**GovSight Project Working Notes (End of Day – July 22, 2025)**

**Author:** Nyx (AI collaborator)  
**Partner:** Mike  
**Session #15 concluded**

This document captures everything we accomplished today, key design decisions, active code structure, known bugs, and the prioritized roadmap for tomorrow. You can upload this doc at the start of the next session so I can instantly re‑hydrate context and continue building without re‑asking the same background questions.

**0. Project North Star**

**Goal:** Build **GovSight**, a persistent, ChatGPT‑style research teammate for lobbying / government affairs / business intelligence work.

GovSight should:

* Remember prior conversations across sessions (long‑term memory).
* Use **Pinecone** + **SQLite** to store structured “facts,” summaries, watchlist tasks, and conversation logs.
* Perform **live, high‑accuracy web lookups** (SerpAPI) when memory is insufficient.
* Dynamically extract **constraints** (location, entity, time, topic, etc.) from each query — nothing hardcoded.
* Distinguish conversational chit‑chat from actionable research queries.
* Track and monitor entities (“add to watchlist”) and later proactively surface updates.
* Act as a *force multiplier* for a lobbying firm’s workflows: tracking officials, bills, deadlines, regulatory changes, etc.

**1. Where the Code Stands Right Now**

**Files currently in play** (as of end of day):

* talk.py – Main CLI conversation loop (large, ~1100 LOC). Handles intent, constraints, retrieval, web fallback, recall, watchlist creation, memory writes, etc.
* memory\_manager.py – SQLite persistence (sessions, turns, facts, watchlist). Also handles fact extraction using GPT and optional promotion to Pinecone.
* serp\_client.py – Earlier stand‑alone SerpAPI helper (most logic now inlined in talk.py; can re‑modularize later).
* web\_reasoner.py – Legacy; mostly superseded by new integrated SerpAPI+eval+synthesis pipeline.
* main.py / web\_ui.py – Not actively used in current CLI runs; will matter later once we modularize.

**1.1 Current Conversation Flow (High Level)**

(user turn) → classify\_interaction() → intent\_type + inherits\_context + needs\_retrieval

↓

extract\_constraints\_with\_gpt()

↓

if not needs\_retrieval:

→ chat / recall / command / followup-buffer

→ escalate? (update triggers, insufficient buffer)

else:

→ Pinecone retrieval (memory)

if score ≥ threshold → answer from memory

else → SerpAPI web fallback

• search

• fetch pages

• GPT relevance eval per doc

• synthesize multi-source answer

• log + optionally promote facts → memory

After answer: buffer push, memory log, optional fact extraction, optional watchlist create.

**1.2 Conversation Buffer**

* We maintain a **deque** of the last **12** turns.
* Used for: followup disambiguation, recall topic extraction, buffer‑QA (can we answer from conversation without retrieval?), and prompt conditioning.

**1.3 Active Context**

* Dict holding merged constraints from prior turns (when inherits=True).
* Carries extracted entities/topics across multi‑turn Q&A.
* Stores internal flags like \_recall\_last to pass signals between phases (e.g., whether user requested update during recall).

**2. What Works**

**✅ Retrieval Decision Logic**

* Classifier roughly separates **chat**, **followup**, **fact\_lookup**, **recall**, **command**.
* Recall no longer auto‑escalates just because input contains a question mark.

**✅ Constraint Extraction**

* Dynamic; not hardcoded to location fields.
* Robust to GPT noise via \_safe\_extract\_json().

**✅ SerpAPI Web Fallback**

* Top N search; fetch; per‑doc relevance eval via GPT; early stop when 3 high‑confidence hits.
* Multi‑source synthesis step producing ranked sources + answer.

**✅ Memory Persistence**

* Session open/close.
* Turn logging.
* Fact extraction → store → embed → upsert to Pinecone.
* Watchlist creation on command.

**✅ Conversation Mode vs Retrieval Mode**

* Behavior toggles based on intent & retrieval need.
* Acks (“cool thanks”) don’t trigger retrieval.

