# GovSight Refactor Hand‑Off Guide (R0→R1 Checkpoint)

**Author:** Mike + Nyx (AI pair)  
**Platform:** Windows (Storage F:\govsight), Python venv active  
**Date/Checkpoint:** July 23, 2025 (end of R0 + mid‑R1a)  
**Purpose:** Hand this doc to a *future* AI session to quickly re‑prime everything we’ve done so far and what remains. This prevents losing momentum when context resets.

## Quick Executive Summary

We are refactoring a monolithic GovSight research/LLM assistant into a modular Python package. We completed **R0 (package bootstrap + module invocation shim)** and began **R1a (new persistent memory layer)**. The new memory schema was added, but bootstrap ran into a **legacy DB schema conflict** (old facts table missing new columns), so we need a migration patch before proceeding.

You (future AI) must **not lose** the following facts:

* Project root: ``.
* We run via **module invocation**: python -m govsight.cli.chat\_cli --profile dev.
* Legacy engine still lives in `` and is currently what runs; the CLI is a **shim**.
* We created a new **Memory API** under govsight/memory/ (sessions, messages, facts, files).
* Conversation logging should be **always on**, store **full content**, and **facts are versioned**.
* User loves **heavy inline comments** and expects future code drops to be self‑explaining.

Until migration is fixed, memory bootstrap throws: sqlite3.OperationalError: no such column: subject\_slug (see “Open Issue” below).

## Terminology (User Asked!)

**Slug:** A machine‑friendly, lowercase, punctuation‑safe identifier derived from a human label. Example: “Grandview, TX” → grandview\_tx. Used as subject\_slug in facts.

**Shim:** A thin compatibility wrapper that lets old code keep working while new code is phased in. Our govsight.cli.chat\_cli module is a shim that calls legacy talk.main().

## Current Directory Layout of Interest

*(Non‑exhaustive; only what future AI needs)*

F:\govsight\  
├─ govsight\ # new package root  
│ ├─ \_\_init\_\_.py # minimal, re‑exports version  
│ ├─ \_initbase.py # package notes  
│ ├─ logging\_utils.py # central logger  
│ ├─ config\  
│ │ ├─ \_\_init\_\_.py # re‑exports Settings/load\_settings  
│ │ ├─ \_initconfig.py # config docs  
│ │ └─ settings.py # Settings dataclass + env/legacy loader  
│ ├─ cli\  
│ │ ├─ \_\_init\_\_.py  
│ │ └─ chat\_cli.py # module entrypoint shim → talk.main()  
│ └─ memory\ # R1a memory package (new)  
│ ├─ \_\_init\_\_.py # re‑exports Memory  
│ ├─ \_initmemory.py # memory design notes  
│ ├─ schema.py # DDL + bootstrap  
│ ├─ records.py # dataclasses  
│ └─ memory.py # Memory API (sessions/messages/facts/files)  
│  
├─ talk.py # legacy chat engine (still active)  
├─ memory\_manager.py # legacy "memory" (pre‑R1)  
├─ config.py # legacy flat config (fallback values)  
├─ smoke\_memory\_test.py # larger smoke test script (user created)  
├─ smoke\_small.py # recommended small smoke test (may not exist yet)  
│  
├─ data\ # persistent DB dir  
│ └─ memory.db # \*\*existing DB\*\* w/ legacy tables; now conflicting  
│  
└─ logs\ # log files

## Decisions Locked In (Carry Forward!)

| Decision Area | Current Setting | Notes |
| --- | --- | --- |
| Invocation style | **Module path** (python -m govsight.cli.chat\_cli) | talk.py remains callable direct, but module path is canonical going forward. |
| Conversation logging | **Always ON** | Every user/assistant/tool turn logged. |
| Message content storage | **Full raw text** | Redaction layer later if needed. |
| Facts | **Structured + versioned** | subject\_type, subject\_slug, attr, value, etc.; new insert supersedes old. |
| Default confidence for user corrections | **0.90** | High but not absolute; allows later verification upgrade. |
| DB | **SQLite at** `| Keep; will migrate. | | Vector store | \*\*Pinecone\*\* (index:gov-index`) | Already in use in legacy code. |
| Comments | **Heavy** | All new code must be tutorial‑grade; future AI must not assume context. |

## Open Issue (Blocking R1a): Schema Conflict on Bootstrap

**Symptom:** Running smoke\_memory\_test.py produced:

sqlite3.OperationalError: no such column: subject\_slug

**Cause:** Existing data/memory.db (legacy) has an old table named facts without our new columns (subject\_slug, etc.). Our bootstrap creates indexes referencing those columns; SQLite errors because table already exists with incompatible schema.

