Agentic Historian — Implementation Tasks Playbook

Generated on 2025-12-05

One page per task: ID, priority, dependencies, quick description, and step-by-step examples aligned to uv, LangGraph, Pydantic, Streamlit, and PyTorch + Hugging Face + PEFT/LoRA.

AH-01 — Project Initialization (uv + repo skeleton)

Priority	P0
Dependencies	None

What needs to be done

Initialize the repository using uv, create the full folder/file structure, and add baseline tooling configs (ruff, mypy, pytest) plus .env.example. Ensure the project installs and smoke tests run.

Step-by-step

- Initialize the project with uv and create a src-layout package.
- Create the agreed repo structure (docs/, src/, tests/, scripts/, etc.).
- Add baseline dependencies and dev tools via uv.
- Add minimal runnable entrypoints (Streamlit placeholder + graph runner placeholder).
- Validate: uv sync + ruff + pytest.

Examples

commands

```
# 1) init
uv init agentic-historian
cd agentic-historian

# 2) add deps (minimal set first)
uv add streamlit pydantic httpx python-dotenv
uv add langgraph langchain
uv add torch transformers datasets peft accelerate
uv add --dev ruff mypy pytest

# 3) create package
mkdir -p src/agentic_historian
touch src/agentic_historian/__init__.py

# 4) run checks
uv sync
uv run ruff check .
uv run pytest -q
```

pyproject_snip

```
# pyproject.toml (excerpt)
[project]
name = "agentic-historian"
version = "0.1.0"
requires-python = ">=3.11"
dependencies = [
  "streamlit",
  "pydantic",
  "httpx",
  "python-dotenv",
  "langgraph",
  "langchain",
  "torch",
  "transformers",
  "datasets",
  "peft",
  "accelerate",
]
```

```
[tool.ruff]
line-length = 100
target-version = "py311"

[tool.pytest.ini_options]
testpaths = ["tests"]
```

AH-02 — Core Schemas & Shared LangGraph State (Pydantic)

Priority	P0
Dependencies	AH-01

What needs to be done

Implement strict Pydantic models for SearchResult, EvidenceBundle, Findings, Citations, and the shared AgentState for the graph. Include validators and test coverage.

Step-by-step

- Create Pydantic v2 models in src/agentic_historian/types/schemas.py.
- Define AgentState in src/agentic_historian/graph/state.py and reuse shared schemas.
- Add validators: evidence_id format ^E\d+$, URL must be http(s), confidence within [0,1].
- Write unit tests that assert validation failures for invalid data.

Examples

code

```
# src/agentic_historian/types/schemas.py
from __future__ import annotations
from pydantic import BaseModel, HttpUrl, Field, field_validator
from typing import Literal

EvidenceVerdict = Literal["supported","not_supported","contested","insufficient"]

class SearchResult(BaseModel):
    title: str
    snippet: str
    url: HttpUrl
    display_domain: str
    rank: int = Field(ge=1, le=50)

class EvidenceItem(BaseModel):
    id: str
    title: str
    snippet: str
    url: HttpUrl

    @field_validator("id")
    @classmethod
    def validate_id(cls, v: str) -> str:
        import re
        if not re.match(r"^E\d+$", v):
            raise ValueError("EvidenceItem.id must look like E1, E2, ...")
        return v

class Finding(BaseModel):
    claim: str
    verdict: EvidenceVerdict
    evidence_ids: list[str] = Field(default_factory=list)

class EvidenceBundle(BaseModel):
    evidence_items: list[EvidenceItem] = Field(default_factory=list)
    findings: list[Finding] = Field(default_factory=list)
    overall_verdict: EvidenceVerdict
```

test

```
# tests/test_schemas_validation.py
import pytest
from pydantic import ValidationError
from agentic_historian.types.schemas import EvidenceItem

def test_evidence_id_validation():
    with pytest.raises(ValidationError):
        EvidenceItem(id="BAD", title="t", snippet="s", url="https://example.com")
```

AH-03 — Config + Logging + Caching (Demo Reliability)

Priority	P0
Dependencies	AH-01, AH-02

What needs to be done

Centralize environment configuration, add structured logging, and implement a simple disk cache for search queries to reduce quota usage and stabilize demos.

Step-by-step

- Add config loader reading env vars (GOOGLE_API_KEY, GOOGLE_CSE_ID, OFFLINE_MODE, TRUSTED_DOMAINS_ONLY, MAX_RESULTS).
- Implement a disk cache helper keyed by query + date bucket (JSON).
- Wrap each node with timing instrumentation and store metrics in state.run_metadata.

