Neuro-Symbolic Memory Network (NSMN)

A Hybrid Embedding–Symbolic Reasoning System for Fact Storage, Retrieval, and Multi-Hop Inference

1.System Overview

The Neuro-Symbolic Memory Network (NSMN) is a hybrid reasoning system combining neural embeddings, symbolic information extraction, and multi-hop graph reasoning.
User text input is decomposed into structured triples via a rule-based pattern extractor. Each triple receives both a symbolic representation (subject, predicate, object) and a dense vector embedding generated by a SentenceTransformer model.

Facts are stored simultaneously in:
(1) a vector database (MemoryStore) supporting cosine similarity retrieval and
(2) a directed symbolic knowledge graph enabling multi-hop reasoning.

Queries follow a retrieve–compose–verify pipeline:
Neural retrieval identifies relevant facts, a reasoning engine composes candidate multi-step chains using both local symbolic joins and graph shortest-path exploration, a deterministic
verifier assigns confidence. The final explanation is rendered by converting chain steps into interpretable natural-language statements.

This project implements a Neuro-Symbolic Memory Network (NSMN)
a unified memory system combining:

1. Neural embeddings for semantic similarity search
2. Symbolic triples (subj–pred–obj) extracted from text
3. Graph-based multi-hop reasoning
4. Local reasoning composition
5. Optional LLM-based augmentation + verification
6. Deterministic scoring + ranking of reasoning chains

The system uses Streamlit for insertion and querying.
Core implementation is spread across:

- *store.py* — Vector memory store
- *core.py* — Main symbolic-neural reasoning engine
- *app.py* — Streamlit interface
- *requirements.txt* — dependencies

---

2. Module-Level Architecture

2.1 MemoryStore (Vector Database Layer)

Purpose

A minimal in-memory vector store enabling

- fact insertion
- dynamic embedding matrix maintenance
- brute-force cosine similarity search

Data Model

Each fact becomes a **MemoryItem** with:
id: UUID
symbol: {subj, pred, obj, raw_text, source}
emb: np.ndarray | None
meta: metadata dict
ts: insertion timestamp

Embedding Matrix Management

MemoryStore maintains an **_emb_matrix**, rebuilt lazily only when needed.

Cosine Similarity Search

search_bruteforce(q_emb, k) implements:

1. L2-normalization of query + matrix
2. Dot-product similarity
3. Argsort for top-k selection

Mathematical formulation:

$$sim(q, x_i) = \frac{q}{\| q \|} \cdot \frac{x_i}{\| x_i \|}.$$

This provides deterministic, model-agnostic ranking.

2.2 SymbolicMemoryNetwork (Core Reasoning Engine)

This is the heart of the system.
It integrates:

- **symbol extraction**
- **embedding generation**
- **graph creation**
- **retrieval**
- **chain composition**
- **verification**
- **LLM augmentation (optional)**

2.2.1 Pattern-Based Symbol Extraction

Method: *_extract_symbols(text)*
Uses ~20 linguistic regex patterns (identity, causation, transformation, inclusion, requirements, etc.) to produce structured triples:

{
pred: "causes" | "part_of" | ...,
subj: normalized left entity,
obj: normalized right entity,
subj_raw: raw text,
obj_raw: raw text,
src: "pattern"
}

Patterns allow approximate IE without external NLP.
If no pattern matches, a fallback "stmt" triple stores the entire sentence.

2.2.2 Embedding Generation

Method: *_embed(text)*
Uses SentenceTransformer (all-MiniLM-L6-v2) to convert:
"pred subj obj"
→ 384-dim vector.
This ensures retrieval is predicate-aware, not just entity-aware.

2.2.3 Symbol Storage

Method: *add_text(text)*
Pipeline:

1. Extract triples via regex
2. Compute embedding for each triple
3. Store into *MemoryStore*
4. Insert nodes+edges into a directed NetworkX graph
    - Node = normalized entity
    - Edge attributes: pred, full metadata

This graph becomes the substrate for multi-hop reasoning.

2.2.4 Retrieval

Method: *retrieve(query, k)*

1. Embed the query string
2. Perform brute-force cosine search in MemoryStore
3. Return top-k triples with their similarity scores

This layer ensures neural recall of semantically related facts.

2.2.5 Multi-Step Reasoning

Method: *compose(retrieved, query, max_hops)*

The reasoning engine constructs **candidate chains** from two mechanisms:

A. Local Symbolic Composition
If:
fact1.obj == fact2.subj
Then chain:
A --p1--> B --p2--> C

Score = mean(similarities of both facts).
This implements basic forward chaining.