**3. Known Problems (Today’s Testing)**

**3.1 Location Contamination / Cross‑Jurisdiction Pollution (High Priority)**

* During early dev we ingested **Grandview, MO** facts (Mayor Leonard Jones, population goal 30K) into memory.
* Later recall of *Grandview, TX* pulled both TX and MO memory because the Pinecone filter was loose (any token match).
* SerpAPI contextual queries inherited polluted tokens → wrong answer synthesis (MO facts labeled TX).

**Fix Goals:**

1. Location‑aware filtering in memory recall & Pinecone retrieval.
2. Location gating in doc relevance eval & synthesis prompts.
3. Memory cleanup: demote or purge polluted facts.

**3.2 Correction Turns Escalate to Web Unnecessarily**

* When user corrects a fact (“X not Y”), classifier returns followup + needs\_retrieval=False, but followup branch escalated because buffer QA returned insufficient.
* Should instead accept user correction → promote corrected fact to memory (high confidence) → optionally recheck.

**3.3 Recall Summaries Are Noisy**

* Recall shows mixed context: summarization + raw fact lines (“Bill Houston: mayor of = Grandview, TX (conf 1)”).
* Should produce natural language recap and optionally list bullet facts by confidence.

**3.4 Fact Extraction Prompt Sensitivity**

* We had multiple KeyError: "entity\_name" or curly brace template issues when formatting prompts with literal JSON specs.
* Current version uses brace‑safe strings; keep that pattern.

**3.5 Performance / API Cost**

* Many OpenAI calls per turn (intent, constraints, recall, doc evals, synthesis, final answer, fact extraction...).
* Later we should batch doc evals or use smaller model for eval (e.g., gpt-4o-mini vs full) and reserve bigger model for synthesis when needed.

**4. Immediate Patch Plan (Tomorrow’s Work)**

**Priority A – Location‑Aware Safeguards (Stop MO→TX bleed)**

* Extract explicit/implicit location tokens from constraints + user input.
* Filter Pinecone memory hits: require *all* location tokens present when user specifies both city & state; fallback to any if only one provided.
* Extend \_doc\_eval\_prompt() to instruct: *score=0 if different jurisdiction.*
* Extend \_synth\_prompt() with same rule.

**Priority B – Clean Existing Memory Pollution**

* Add helper mem.purge\_facts(match\_text:str=None, like=None, id\_list=None).
* Or bulk soft‑downgrade facts containing grandview, mo, leonard jones, etc.
* Re‑embed after purge? (Optional; Pinecone update will supersede old.)

**Priority C – Recall Summary Formatter**

* Clean human‑readable recall answer that lists:
  + Known correct entries for **Grandview, TX**
  + Flag conflicting memory (Grandview MO) and ask if user wants to remove.

**Priority D – Correction Turn Handler**

* Detect pattern: X is ..., not Y or No, Y is wrong.
* Extract corrected fact → update memory (high conf) → mark conflicting memory low.

**Priority E – Retrieval Throttle for Corrections**

* If user gives correction and no explicit “check” token, respond conversationally + update memory; don’t hit web.
* If user correction includes “check”, “verify”, “confirm” → escalate.

**5. Test Protocol (Regression Checklist)**

Use this after each patch cycle.

**5.1 Clean Start**

python talk.py

Ask: who is the mayor of grandview tx?  
Expected: Bill Houston (web or memory depending on prior run).  
Should **NOT** mention Leonard Jones.

**5.2 Followup**

how long is his term?  
Expected: 2‑year term, current 2023–2025, next election Nov 4 2025.

**5.3 Non‑Question Chat**

cool thanks → Should respond conversationally; **no web lookup**.

**5.4 Recall Next Session**

New run; ask: do you remember our conversation about grandview tx?  
Expected: Summarize Bill Houston + election; no MO contamination; ask if check for updates.

**5.5 Update Trigger**

yeah check for updates → Should escalate web; confirm whether new info.

