ATT&CK Knowledge Graph

A Neuro-Symbolic System for Threat Intelligence

January 29, 2026

Abstract

The ATT&CK Knowledge Graph is a neuro-symbolic system that combines MITRE ATT&CK threat intelligence data as an RDF knowledge graph with vector embeddings for intelligent threat intelligence querying. This document describes the system architecture, data flow, and key design decisions that enable security teams to analyze attack narratives, identify relevant techniques, and receive remediation recommendations through both structured (SPARQL) and semantic (vector similarity) query mechanisms.

Contents

1 Introduction

The MITRE ATT&CK framework is an industry-standard knowledge base of adversary tactics, techniques, and procedures (TTPs). While the raw data is available in STIX 2.1 format, effectively querying and analyzing this information requires specialized tooling.

The ATT&CK Knowledge Graph addresses this need by providing:

- Natural language querying of attack techniques
- Structured SPARQL queries over an RDF knowledge graph
- Hybrid neuro-symbolic queries combining both approaches
- LLM-powered analysis of security findings
- Automated remediation recommendations

1.1 Design Philosophy

The system follows a **neuro-symbolic** architecture, combining:

Neural Component Vector embeddings enable semantic similarity search, allowing queries like "credential theft from memory" to find relevant techniques without exact keyword matching.

Symbolic Component An RDF knowledge graph provides structured relationships between entities (techniques, groups, software, mitigations), enabling precise graph traversals and SPARQL queries.

This dual approach leverages the strengths of both paradigms: neural networks excel at fuzzy matching and semantic understanding, while symbolic systems excel at precise reasoning over structured relationships.

2 System Architecture

Figure 1 shows the high-level architecture of the system.

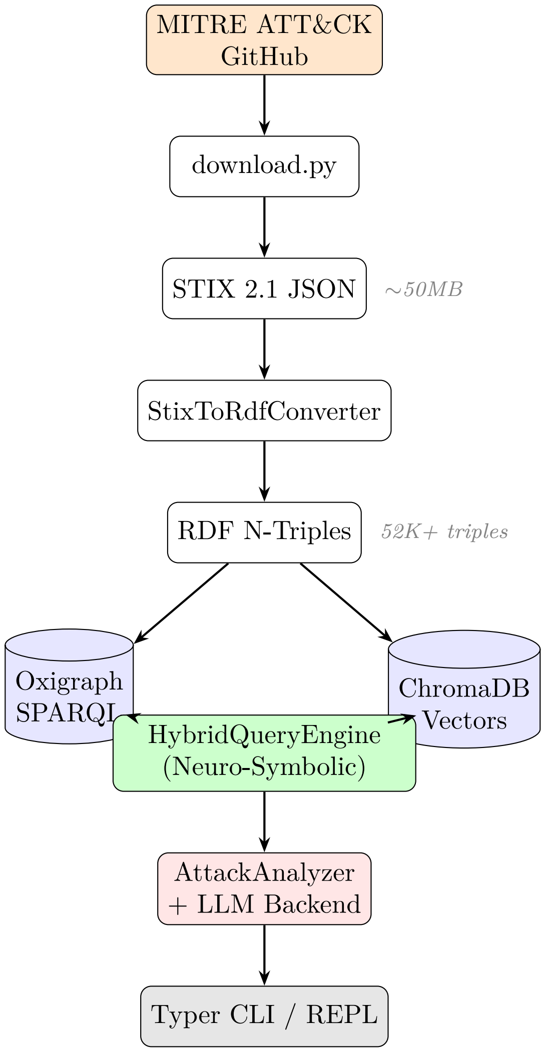

Figure 1: High-level system architecture showing data flow from MITRE ATT&CK through the ingestion pipeline to the query and reasoning layers.

