SymptSpher — Full-Scale Chatbot Documentation (OpenAI NLU)

*Version 1.0*

This document describes the complete architecture, components, data, APIs, deployment, and runbook for the SymptSpher training chatbot using OpenAI for Natural Language Understanding (NLU).

# 1. Purpose & Learning Goals

* Train medical students in structured history-taking through a realistic chat simulation.
* Provide explainable scoring and feedback: Diagnosis Credit, Positive Evidence Recall (PER), and Interaction Length (IL).
* Leverage the DDXPlus dataset (structured evidence codes) for consistent cases and assessment.

# 2. Dataset & Codebooks

## 2.1 DDXPlus Dataset

* Case fields: AGE, SEX, PATHOLOGY (ground truth), DIFFERENTIAL\_DIAGNOSIS (list of [disease, probability]), EVIDENCES (E\_\* tokens), INITIAL\_EVIDENCE (chief complaint).
* Splits: release\_train\_patients, release\_validate\_patients, release\_test\_patients.

## 2.2 Codebooks

* release\_evidences.json — evidence dictionary: code\_evidence (E\_\*), question\_en, data\_type (B/C/M), possible-values (V\_\*).
* release\_conditions.json — disease dictionary: canonical names, ICD-10, and attributes.

Evidence token formats:

* E\_XX → binary present (e.g., E\_53 = cough).
* E\_XX\_@\_V\_YYY → categorical/multi-choice value (e.g., pain location = V\_167).
* E\_XX\_@\_<number> → numeric/ordinal value (e.g., pain intensity = 5).

# 3. System Architecture

High-level components and data flow:

Student Browser  
 │  
 │ UI (HTTP)  
 ▼  
Streamlit UI (port 8501) — chat, grading views  
 │ calls JSON  
 ▼  
FastAPI API (port 8000, served by Uvicorn) — REST & WebSocket (optional)  
 ├─ /v1/patient/start (start session; pick case; return chief complaint)  
 ├─ /v1/patient/ask (OpenAI NLU → E\_\* code [+ value] → answer; reveal evidence)  
 ├─ /v1/professor/grade (normalize diagnosis → Credit, PER, IL)  
 └─ /v1/professor/report (transcript + feedback)  
Domain services:  
 • Codebook loader (decode E\_\*/V\_\* to text)  
 • NLU: OpenAI (tool calling; structured outputs)  
 • Diagnosis normalizer (local SBERT or OpenAI embeddings)  
 • Scoring engine (Credit, PER, IL)  
 • Feedback generator (missed, high-yield questions)  
Storage/Infra:  
 • Postgres (sessions, turns, grades; optional in-memory store for dev)  
 • Redis (optional cache/queues)  
 • Docker (local) → Kubernetes (cloud)

# 4. Repository Structure

symptspher/  
├─ apps/  
│ ├─ api/  
│ │ ├─ main.py  
│ │ ├─ routers/  
│ │ │ ├─ patient.py  
│ │ │ └─ professor.py  
│ │ ├─ domain/  
│ │ │ ├─ codebook.py  
│ │ │ ├─ nlu\_openai.py  
│ │ │ ├─ normalize\_openai.py  
│ │ │ ├─ scoring.py  
│ │ │ └─ store.py  
│ │ ├─ models/  
│ │ │ └─ schema.py  
│ │ └─ requirements.txt  
│ └─ ui/  
│ └─ streamlit\_app.py  
├─ data/  
├─ cases/  
├─ infra/  
└─ .env.example

# 5. Dependencies & Environment

API requirements.txt:

fastapi==0.118.0  
uvicorn[standard]==0.30.6  
pydantic==2.12.0  
python-dotenv==1.0.1  
openai==1.51.2

UI requirements.txt:

streamlit==1.38.0  
requests==2.32.3

.env keys:

OPENAI\_API\_KEY=sk-...  
OPENAI\_NLU\_MODEL=gpt-4o-mini  
ALLOWED\_ORIGINS=http://localhost:8501

# 6. NLU with OpenAI

* Map student text to (feature\_head, value): ('E\_53', None), ('E\_55','V\_167'), or ('E\_56', 5).
* Use function/tool calling for structured outputs and reliability.
* Set temperature=0 for deterministic behavior in assessment.

Abbreviated tool schema:

{"type":"function","function":{"name":"map\_to\_evidence","parameters":{"type":"object","properties":{"feature":{"type":"string"},"value":{"anyOf":[{"type":"string"},{"type":"integer"},{"type":"null"}]}}}}}

Flow:

1. Streamlit → /v1/patient/ask.
2. FastAPI → OpenAI with tool schema.
3. OpenAI → {feature, value}.
4. FastAPI composes token and checks case evidence; decodes with codebook; returns answer + revealed findings.

# 7. Scoring & Feedback

* Diagnosis Credit = 100 × p(selected disease) from case differential.
* PER = (# positive heads revealed) / (# positive heads in case) × 100.
* IL = number of student→patient turns (report as count or invert for score).
* Composite score example: 0.6\*Credit + 0.3\*PER + 0.1\*(100 - IL).
* Feedback: top-3 missed evidence heads or info-gain heuristic.

# 8. API Specification (v1)

Patient:

* POST /v1/patient/start — body: {session\_id}; returns age, sex, initial\_evidence.
* POST /v1/patient/ask — body: {session\_id, text}; returns answer, revealed[], decoded[].

Professor:

* POST /v1/professor/grade — body: {session\_id, diagnosis\_text}; returns normalized\_dx, credit, per, il, score, feedback.
* GET /v1/professor/report/{session\_id} — optional.

# 9. Runbook (Local)

1. Create venv; install API/UI requirements.
2. Set OPENAI\_API\_KEY as environment or in .env.
3. Start API: python -m uvicorn apps.api.main:app --reload --port 8000
4. Start UI: streamlit run apps/ui/streamlit\_app.py
5. Browse http://localhost:8501

# 10. Deployment & Ops

* Docker Compose (api, ui, db, redis).
* Kubernetes: HPA, Ingress + TLS, managed Postgres.
* Prometheus/Grafana for metrics; structured JSON logs.
* Rate limiting on /ask; server-side OpenAI key only.

# 11. Security & Safety

* Display disclaimer: 'Training tool — not medical advice.'
* Refuse treatment/dosage queries; keep focus on history-taking.
* Minimal PII; session log purges; JWT for instructor routes.

# 12. Troubleshooting

* 404 at root → use /docs or /v1/\* routes.
* Module not found → install into active venv.
* FileNotFoundError → verify relative paths or use Path(\_\_file\_\_).
* CORS error → add Streamlit origin to ALLOWED\_ORIGINS.
* NLU UNKNOWN → check API key, model name, and tool schema.

# 13. Key Code References (Snippets)

App wiring (apps/api/main.py):

app.state.codebook = Codebook('data/release\_evidences.json','data/release\_conditions.json')  
app.state.nlu = OpenAINLU()  
app.state.store = SessionStore('cases/sample\_cases.jsonl')

Patient /ask core:

feature, value = await nlu.parse(body.text)  
token = feature if value is None else f"{feature}\_@\_{value if isinstance(value,int) else (value if value.startswith('V\_') else 'V\_'+value)}"  
has\_feature = any(token == ev or feature == ev.split('\_@\_')[0] for ev in case['evidences'])