**5.6 Correction Turn**

Bill Houston is still mayor right? or Is he still mayor? → Should use memory + maybe web if time passed > configured staleness (future feature).

**6. Environment & Keys Recap**

**Do not store plaintext keys in version control.** Keep them in .env.

Current .env expected keys:

OPENAI\_API\_KEY=

PINECONE\_API\_KEY=

PINECONE\_ENV= # optional, some SDKs no longer require

CONGRESS\_API\_KEY=

SERP\_API\_KEY=

config.py loads .env and exposes:

OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")

PINECONE\_API\_KEY = os.getenv("PINECONE\_API\_KEY")

CONGRESS\_API\_KEY = os.getenv("CONGRESS\_API\_KEY")

PINECONE\_INDEX\_NAME = "gov-index"

SERPAPI\_API\_KEY = os.getenv("SERP\_API\_KEY") # alias supported in talk.py

DEFAULT\_OPENAI\_MODEL = "gpt-4o-mini"

DEFAULT\_EMBED\_MODEL = "text-embedding-3-small"

**7. Data & Storage Layout**

Current working expectation (please confirm tomorrow):

project\_root/

talk.py

memory\_manager.py

serp\_client.py

web\_reasoner.py

web\_ui.py

main.py

config.py

.env

logs/

web\_fallback.log

data/

govsight\_memory.db # SQLite

pinecone\_cache/ (optional future)

user\_files/ # raw docs for ingestion (future)

SQLite table summary (from memory\_manager.py):

* sessions(id, started\_at, ended\_at, summary)
* turns(id, session\_id, role, text, ts)
* facts(id, text, entity, topic, ts, confidence, source, pinecone\_id)
* watchlist(id, topic, entity\_name, frequency, created\_at, active)

**8. Memory Manager Behavior Summary**

**8.1 Session Lifecycle**

* open\_session() → row in sessions, returns session\_id.
* log\_turn(session\_id, role, text) → persist raw dialog.
* extract\_facts\_from\_turn(user\_text, answer\_text) → GPT extracts structured fact items.
* store\_facts() → insert into facts, embed, upsert to Pinecone.
* close\_session(session\_id) → GPT summarization of session; stored in sessions.summary.
* get\_latest\_session\_summary() used to seed new runs.

**8.2 Fact Schema (Python dict before store)**

{

"text": "Bill Houston is mayor of Grandview, TX.",

"entity": "Bill Houston",

"topic": "Grandview TX mayor",

"confidence": 0.9,

"source": "session:14" # or URL list

}

**8.3 Watchlist Workflow**

* detect via mem.detect\_watchlist\_from\_turn() (currently simple NLP; we improved command path in talk.py).
* stored w/ frequency (“weekly” default now).
* (Future) scheduler/service will re‑query web for watchlist topics.

**9. Prompt Engineering Conventions**

**Critical:** Many runtime errors earlier came from .format() applied to prompt strings containing {} JSON examples.

**Rule going forward:**

* Build prompts by concatenating string parts ("".join(parts)) or f‑string only where no literal braces appear.
* When you must show JSON schema, either escape braces {{ }} or append schema via variable constant.
* Validate prompts by quick python -m py\_compile talk.py before run.

**10. Open Design Questions for Mike (Tomorrow)**

1. **Memory pruning:** Should we keep *all* stored facts forever, or age out low-confidence ones?
2. **Location taxonomy:** Do you want a structured geocoder step (city,state,country IDs) to hard‑scope retrieval?
3. **Staleness policy:** When to auto‑check web again? (e.g., if memory fact older than 30 days & user asks status.)
4. **Security:** Are keys local only? Will this run on shared workstation? (Permissions / redaction?).
5. **File ingestion scale:** Roughly how many PDFs/CSVs do you expect to ingest? (Chunking strategy matters.)
6. **Watchlist cadence integration:** Cron? Windows Task Scheduler? Always‑running service?