### Confirm the Problem

Run from project root:

python - <<PY  
import sqlite3  
conn = sqlite3.connect('data/memory.db')  
cur = conn.cursor()  
cur.execute("PRAGMA table\_info(facts)")  
print('legacy facts columns:', cur.fetchall())  
PY

You’ll likely see a different shape (e.g., id, key, value…).

### Migration Strategy (Non‑destructive)

We’ll migrate in place by *renaming* legacy table → facts\_legacy, then creating new facts table.

**SQL migration:**

ALTER TABLE facts RENAME TO facts\_legacy;

Then re‑run the memory bootstrap (instantiate Memory(settings) again). New schema will create fresh facts table.

#### Optional: Port Rows Forward

If legacy facts\_legacy contains useful data, we can map its columns into the new schema. For example, if it stored text keys like mayor:grandview\_tx we parse into subject\_slug + attr.

## Minimal Migration Script (migrate\_memory\_db.py)

Create in project root and run once. Adjust mapping logic if legacy table differs.

"""One‑time migration: upgrade legacy memory.db to R1 schema.  
  
Steps:  
1. Detect existing `facts` table structure.  
2. If it does NOT match R1 schema, rename to `facts\_legacy`.  
3. Instantiate Memory() to bootstrap new schema.  
4. Optional: Attempt to port any rows from legacy format.  
  
\*\*\* BACK UP data/memory.db before running! \*\*\*  
"""  
  
import os  
import shutil  
import sqlite3  
  
from govsight.config import load\_settings  
from govsight.memory import Memory  
  
BACKUP = 'data/memory.db.bak'  
LEGACY\_TABLE = 'facts\_legacy'  
  
s = load\_settings()  
print('Using DB:', s.db\_path)  
  
# 1. Backup  
if not os.path.exists(BACKUP):  
 shutil.copyfile(s.db\_path, BACKUP)  
 print('Backup created ->', BACKUP)  
else:  
 print('Backup already exists; not overwriting.')  
  
conn = sqlite3.connect(s.db\_path)  
cur = conn.cursor()  
  
# 2. Inspect existing facts schema  
cur.execute("PRAGMA table\_info(facts)")  
cols = [r[1] for r in cur.fetchall()]  
print('Existing facts cols:', cols)  
  
EXPECTED = {"subject\_type","subject\_slug","attr","value","source","confidence","status","provenance","latest"}  
if not EXPECTED.issubset(set(cols)):  
 print('Legacy facts schema detected; renaming table...')  
 cur.execute("ALTER TABLE facts RENAME TO %s" % LEGACY\_TABLE)  
 conn.commit()  
else:  
 print('Facts table already R1‑compatible; no rename needed.')  
  
# 3. Close legacy conn before bootstrap  
conn.close()  
  
# 4. Bootstrap new schema (will create fresh tables)  
mem = Memory(s)  
print('R1 schema ensured.')  
  
# 5. Optional: Port rows (disabled by default; enable if mapping known)  
# TODO: read from facts\_legacy and insert with mem.remember\_fact(...)  
  
print('Migration complete.')

**Run:**

python migrate\_memory\_db.py

Then retry the smoke test.

## Smoke Test Script (Small, Copy‑Safe)

**File:** smoke\_small.py

from govsight.config import load\_settings  
from govsight.memory import Memory  
  
s = load\_settings()  
print("Settings loaded OK; db:", s.db\_path)  
  
m = Memory(s)  
sid = m.start\_session(profile=s.profile)  
print("session id:", sid)  
  
m.log\_message(sid, "user", "ping")  
m.log\_message(sid, "assistant", "pong")  
  
slug = m.subject\_slug\_city("Grandview", "TX")  
m.remember\_fact(subject\_type="city", subject\_slug=slug, attr="mayor", value="Bill Houston", source="user")  
  
fact = m.get\_fact(slug, "mayor")  
print("fact:", fact)

**Expected output (after migration):**

Settings loaded OK; db: data/memory.db  
session id: <int>  
fact: FactRecord(id=..., subject\_type='city', subject\_slug='grandview\_tx', attr='mayor', value='Bill Houston', ...)