Examples

code

```
# src/agentic_historian/config.py
from pydantic import BaseModel
from dotenv import load_dotenv
import os

load_dotenv()

class Settings(BaseModel):
    google_api_key: str | None = os.getenv("GOOGLE_API_KEY")
    google_cse_id: str | None = os.getenv("GOOGLE_CSE_ID")
    offline_mode: bool = os.getenv("OFFLINE_MODE","0") == "1"
    trusted_domains_only: bool = os.getenv("TRUSTED_DOMAINS_ONLY","0") == "1"
    max_results: int = int(os.getenv("MAX_RESULTS","10"))

SETTINGS = Settings()
```

cache

```
# src/agentic_historian/utils/cache.py
import json, hashlib, datetime
from pathlib import Path

CACHE_DIR = Path(".cache/search")
CACHE_DIR.mkdir(parents=True, exist_ok=True)

def _key(query: str) -> str:
    day = datetime.date.today().isoformat()
    h = hashlib.sha256(f"{day}|{query}".encode()).hexdigest()[:24]
    return h

def load(query: str) -> dict | None:
    p = CACHE_DIR / f"{_key(query)}.json"
    return json.loads(p.read_text()) if p.exists() else None

def save(query: str, payload: dict) -> None:
    p = CACHE_DIR / f"{_key(query)}.json"
    p.write_text(json.dumps(payload, ensure_ascii=False, indent=2))
```

AH-04 — Google Custom Search Tool Client (httpx, no scraping)

Priority	P0
Dependencies	AH-01, AH-02, AH-03

What needs to be done

Implement a robust Google Custom Search JSON API client. Normalize responses to SearchResult. Support timeouts and typed exceptions. No page fetching/scraping.

Step-by-step

- Implement GoogleSearchClient(search) that calls the Custom Search JSON endpoint.
- Read API key and CSE ID from config.
- Validate auth/quota errors and raise typed exceptions.
- Normalize 'items' into SearchResult objects.
- Unit test with mocked httpx responses.

Examples

code

```
# src/agentic_historian/tools/google_search.py
from __future__ import annotations
import httpx
from agentic_historian.config import SETTINGS
from agentic_historian.types.schemas import SearchResult

class AuthError(RuntimeError): ...
class QuotaError(RuntimeError): ...
class NetworkError(RuntimeError): ...

class GoogleSearchClient:
    BASE_URL = "https://www.googleapis.com/customsearch/v1"

    def __init__(self, client: httpx.Client | None = None) -> None:
        self._client = client or httpx.Client(timeout=10.0)

    def search(self, query: str, num_results: int = 10) -> list[SearchResult]:
        if not SETTINGS.google_api_key or not SETTINGS.google_cse_id:
            raise AuthError("Missing GOOGLE_API_KEY or GOOGLE_CSE_ID")

        params = {"key": SETTINGS.google_api_key, "cx": SETTINGS.google_cse_id,
                  "q": query, "num": min(max(num_results, 1), 10)}

        try:
            r = self._client.get(self.BASE_URL, params=params)
        except httpx.RequestError as e:
            raise NetworkError(str(e)) from e

        if r.status_code in (401, 403):
            raise AuthError(r.text)
        if r.status_code == 429:
            raise QuotaError(r.text)
        r.raise_for_status()

        data = r.json()
        items = data.get("items", []) or []
        out: list[SearchResult] = []
```

```
for i, it in enumerate(items, start=1):
    out.append(SearchResult(
        title=it.get("title",""),
        snippet=it.get("snippet",""),
        url=it.get("link","https://example.com"),
        display_domain=(it.get("displayLink") or ""),
        rank=i
    ))
return out
```

AH-05 — Router Node (routing + explanation metadata)

Priority	P0
Dependencies	AH-02

What needs to be done

Implement deterministic routing: fact_check vs clarify vs out_of_scope. Save decision rationale into run_metadata for UI display.

Step-by-step

- Define simple routing rules and a small list of non-historical intents to detect.
- Write router_node(state) that updates route and run_metadata['router'].
- Add unit tests with representative queries.

Examples

code

```
# src/agentic_historian/graph/nodes/router.py
from __future__ import annotations
from agentic_historian.graph.state import AgentState

NON_HISTORICAL_HINTS = ("poem", "song", "joke", "recipe", "code this", "write me")

def router_node(state: AgentState) -> AgentState:
    q = state.user_query.strip()
    meta = {"reason": None}

    if not q:
        state.route = "clarify"; meta["reason"] = "empty_query"
    elif len(q) < 8 or q.lower() in ("did it happen?", "is it true?"):
        state.route = "clarify"; meta["reason"] = "underspecified_query"
    elif any(h in q.lower() for h in NON_HISTORICAL_HINTS):
        state.route = "out_of_scope"; meta["reason"] = "non_historical_intent"
    else:
        state.route = "fact_check"; meta["reason"] = "default_fact_check"

    state.run_metadata["router"] = meta
    return state
```

AH-06 — Researcher Node (query expansion + dedupe + cache)

Priority	P0
Dependencies	AH-03, AH-04, AH-05

What needs to be done

Expand the user query into up to 3 search queries, call the Google tool, merge and dedupe results, and store SearchResult list in state.

