ELEUTHERIOS

AEGIS INSIGHT

Multi-Dimensional Knowledge Graph Analytics

with Pattern Recognition and Topology Analysis

Technical Reference Manual

Version 3.0
December 2025

Preface

This technical reference provides comprehensive documentation for Aegis Insight, a knowledge graph analytics platform that enables researchers to explore epistemic topology—the structural relationships between claims, sources, and narrative patterns within large document corpora.

Traditional information retrieval systems focus on finding relevant documents. Aegis Insight takes a fundamentally different approach: it extracts structured knowledge from unstructured text, builds a queryable graph of claims and entities, and provides analytical tools to examine the shape and structure of knowledge itself.

The platform combines several cutting-edge technologies:

- Local large language model processing via Ollama for privacy-preserving extraction
- Graph database storage in Neo4j for relationship-rich queries
- Vector embeddings in PostgreSQL with pgvector for semantic similarity
- Real-time D3.js visualization for interactive exploration
- Multi-dimensional analysis across temporal, geographic, emotional, and citation axes

This manual is intended for researchers, analysts, and developers who want to understand both the theoretical foundations and practical operation of the system. It covers architecture, installation, operation, customization, and the algorithms underlying each analytical capability.

Who Should Read This Manual

- **Research Analysts:** Researchers exploring document corpora for patterns and relationships
- **System Architects:** Teams building knowledge management or analytical systems
- **Developers:** Engineers extending or customizing the platform
- **IT Administrators:** Professionals deploying the system in their organizations

Document Conventions

Throughout this manual:

- Command-line examples are shown in monospace font
- Configuration values use <angle brackets> for placeholders
- Important notes are highlighted in bold
- API endpoints are documented with HTTP method and path

Table of Contents

1. Introduction to Aegis Insight

1.1 The Challenge of Knowledge Analysis

Modern research increasingly depends on synthesizing insights from large, heterogeneous document collections. Whether analyzing historical archives, scientific literature, news corpora, or organizational knowledge bases, researchers face a common challenge: documents contain rich, interconnected knowledge, but that knowledge is locked in unstructured text.

Traditional approaches to this challenge fall into two categories:

Keyword Search Systems

Full-text search engines like Elasticsearch excel at finding documents containing specific terms. However, they treat documents as bags of words, ignoring the semantic structure within. A search for 'climate policy' returns documents mentioning those words, but cannot answer questions like 'Which claims about climate policy are contested?' or 'How do citation patterns differ between mainstream and alternative perspectives?'

Manual Analysis

Human researchers can perform deep structural analysis, but this approach doesn't scale. A scholar might spend years analyzing a few hundred documents, carefully tracking claims, sources, and relationships. This produces rich insight but cannot be applied to corpora of thousands or millions of documents.

Aegis Insight bridges this gap by automating the extraction of structured knowledge while preserving the relational richness that makes human analysis valuable.

1.2 The Aegis Insight Approach

Aegis Insight implements a four-stage pipeline that transforms unstructured documents into a queryable knowledge graph:

Stage 1: Document Ingestion

Documents in various formats (PDF, TXT, DOCX, Markdown) are normalized and chunked into processable segments. The system preserves document metadata and maintains provenance throughout the pipeline.

Stage 2: Multi-Dimensional Extraction

Each chunk passes through seven specialized extractors, each powered by local large language models. These extractors identify:

- Named entities (people, organizations, places, concepts, events)
- Factual claims with confidence scoring and claim type classification
- Temporal references (dates, periods, sequences, relative time)
- Geographic references (locations, regions, spatial relationships)
- Citations and source attributions
- Emotional content and rhetorical tone
- Authority indicators and domain expertise markers

Stage 3: Graph Construction

Extracted knowledge is loaded into a Neo4j graph database as interconnected nodes and relationships. Claims link to the entities they mention, documents link to the claims they contain, and alias relationships connect variant names for the same entity.