3 Technology Stack

Layer	**Technology**	**Purpose**
Language	Python 3.11+	Core implementation
CLI Framework	Typer	Command-line interface with rich formatting
RDF Store	Oxigraph (pyoxigraph)	SPARQL query engine
Vector Store	ChromaDB	Cosine similarity search
Embeddings	sentence-transformers	nomic-embed-text-v1.5 (8K context)
LLM (Local)	Ollama	Local model inference
LLM (Cloud)	OpenAI	GPT-4 API access
Data Format	STIX 2.1 → RDF	Threat intelligence representation
Package Manager	UV	Fast Python dependency management
Containerization	Docker	Multi-stage build for deployment

Table 1: Technology stack overview.

4 Module Architecture

The codebase is organized into four primary modules, each with distinct responsibilities.

4.1 Ingest Module (`src/ingest/`)

Responsible for converting MITRE STIX data to an RDF knowledge graph.

`download.py` Downloads the STIX bundle from MITRE GitHub with progress indication.

`stix_to_rdf.py` Two-pass converter that first processes entities (techniques, groups, software, mitigations, campaigns, tactics, data sources, data components, detection strategies, analytics) to build a STIX ID → URI mapping, then processes relationships using that mapping.

`embeddings.py` Generates vector embeddings for technique descriptions using sentence-transformers and stores them in ChromaDB.

4.1.1 Two-Pass Conversion Strategy

The STIX-to-RDF conversion uses a two-pass approach:

1. **Pass 1: Entity Processing** — Parse all STIX objects (techniques, groups, software, mitigations, campaigns, tactics, data sources, data components, detection strategies, analytics) and create RDF entities. Build a mapping from STIX IDs to RDF URIs.
2. **Pass 2: Relationship Processing** — Process relationship objects (uses, mitigates, subtechniqueOf, detects, attributedTo, targets), using the ID mapping to resolve source and target references to valid URIs.

This ensures that all entity URIs exist before creating relationships, avoiding dangling references.

4.2 Store Module (`src/store/`)

Provides persistent storage abstractions for both structured and vector data.

`graph.py` `AttackGraph` class wrapping Oxigraph with 32 query methods. Provides:

- RDF file loading (N-Triples, Turtle)
- SPARQL query execution
- Technique queries: `get_technique()`, `get_subtechniques()`, `get_techniques_for_tactic()`
- Group queries: `get_techniques_for_group()`, `get_groups_using_technique()`
- Defense queries: `get_mitigations_for_technique()`, `get_detections_for_technique()`
- Campaign queries: `get_campaign()`, `get_techniques_for_campaign()`
- Data source queries: `get_data_sources()`, `get_techniques_by_data_source()`

`vectors.py` `SemanticSearch` class wrapping ChromaDB. Provides:

- Vector similarity search
- Metadata filtering (tactic, platform)
- Returns `SemanticResult` objects with similarity scores

4.3 Query Module (`src/query/`)

Implements the neuro-symbolic query engine.

`semantic.py` `SemanticSearchEngine` for pure vector-based queries.

`sparql.py` `QueryTemplates` with 20+ SPARQL patterns for common ATT&CK queries.

`hybrid.py` `HybridQueryEngine` — the neuro-symbolic core combining vector search with SPARQL enrichment.

4.3.1 HybridQueryEngine Capabilities

The `HybridQueryEngine` is the central component providing five capability areas:

Core Queries Basic neuro-symbolic search: `query()`, `find_defenses()`, `get_threat_context()`

Campaign Analysis Threat campaign intelligence: `get_campaign_context()`, `find_similar_campaigns()`

Detection Analysis Detection coverage mapping: `get_detection_coverage()`, `find_by_data_source()`

Kill Chain Analysis Attack progression analysis: `analyze_kill_chain()`, `get_attack_surface()`

Entity Search Generic entity operations: `search_entities()`, `get_entity()`, `get_relationships()`

4.3.2 Hybrid Query Flow

The neuro-symbolic pattern combines neural and symbolic approaches:

1. **Vector Search:** Embed the query and find semantically similar techniques.
2. **SPARQL Enrichment:** For each candidate, query the RDF graph for:
   - Associated tactics (kill chain phases)
   - Threat groups and campaigns using the technique
   - Available mitigations and detections
   - Software implementations
   - Parent/child technique relationships
   - Data sources for detection
3. **Return Enriched Results:** `EnrichedTechnique` objects containing both semantic match data and structured graph context.