B. Graph Multi-Hop Reasoning
Using NetworkX *shortest_simple_paths*, up to *max_hops* edges:
Advantages:

- deterministic
- guarantees simplest explanation first
- avoids cycles
- multi-hop generalization

Chain example:
X → Y → Z → Q
Each hop is converted back to a symbolic triple.
Score heuristic:

$$score = 0.5 + 0.4 \times (\text{chain length})$$

This rewards deeper explanations

C. Single-Fact Fallback
If no multi-hop or compositions, system uses retrieved facts independently.

2.2.6 Chain Verification

Method: *verify_chain(chain, query)*

Two layers:

1. Heuristic Confidence Score
Deterministic:

$$conf = clamp(0.5 + 0.1 \cdot len(chain), 0.2, 0.95)$$

Larger chains → higher confidence.
2. Optional LLM Verification
If enabled:

- System passes facts + question to a T5 model
- Asks for 1-sentence verification rationale
- Not used for ranking, only explanation

This avoids LLM hallucination affecting reasoning correctness.

2.2.7 LLM-Based Fact Augmentation (Optional)

Method: *_augment_with_llm(query, retrieved)*
If weak reasoning occurs, the system asks an LLM for:

"bridging facts" that logically connect retrieved facts to the question.

LLM returns JSON describing subject–predicate–object triples, which are inserted back into memory.
This forms a **neuro-symbolic self-extending memory system**.

2.2.8 Final Ranking + Explanation

Method: *answer(query)*
Candidate chains are ranked by:

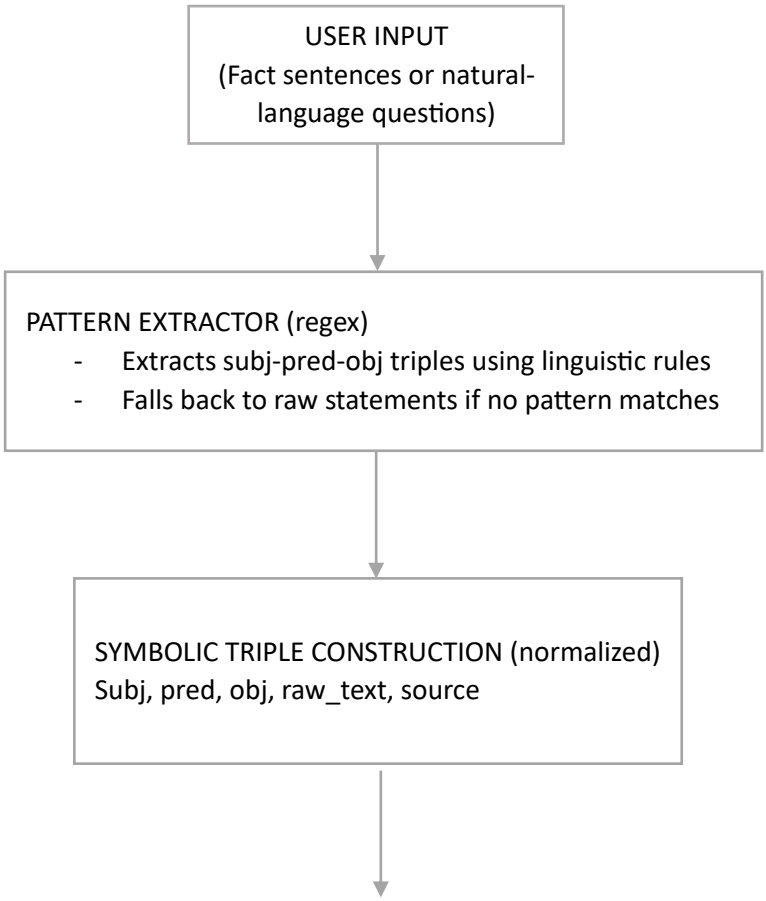
$$score = 0.4 \cdot raw\_score + 0.35 \cdot lexical\_overlap(query, chain) + 0.25 \cdot length\_bonus$$