Step-by-step

- Implement query expansion: user_query -> list[str].
- Use caching before calling the API; save normalized results.
- Merge results across queries and dedupe by URL.
- Optionally add trusted-domain mode using site: filters.

Examples

code

```
# src/agentic_historian/graph/nodes/researcher.py (sketch)
from __future__ import annotations
from agentic_historian.graph.state import AgentState
from agentic_historian.tools.google_search import GoogleSearchClient
from agentic_historian.config import SETTINGS
from agentic_historian.utils import cache

TRUSTED = "(site:.gov OR site:.edu OR site:britannica.com OR site:loc.gov)"

def expand_queries(q: str) -> list[str]:
    q = q.strip()
    return [q, f"{q} date", f"{q} primary source"][:3]

def researcher_node(state: AgentState, client: GoogleSearchClient | None = None) -> AgentState:
    client = client or GoogleSearchClient()
    queries = expand_queries(state.user_query)
    state.search_queries = queries

    seen, merged = set(), []
    for q in queries:
        q_eff = f"{q} {TRUSTED}" if SETTINGS.trusted_domains_only else q
        cached = cache.load(q_eff)
        batch = cached["results"] if cached else [r.model_dump() for r in client.search(q_eff, SE
        if not cached:
            cache.save(q_eff, {"results": batch})

        for item in batch:
            if item["url"] not in seen:
                seen.add(item["url"])
                merged.append(item)

    state.search_results = merged
    state.run_metadata.setdefault("researcher", {})["num_results"] = len(merged)
    return state
```

AH-07 — Fact Analyst Node (evidence pack + verdict + contradictions)

Priority	P0
Dependencies	AH-02, AH-06

What needs to be done

Convert SearchResults into a compact EvidenceBundle (E1..En), extract findings, score credibility, detect contradictions, and assign an overall verdict.

Step-by-step

- Score sources using domain heuristics and cross-source agreement signals.
- Create EvidenceItems with IDs E1..EN (keep top N=6–10).
- Extract simple claims (years/numbers) from snippets using lightweight regex.
- If conflicting claims appear for the same field, mark overall as contested.

Examples

code

```
# src/agentic_historian/graph/nodes/fact_analyst.py (sketch)
from __future__ import annotations
import re
from urllib.parse import urlparse
from agentic_historian.graph.state import AgentState
from agentic_historian.types.schemas import EvidenceBundle, EvidenceItem, Finding

DATE_RE = re.compile(r"\b(\d{4})\b")
TRUSTED_DOMAINS = (".gov", ".edu", "britannica.com", "loc.gov")

def domain_score(url: str) -> int:
    host = urlparse(url).netloc.lower()
    return 2 if any(host.endswith(d) for d in TRUSTED_DOMAINS) else 1

def fact_analyst_node(state: AgentState) -> AgentState:
    sr = state.search_results or []
    if not sr:
        state.evidence_bundle = EvidenceBundle(evidence_items=[], findings=[], overall_verdict="in
        return state

    scored = sorted([(domain_score(it["url"]), it) for it in sr], key=lambda x: x[0], reverse=True
    evidence_items, years = [], set()

    for idx, (_, it) in enumerate(scored[:8], start=1):
        evidence_items.append(EvidenceItem(id=f"E{idx}", title=it["title"], snippet=it["snippet"],
        m = DATE_RE.search(it.get("snippet","") or "")
        if m: years.add(m.group(1))

    overall = "supported" if len(evidence_items) >= 2 else "insufficient"
    if len(years) >= 2: overall = "contested"

    findings = [Finding(claim="Evidence collected from live search results.", verdict=overall,
                        evidence_ids=[e.id for e in evidence_items])]
    state.evidence_bundle = EvidenceBundle(evidence_items=evidence_items, findings=findings, overa
    return state
```

AH-08 — Writer Node (LoRA model + citation enforcement)

Priority	P0
Dependencies	AH-02, AH-07

What needs to be done

Load base model + LoRA adapter (PEFT) and generate a final answer constrained to evidence. Enforce evidence-ID citations and safe fallback when missing.