## Next Planned Work (R1b → R2 → R3)

### R1b – Wire Legacy Chat to Memory

* Import Memory in talk.py.
* On startup: session\_id = mem.start\_session(profile=settings.profile).
* Before sending to LLM: mem.log\_message(session\_id, "user", user\_text).
* After response: mem.log\_message(session\_id, "assistant", answer\_text).
* Detect user correction patterns (e.g., “No, the mayor is …”) → call mem.remember\_fact().

### R1c – Use Facts During Answers

* Before hitting web/Pinecone, check mem.get\_fact(slug, attr).
* If found, surface as high‑confidence answer (cite memory vs web).

### R2 – Retrieval Cascade Extraction

* Move Pinecone + SERP + doc retrieval out of talk.py into govsight/retrieval/.
* Provide ranked results + provenance tags.

### R3 – Persona / Prompt Builder

* Inject user preferences (tone, detail level).
* Insert memory slices (recent session summary + stable facts) into prompt.
* Add follow‑up question heuristics to make local agent feel “alive.”

## Sensitive Data Handling Reminder

User printed API keys in a test run. **Do NOT store or echo actual keys** in logs, code, or this doc. Always refer to them symbolically (e.g., OPENAI\_API\_KEY=<redacted>). When re‑priming future sessions, ask user to re‑set environment variables or re‑upload config.py if needed.

## Quick Start Checklist for Future Session

When Mike returns and uploads code again:

1. **Confirm environment**: Windows path F:\govsight; active venv.
2. **Verify package import:** python -m govsight.cli.chat\_cli --profile dev works.
3. **Check memory layer:** run python smoke\_small.py; if schema error, run migration.
4. **Ask user:** Do you want to wire talk.py to new memory now (R1b)?
5. **Ask user:** Are we ready to migrate legacy facts data or ignore? (If ignore, start clean.)
6. **Proceed** with R1b patch diff after confirmation.

## What To Ask Mike (Future Session Kickoff Script)

Copy/paste this as a first message when re‑priming:

Hi Mike — re‑priming GovSight. We left off mid‑R1a (memory layer) after installing the modular package. Your DB threw a schema conflict (subject\_slug missing) from a legacy facts table. Did you run the migration yet? If not, I can patch it now. Also: ready to wire talk.py so each chat logs to persistent memory? Let me know.

## Appendix A – Migration Diff Stub (talk.py R1b Hook)

*(Do NOT apply yet; reference only.)*

**Imports:**

from govsight.config import load\_settings  
from govsight.memory import Memory

**Startup:**

settings = load\_settings(profile="dev") # or args.profile  
mem = Memory(settings)  
session\_id = mem.start\_session(profile=settings.profile)

**In main loop (user turn):**

mem.log\_message(session\_id, "user", user\_input)

**After model answer:**

mem.log\_message(session\_id, "assistant", answer\_text)

**Correction capture (pseudo):**

if parsed\_mayor\_correction:  
 slug = mem.subject\_slug\_city(city, state)  
 mem.remember\_fact(subject\_type="city", subject\_slug=slug, attr="mayor", value=new\_name, source="user")

We will produce a real diff once user says “do it.”

## Appendix B – Slug Rules (GovSight Standard)

* Lowercase.
* Replace spaces with \_.
* Strip commas, periods.
* Append state 2‑letter postal (lowercase) for geo if supplied.
* Multi‑word states convert to postal code upstream (preferred).
* Example mapping: “Los Angeles County, CA” → los\_angeles\_county\_ca.

## Appendix C – Shim Rules

* Minimal wrapper.
* Accept old entrypoint, call new.
* Print deprecation notice (optional toggle).
* Remove only when user confirms.

**End of Hand‑Off Guide.**

If you (future AI) need *any* code from this checkpoint (settings.py, memory.py, migration script, smoke tests), ask Mike to re‑upload or request the snippets by name.