Stage 4: Semantic Indexing

Claims are converted to vector embeddings using local language models and stored in PostgreSQL with the pgvector extension. This enables semantic similarity search—finding conceptually related claims even without keyword overlap.

1.3 What Makes Aegis Insight Unique

Several architectural decisions distinguish Aegis Insight from similar systems:

Local-First Processing

All language model inference runs locally via Ollama. No document content is sent to external APIs. This enables deployment in air-gapped environments and ensures complete data privacy. The mistral-nemo:12b model provides excellent extraction quality while running efficiently on consumer GPUs.

Claim-Centric Data Model

Rather than treating documents as atomic units, Aegis Insight decomposes them into individual claims. This enables analysis at the assertion level: tracking how specific claims propagate across sources, identifying contested claims, and measuring citation patterns for individual assertions.

Multi-Dimensional Metadata

Each claim carries rich metadata across multiple dimensions. Temporal data enables chronological analysis. Geographic data enables spatial analysis. Emotional data enables sentiment analysis. This multi-dimensional richness supports analytical questions that single-dimension systems cannot address.

Pattern Detection Algorithms

Beyond search and retrieval, Aegis Insight includes pattern detection algorithms that identify structural anomalies in the knowledge graph. These algorithms detect unusual clustering patterns, narrative synchronization, and citation network topology that may warrant further investigation.

Interactive Visualization

The web interface provides real-time, interactive visualization of knowledge graph structure. Users can explore relationships, filter by claim type, examine temporal distributions, and visually identify patterns that might be invisible in tabular data.

1.4 Use Cases and Applications

Aegis Insight serves diverse analytical needs:

Historical Research

Analyze historical document collections to trace the evolution of ideas, identify influential sources, and map relationships between historical actors. The temporal extraction capability enables chronological analysis across large corpora.

Literature Review

Synthesize scientific or academic literature by extracting key claims, mapping citation relationships, and identifying areas of consensus or disagreement. The claim-centric model naturally supports systematic review methodologies.

Investigative Analysis

Explore document collections for patterns and relationships that warrant further investigation. The pattern detection algorithms can identify unusual structural features that human analysts might miss in large corpora.

Knowledge Management

Build organizational knowledge bases that preserve not just documents but the structured knowledge within them. The graph model enables rich querying across the knowledge base.

Media Analysis

Analyze news and media corpora to understand narrative patterns, source relationships, and coverage evolution over time. The coordination detection algorithm can identify synchronized messaging patterns.

1.5 Beyond Traditional RAG: Multi-Perspective Knowledge

Retrieval-Augmented Generation (RAG) has become the standard approach for grounding large language models in external knowledge. However, traditional RAG systems have significant limitations that Aegis Insight addresses.

Limitations of Standard RAG

Traditional RAG pipelines typically:

- Return the 'top K' most relevant chunks without perspective diversity
- Lose claim-level granularity by retrieving entire document chunks
- Cannot distinguish between primary assertions and meta-commentary
- Lack temporal, geographic, and emotional context
- Cannot identify when retrieved content represents contested knowledge
- Provide no visibility into source relationships or citation patterns

These limitations can cause AI systems to present single-perspective answers as authoritative, miss important nuances, or unknowingly amplify contested claims without appropriate caveats.

The Aegis Insight Approach

Aegis Insight provides structured knowledge that enables more sophisticated AI grounding:

- Claim-level retrieval returns specific assertions, not document chunks
- Claim type classification distinguishes facts from meta-commentary
- Multiple perspectives can be retrieved and presented together
- Detection scores flag when topics may be contested or coordinated
- Temporal data enables chronological context
- Citation data shows source relationships and authority

Model Context Protocol (MCP) Integration

Aegis Insight exposes its knowledge graph through the Model Context Protocol (MCP), enabling integration with AI assistants and LLM applications. Through the MCP endpoint, AI systems can:

- Query for claims related to a topic with multi-perspective results
- Retrieve detection scores indicating contested or coordinated topics
- Access claim metadata (temporal, geographic, emotional, citation)
- Traverse entity relationships for context expansion
- Filter by claim type to separate facts from meta-claims

This enables AI applications to provide more nuanced, multi-perspective responses with appropriate epistemic humility when addressing contested topics.