**11. Tomorrow’s Action Plan (Suggested Order)**

**Step 0:** Confirm directory + DB path constants. **Step 1:** Memory cleanup – purge Leonard Jones (MO) contamination. **Step 2:** Location‑aware filtering in Pinecone recall + doc eval gating. **Step 3:** Recall summary rewrite (clean bullet list; highlight conflicting memory; ask if purge). **Step 4:** User correction override → promote corrected fact, downgrade conflicting. **Step 5:** Regression test run (#5 list above). **Step 6:** If time: begin Phase 1 refactor (utils + prompts extracted).

**12. One‑Click Memory Cleanup Helper (Preview Snippet)**

*Do not run yet; we’ll wire tomorrow.*

# memory\_manager.py (future)

def purge\_facts(match\_substr: str, hard\_delete: bool = False):

sql = "SELECT id, text, confidence FROM facts WHERE LOWER(text) LIKE ?"

rows = \_db.fetchall(sql, (f"%{match\_substr.lower()}%",))

if hard\_delete:

\_db.execute("DELETE FROM facts WHERE id IN ({})".format(

",".join(str(r[0]) for r in rows)

))

else:

for rid, \_, \_ in rows:

\_db.execute("UPDATE facts SET confidence=? WHERE id=?", (0.05, rid))

\_db.commit()

return rows

**13. Example Session Transcript (Good Behavior Target)**

**User:** who is the mayor of grandview tx?  
**GovSight:** Bill Houston (source: City of Grandview, TX). Term 2023–2025. Next election Nov 4 2025. Want to track?  
**User:** yeah track him.  
**GovSight:** Added to watchlist: Bill Houston (weekly).  
**User (next day):** remember our conversation about grandview tx?  
**GovSight:** We discussed Bill Houston is the mayor (term thru Nov 2025) and added him to your watchlist. Check for updates?

**14. Quick Troubleshooting Cheatsheet**

**KeyError: "entity\_name"** – usually prompt literal brace formatting bug in memory\_manager.\_FACT\_EXTRACT\_PROMPT. Use brace‑safe templates.

**Intent always fact\_lookup** – Classifier may fail JSON; confirm \_safe\_extract\_json() returning dict; add debug print of raw.

**Infinite web calls** – If min\_high\_conf too high and every doc eval < cutoff, we loop across all top N; increase cutoff or early stop after N.

**Wrong city disambiguation** – Confirm location tokens extracted & enforced; check memory for polluted rows.

**15. Scratchpad – Facts Captured Today (Unvetted)**

These are *raw* and may include errors; we’ll clean tomorrow.

* **Bill Houston** – Mayor of **Grandview, TX**. Resident since 2019. Master electrician / business owner. Term 2023–2025. Next election **Nov 4 2025**. (Confidence: high; multiple city website sources.)
* **Grandview, MO** memory pollution – Leonard D. Jones Jr., 30K population growth goal tied to tax incentives. Should be flagged as **Grandview, MO**, not TX.
* Watchlist created: topic=mayor (imprecise), also implicitly track Bill Houston (we should normalize watchlist entry to entity + location).

**16. What I Need From You Tomorrow**

When you start the next session, please:

1. Upload this document (copy/paste or re‑upload file) so I regain full context.
2. Tell me whether to **purge** or **downgrade** the Grandview, MO memory contamination.
3. Confirm your tolerance for missed recall vs false recall (strict location vs loose).
4. Confirm DB file path (default data/govsight\_memory.db?).
5. Give me a green light to ship the **Location Patch (A+B+C)** code.

Once I get those answers, we’ll move fast.

**17. Closing**

You’ve already got something powerful: a local, persistent, research‑grade AI able to blend memory and live web intelligence. Today we stabilized recall behavior, added conversation buffering, reduced unneeded web calls for acks, and got multi‑turn Q&A working end‑to‑end.

Tomorrow we harden correctness (no more Grandview MO bleed), improve usability, and start carving this into a reusable agent framework.

I’ll be ready when you are.

— **Nyx** 🛰️