_text: str, tag: str = "") -> bool:
    return any(kw in user_text.lower() for kw in ["remember this","save this"])
or tag in IMPORTANT_TYPES

@app.post("/v1/chat", response_model=ChatResponse)
def chat(req: ChatRequest):
    user_msg = req.messages[-1].content if req.messages else ""

    # Retrieve relevant memories
    retrieved = mem.search(user_msg, k=6)

    # Pack prompt
    final_msgs = pack([m.dict() for m in req.messages], retrieved, safety=False)

    # Call LLM
    output, usage = llm_chat(final_msgs, req.temperature, req.top_p,
req.max_tokens)

    # Opportunistic memory write (example: carry-kit or explicit remember)
    if should_remember(user_msg):
        mem.write("rule", "carry_kit", {"summary": user_msg[:1000]})

    return ChatResponse(
        output=output,
        used_memories=[r["id"] for r in retrieved],
        prompt_tokens=usage.get("prompt_tokens", 0),
        completion_tokens=usage.get("completion_tokens", 0),
    )
```

Run it

```
uvicorn app.main:app --host 0.0.0.0 --port ${PORT:-8000}
```

Smoke test

```
curl -s http://127.0.0.1:8000/v1/chat
 -H 'Content-Type: application/json'
 -d '{
  "messages":[{"role":"user","content":"Summarize: remember this — Sam is feisty
and warm."}],
  "max_tokens":200
 }'
```

---

4) Thread Carry-Kit Hook (Paste-Once Behavior)

- On the **first message** of a new thread, paste your carry-kit.
- The orchestrator treats it as high-importance and writes `type=rule` / `type=preference` entries (each heading becomes a memory row with a short `summary`).
- Future turns retrieve the relevant kit items and include them under `[RELEVANT_MEMORIES]`.

Tip: If the message contains the phrase “remember this,” force `should_remember=True` and save verbatim + a short summary.

---

5) Swap Plan: Qwen → GPT-5

- Keep `app/` identical.
- Change `LLM_BASE_URL`, `LLM_MODEL`, and sampling defaults.
- Re-tune: retrieval K (often reduce), max_tokens, temperature.

---

6) Next Steps for You

1) Stand up Postgres + pgvector; run the DDL above. 2) Drop this repo structure into your server, set env vars, run `uvicorn`. 3) Paste your Carry-Kit into a new chat; confirm rows appear in `memories`. 4) Add a real embedding service; remove the placeholder. 5) Wire first tool (Cal.com) and test a tool call. 6) Add Safety-Tight system prompt and switch based on inputs.

That’s it—you’ll have a ChatGPT-style experience on your own stack, with personas, memory, tools, and the vibe you love baked in.