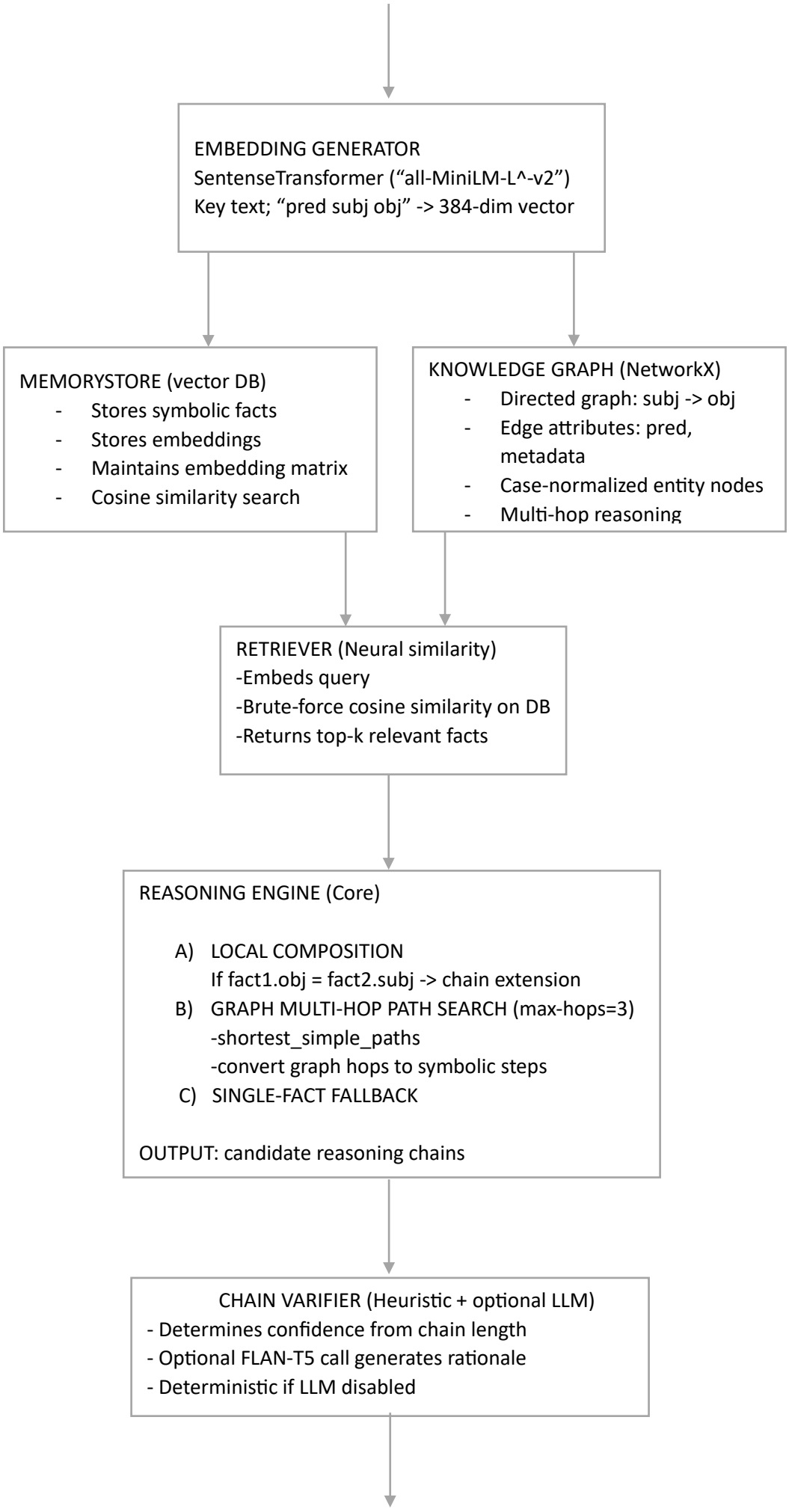
Where:

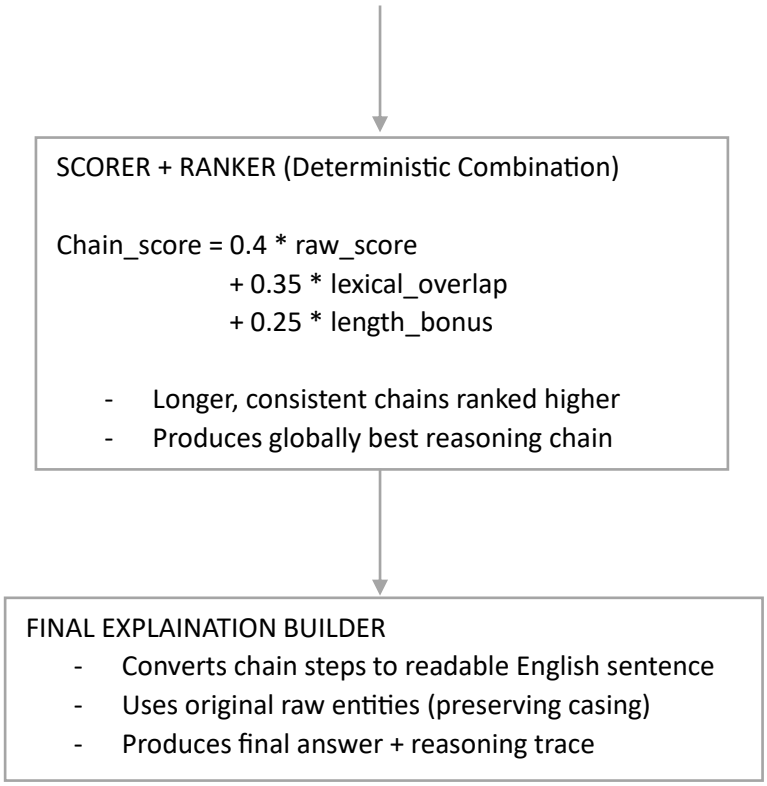
- lexical overlap = token intersection
- length_bonus favors deeper reasoning (academic-style logic)