2. System Architecture

2.1 Architecture Overview

Aegis Insight follows a microservices-inspired architecture with four primary components communicating via well-defined interfaces:

Component	Technology	Role	Port
Knowledge Graph	Neo4j 5.x	Stores claims, entities, relationships	7474, 7687
Vector Store	PostgreSQL 16 + pgvector	Stores embeddings for similarity search	5432
API Server	Python FastAPI	REST API, business logic, orchestration	8001
LLM Engine	Ollama	Local inference for extraction/analysis	11434
Web Interface	JavaScript/D3.js	Visualization and user interaction	(via API)

The architecture separates concerns cleanly: Neo4j handles graph traversal and relationship queries, PostgreSQL handles vector similarity search, Ollama handles language model inference, and FastAPI orchestrates workflows and exposes the unified API.

2.2 Neo4j Knowledge Graph

Neo4j serves as the primary knowledge store, holding the graph of claims, entities, documents, and their relationships. The property graph model naturally represents the multi-dimensional structure of extracted knowledge.

2.2.1 Node Types

The graph contains five primary node types:

Claim Nodes

Claims are the atomic units of knowledge in the system. Each claim represents a single factual assertion extracted from source text.

Property	Type	Description
claim_id	String	Unique identifier (UUID-based)
claim_text	String	The assertion text
claim_type	String	PRIMARY, SECONDARY, CONTEXTUAL, or META
confidence	Float	Extraction confidence (0.0-1.0)
source_file	String	Origin document path
temporal_data	JSON	Extracted temporal information
geographic_data	JSON	Extracted geographic information
emotional_data	JSON	Emotional analysis results
citation_data	JSON	Citation and reference information
authority_data	JSON	Authority/expertise indicators
excluded	Boolean	Soft-delete flag

Entity Nodes

Entities represent named things mentioned in claims: people, organizations, places, concepts, and events.

Property	Type	Description
name	String	Canonical entity name
type	String	PERSON, ORG, PLACE, CONCEPT, EVENT
aliases	List	Alternative names/spellings

Document Nodes

Documents track source provenance and aggregate metadata.

Property	Type	Description
doc_id	String	Unique document identifier
source_file	String	File path or URL
title	String	Document title if available
doc_date	String	Publication/creation date

Chunk Nodes

Chunks represent segments of documents, preserving the document structure.

Property	Type	Description
chunk_id	String	Unique chunk identifier
text	String	Original chunk text
chunk_index	Integer	Position within document

Topic Nodes

Topics provide subject matter categorization for documents.

Property	Type	Description
name	String	Topic name (lowercase)
category	String	primary or secondary

2.2.2 Relationship Types

Relationships connect nodes and carry semantic meaning:

Relationship	From	To	Meaning
CONTAINS	Document	Chunk	Document contains this text chunk
CONTAINS_CLAIM	Chunk	Claim	Chunk contains this claim
MENTIONS	Claim	Entity	Claim references this entity
MENTIONS	Chunk	Entity	Chunk references this entity
SAME_AS	Entity	Entity	Entities are aliases for same thing
ABOUT	Document	Topic	Document covers this topic
CITES	Claim	Claim	Claim cites another claim
SUPPORTS	Claim	Claim	Claim supports another
CONTRADICTS	Claim	Claim	Claim contradicts another

2.2.3 Graph Query Patterns

The graph structure enables powerful query patterns. Here are common Cypher queries used throughout the system:

Topic Traversal Query

Find all claims related to a topic by traversing from Topic through Documents to Claims:

```
MATCH (t:Topic)
WHERE toLower(t.name) CONTAINS toLower($topic)
MATCH (t)<-[:ABOUT]-(d:Document)
MATCH (d)-[:CONTAINS]->(ch:Chunk)-[:CONTAINS_CLAIM]->(c:Claim)
RETURN c.claim_id, c.claim_text, c.claim_type
```