Step-by-step

- Implement HF model loader with optional PEFT adapter attachment.
- Build strict writer prompt that includes evidence items and forces [E#] citations.
- Generate output; validate citations exist.
- Fallback to safe uncertainty when citations are missing.

Examples

code

```
# src/agentic_historian/llm/prompts.py (sketch)
def build_writer_prompt(user_query: str, evidence_bundle) -> str:
    evidence_txt = "\n".join([f"{e.id}: {e.title} — {e.snippet} ({e.url})" for e in evidence_bund
    return f"""You are The Agentic Historian.
"
Rules:
- Use ONLY the evidence items below.
- Every factual statement MUST include at least one citation like [E1].
- If evidence is insufficient or contested, say so explicitly.

"
Question: {user_query}

"
Evidence:
{evidence_txt}

"
Write:
- Answer
- Confidence (0-1)
- Evidence IDs used
- Limitations
"""
```

validator

```
# citation enforcement idea (sketch)
import re
CITE_RE = re.compile(r"\[E\d+\]")

if not CITE_RE.search(answer_text):
    answer_text = "I can't verify this claim from the retrieved evidence. Please refine the questi
```

AH-09 — LangGraph Assembly + CLI Runner

Priority	P0
Dependencies	AH-05, AH-06, AH-07, AH-08

What needs to be done

Build the LangGraph pipeline graph and expose run_graph(). Add a small CLI runner for debugging without Streamlit.

Step-by-step

- Assemble StateGraph(AgentState) with nodes router/researcher/analyst/writer.
- Add conditional routing after router (END for clarify/out_of_scope).
- Expose run_graph(user_query) returning final AgentState.
- Create scripts/run_graph_cli.sh or a python -m entry.

Examples

code

```
# scripts/run_graph_cli.sh
uv run python -c "from agentic_historian.graph.graph import run_graph; print(run_graph('When did W
```

AH-10 — Streamlit UI (Chat + Evidence Panel)

Priority	P0
Dependencies	AH-09

What needs to be done

Implement the Streamlit chat UI, plus an evidence panel showing E1..En with URLs, verdict badge, confidence, and run metadata. Include OFFLINE_MODE toggle.

Step-by-step

- Build chat UI using st.chat_input + st.session_state messages.
- Call run_graph() and render final answer and evidence items.
- Add verdict/confidence display and expandable run_metadata.
- Add OFFLINE_MODE switch or auto-use SETTINGS.offline_mode.

Examples

code

```
# scripts/run_app.sh
uv run streamlit run src/agentic_historian/app/streamlit_app.py
```

AH-11 — Training Pipeline (TruthfulQA → LoRA SFT)

Priority	P1
Dependencies	AH-01, AH-02, AH-08

What needs to be done

Prepare TruthfulQA instruction pairs and fine-tune a LoRA adapter; save adapters + logs.

Step-by-step

- Load TruthfulQA (HF datasets) and format instruction-response samples.
- Tokenize and train LoRA adapter using PEFT.
- Save adapter to models/lora_adapters// and record config/metrics.

Examples

code

```
# scripts/train_lora.sh (sketch)
uv run python -m agentic_historian.training.train_lora \
  --base_model mistralai/Mistral-7B-Instruct-v0.2 \
  --output_dir models/lora_adapters/run1 \
  --epochs 1
```

AH-12 — Evaluation Harness (Offline + E2E)

Priority	P1
Dependencies	AH-09, AH-11

What needs to be done

Add offline eval on TruthfulQA and an end-to-end evaluation runner for curated historical queries. Output JSONL results and summary stats.

Step-by-step

- Create a small curated query set (10–30) as JSONL for E2E.
- Run run_graph() over the set and save results (answer, verdict, citations, confidence).
- Compute simple summary stats: citation rate, contested rate, avg latency.

Examples

code

```
# data/processed/e2e_eval.jsonl (example lines)
{"query":"When did the Berlin Wall fall?"}
{"query":"Did Cleopatra live closer to the moon landing than to the Great Pyramid?"}
```

AH-13 — Testing + CI (uv-based quality gate)

Priority	P1
Dependencies	AH-01, AH-02, AH-04, AH-09

What needs to be done

Add unit tests and GitHub Actions CI running uv sync, ruff, mypy (optional strictness), and pytest. Ensure all network calls are mocked.

Step-by-step

- Write unit tests for router, schemas, tool parsing, analyst verdicting, graph smoke.
- Add CI workflow using uv and run checks.
- Ensure tests run without API keys (mock httpx / tool calls).

Examples

ci

```
# .github/workflows/ci.yml (sketch)
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: astral-sh/setup-uv@v3
      - run: uv sync
      - run: uv run ruff check .
      - run: uv run pytest -q
```