Then:

- Best chain is rendered to readable English
- UI displays candidate chains + verification info




EMBEDDING GENERATOR
SentenseTransformer (“all-MiniLM-L^-v2”)
Key text; “pred subj obj” -> 384-dim vector
MEMORYSTORE (vector DB)
- Stores symbolic facts
- Stores embeddings
- Maintains embedding matrix
- Cosine similarity search
KNOWLEDGE GRAPH (NetworkX)
- Directed graph: subj -> obj
- Edge attributes: pred, metadata
- Case-normalized entity nodes
- Multi-hop reasoning
RETRIEVER (Neural similarity)
-Embeds query
-Brute-force cosine similarity on DB
-Returns top-k relevant facts
REASONING ENGINE (Core)
A) LOCAL COMPOSITION
If fact1.obj = fact2.subj -> chain extension
B) GRAPH MULTI-HOP PATH SEARCH (max-hops=3)
-shortest_simple_paths
-convert graph hops to symbolic steps
C) SINGLE-FACT FALLBACK
OUTPUT: candidate reasoning chains
CHAIN VARIFIER (Heuristic + optional LLM)
- Determines confidence from chain length
- Optional FLAN-T5 call generates rationale
- Deterministic if LLM disabled



---

3. Graph Semantics

The directed graph stores semantic relationships:
node = concept
edge.pred = relation type
edge.meta = provenance

This enables:

- causal chains
- entity linking
- semantic propagation
- conceptual clustering

The system avoids uncontrolled graph explosion by only adding triples from:

- pattern extraction
- LLM-generated facts (optional)
- user input

---

4. Streamlit Application Layer

The UI provides:

Left Panel

- Text input

- Buttons:
    - **Add to memory** → calls core.add_text()
    - **Ask** → calls core.answer()
- Displays:
    - best chain explanation
    - candidate chains
    - retrieved facts

Right Panel

- Memory browser (list all facts)
- Clear memory button

---

5. System Strengths

1. Full Neuro-Symbolic Pipeline

This architecture blends:

- neural similarity
- symbolic logic
- graph reasoning
- LLM augmentation

2. Deterministic, Inspectable Reasoning

Unlike LLMs:

- every chain is traceable
- every inference is explainable
- no hallucination distortion

3. Modular & Extensible

Swapping components is trivial:

- embedding model
- pattern extractor
- graph constraints
- reasoning scoring

---

6. Limitations

- pattern dependency,
- no learning mechanism,
- limited linguistic coverage,
- no contradiction detection,
- no probabilistic interpretation,
- performance not optimized for large-scale memory.

7. Future Directions

Neural Extensions

- relation extraction via transformers,
- entity linking across sentences,
- transformer-based paraphrase compression.

Symbolic Enhancements

- graph visualization,
- rule induction,
- contradiction detection.

Hybrid Models

- LLM verification layers,
- uncertainty scoring,
- reinforcement for chain ranking.

---

8. Conclusion

The NSMN architecture integrates neural semantic retrieval with explicit symbolic reasoning in a fully inspectable pipeline. By isolating vector search, symbol extraction, graph construction, multi-hop reasoning, and verification into modular components, the system supports deterministic, interpretable inference while remaining extensible to stronger encoders or LLM-based augmentation.

Its end-to-end structure pattern-based extraction, embedding-driven recall, graph-based composition, heuristic verification, and explanation rendering creates a functional neuro-symbolic reasoning loop suitable for research-level applications in explainable AI, factual QA, and hybrid memory architectures.