4.4 Reasoning Module (`src/reasoning/`)

LLM-powered analysis and remediation generation.

`llm.py` Abstract `LLMBackend` with implementations for Ollama (local) and OpenAI (cloud).

`analyzer.py` `AttackAnalyzer` that:

- Classifies security findings against candidate techniques
- Assigns confidence levels (high/medium/low)
- Extracts evidence from narratives
- Generates prioritized remediation recommendations
- Generates LLM-based detection recommendations using graph-stored data sources, detection strategies, and analytics as context

5 Knowledge Graph Schema

5.1 RDF Namespaces

```
PREFIX attack: <https://attack.mitre.org/>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

Listing 1: RDF namespace prefixes

5.2 Entity Types

RDF Type	STIX Type	Description
`attack:Technique`	attack-pattern	Attack technique
`attack:Group`	intrusion-set	Threat actor group
`attack:Malware`	malware	Malicious software
`attack:Tool`	tool	Legitimate tool used maliciously
`attack:Mitigation`	course-of-action	Defensive measure
`attack:Tactic`	x-mitre-tactic	Kill chain phase
`attack:Campaign`	campaign	Threat campaign
`attack:DataSource`	x-mitre-data-source	Detection data source
`attack:DataComponent`	x-mitre-data-component	Detection data component
`attack:DetectionStrategy`	x-mitre-detection-strategy	Detection approach
`attack:Analytic`	x-mitre-analytic	Specific detection rule

Table 2: Mapping between RDF types and STIX types.

5.3 Key Properties

Property	Description
`attack:attackId`	ATT&CK identifier (e.g., T1110.003)
`rdfs:label`	Human-readable name
`attack:description`	Full technique description
`attack:uses` / `attack:usedBy`	Group/software/campaign uses technique (+ inverse)
`attack:mitigates` / `attack:mitigatedBy`	Mitigation addresses technique (+ inverse)
`attack:subtechniqueOf`	Parent technique relationship
`attack:tactic`	Kill chain phase association
`attack:platform`	Target platform (Windows, Linux, etc.)
`attack:detects` / `attack:detectedBy`	Data component detects technique (+ inverse)
`attack:attributedTo` / `attack:hasCampaign`	Campaign attributed to group (+ inverse)
`attack:targets`	Technique targets asset
`attack:hasAnalytic`	Detection strategy has analytic

Table 3: Key RDF properties in the knowledge graph.

5.4 Example SPARQL Query

```
PREFIX attack: <https://attack.mitre.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

```

```
SELECT ?name ?tactic ?mitigation ?mitigationName WHERE {
  <https://attack.mitre.org/technique/T1110.003>
    rdfs:label ?name ;
    attack:tactic ?tactic .

  OPTIONAL {
    ?mitigation attack:mitigates
      <https://attack.mitre.org/technique/T1110.003> ;
      rdfs:label ?mitigationName .
  }
}
```

Listing 2: SPARQL query to find technique details with mitigations

6 Data Flow

6.1 Ingestion Pipeline

1. **Download:** Fetch `enterprise-attack.json` from MITRE GitHub (~50MB STIX bundle).
2. **Parse:** Load JSON and extract STIX objects.
3. **Convert (Pass 1):** Process entities, build ID→URI mapping.
4. **Convert (Pass 2):** Process relationships using the mapping.
5. **Serialize:** Write RDF as N-Triples for fast loading.
6. **Load Graph:** Bulk load into Oxigraph.
7. **Generate Embeddings:** Extract techniques via SPARQL, embed descriptions, store in ChromaDB.

6.2 Query Pipeline

Given a natural language query (e.g., "Found password spraying attack"):

1. **Embed Query:** Generate vector embedding using sentence-transformers.
2. **Vector Search:** Find top-k similar techniques in ChromaDB.
3. **Graph Enrichment:** For each candidate, execute SPARQL queries to fetch:
   - Tactic associations
   - Threat groups and campaigns using the technique
   - Available mitigations
   - Related software
   - Sub-technique relationships
   - Data sources and detection analytics
4. **LLM Classification:** Present enriched candidates to LLM for:
   - Confidence scoring
   - Evidence extraction
   - Technique selection
5. **Remediation & Detection:** Generate prioritized mitigation and detection recommendations.

7 Knowledge Base Statistics

After a full build, the knowledge graph contains:

Entity Type	Count
Techniques (with sub-techniques)	∼835
Threat Groups	∼187
Software (malware + tools)	∼787
Mitigations	∼268
Campaigns	∼52
Tactics	14
Data Sources	∼38
Data Components	∼109
Detection Strategies	∼691
Analytics	∼1,739
RDF Triples (total)	varies

Table 4: Knowledge base statistics (approximate, varies by ATT&CK version).

8 CLI Interface

The system provides a Typer-based CLI with the following commands:

Command	Description
`download`	Fetch STIX bundle from MITRE
`ingest`	Convert STIX to RDF
`build`	Load RDF and build vector index
`stats`	Display knowledge graph statistics
`query`	Execute raw SPARQL queries
`technique`	Get technique details by ID
`group`	Get group's techniques
`search`	Semantic search for techniques
`analyze`	Analyze narrative and suggest remediations
`repl`	Interactive session with history

Table 5: CLI commands.

8.1 Usage Examples

```
# Data pipeline
uv run attack-kg download
uv run attack-kg ingest
uv run attack-kg build

# Lookups
uv run attack-kg technique T1110.003
uv run attack-kg group APT29

# Semantic search
uv run attack-kg search "credential theft from memory"

# Full analysis
uv run attack-kg analyze "Found password spraying attack"
```

```
uv run attack-kg analyze --file finding.txt --backend openai
```

Listing 3: CLI usage examples

9 Design Decisions

Decision	Rationale
Oxigraph over oxrdflib	oxrdflib had hanging issues; pyoxigraph provides direct, performant SPARQL access
N-Triples over Turtle	10–30× faster loading; sacrifices human readability for performance
Two-pass STIX conversion	Ensures all entity URIs exist before linking relationships
nomic-embed-text-v1.5	8K token context handles long technique descriptions
ChromaDB over cloud vectors	Local, persistent, no API keys required
Full URIs in SPARQL	ATT&CK IDs with dots (T1110.003) break prefix notation
Lazy-load LLM backends	REPL can function without LLM initialization overhead

Table 6: Key design decisions and their rationale.

10 Deployment

10.1 Docker

The system uses a multi-stage Docker build:

1. **Builder Stage:** Downloads STIX data, builds knowledge graph, installs dependencies.
2. **Runtime Stage:** Slim image with pre-built stores for fast startup.

10.2 Environment Variables

Variable	Description
`OLLAMA_HOST`	Ollama server URL (default: `http://localhost:11434`)
`OPENAI_API_KEY`	OpenAI API key for cloud LLM

Table 7: Environment variables.

11 Conclusion

The ATT&CK Knowledge Graph demonstrates the power of neuro-symbolic architectures for threat intelligence. By combining vector similarity search with structured RDF graph queries, the system provides flexible, powerful querying capabilities that would be difficult to achieve with either approach alone.

The modular architecture separates concerns cleanly:

- **Ingest** handles data transformation

- **Store** manages persistence
- **Query** implements the hybrid search logic
- **Reasoning** adds LLM intelligence

This separation enables independent evolution of each component while maintaining a cohesive system for security analysts.