Entity Co-occurrence Query

Find claims mentioning two entities together:

```
MATCH (c:Claim)-[:MENTIONS]->(e1:Entity {name: $entity1})
MATCH (c)-[:MENTIONS]->(e2:Entity {name: $entity2})
RETURN c
```

Alias Resolution Query

Find all variants of an entity name:

```
MATCH (e:Entity)-[:SAME_AS*0..2]-(related:Entity)
WHERE e.name = $name
RETURN DISTINCT related.name
```

2.3 PostgreSQL Vector Store

PostgreSQL with the pgvector extension provides semantic similarity search capabilities. While Neo4j excels at graph traversal, pgvector excels at finding semantically similar content regardless of keyword overlap.

2.3.1 Embeddings Table Schema

```
CREATE TABLE claim_embeddings (
    id SERIAL PRIMARY KEY,
    claim_id VARCHAR(255) UNIQUE NOT NULL,
    claim_text TEXT,
    embedding vector(1024),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX ON claim_embeddings
USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);
```

2.3.2 Similarity Search

The system uses cosine similarity for semantic matching:

```
SELECT claim_id, claim_text,
       1 - (embedding <=> $query_embedding) AS similarity
FROM claim_embeddings
ORDER BY embedding <=> $query_embedding
```

```
LIMIT 100;
```

The IVFFlat index enables approximate nearest neighbor search, providing sub-second query times even with millions of embeddings.

2.4 Ollama LLM Engine

Ollama provides local large language model inference, enabling privacy-preserving extraction and analysis. The system uses Ollama for:

- Claim extraction from document chunks
- Entity recognition and classification
- Temporal and geographic parsing
- Emotional content analysis
- Embedding generation for similarity search
- LLM-powered query synthesis

2.4.1 Model Selection

The default model is mistral-nemo:12b, chosen for its balance of quality and efficiency:

Factor	mistral-nemo:12b	Notes
Parameters	12 billion	Good quality/speed tradeoff
Context Window	128K tokens	Handles long documents
VRAM Required	~8GB	Fits consumer GPUs
Speed (GPU)	~50 tokens/sec	Practical for batch processing
Extraction Quality	Excellent	Strong instruction following

Alternative models can be configured for specific use cases:

- llama3.1:70b - Higher quality for critical extraction, requires 48GB+ VRAM
- mistral:7b - Faster processing, slightly lower quality
- nomic-embed-text - Specialized embedding model

2.5 FastAPI Application Server

The Python FastAPI application provides the REST API, orchestrates workflows, and implements business logic. Key architectural features:

- Async request handling for concurrent operations
- Background task processing for long-running jobs
- Modular router architecture for API organization
- Connection pooling for database efficiency
- Static file serving for the web interface

2.5.1 API Router Organization

Router	Prefix	Responsibility
Main	/api	Core search and graph operations
Detection	/api/detect	Pattern detection algorithms
Admin	/api/admin	System administration
Data Management	/api/data-management	CRUD operations
Calibration	/api	Detection calibration

3. Multi-Dimensional Extraction Pipeline

3.1 Pipeline Overview

The extraction pipeline transforms unstructured documents into structured knowledge through a series of specialized extractors. Each extractor focuses on a single dimension of knowledge, and their combined output provides a rich, multi-dimensional representation of document content.

The pipeline processes documents in four phases:

1. Document Conversion - Transform source formats to JSONL chunks
2. Multi-Extractor Processing - Run all seven extractors on each chunk
3. Graph Building - Load extracted data into Neo4j
4. Embedding Generation - Create vector embeddings for similarity search

3.2 Document Conversion

The conversion phase normalizes diverse document formats into a consistent JSONL (JSON Lines) format for processing.

3.2.1 Supported Input Formats

Aegis Insight supports diverse document formats, including scanned historical documents:

Format	Extension	Conversion Method	Notes
PDF (text)	.pdf	PyMuPDF (fitz)	Native text extraction
PDF (scanned)	.pdf	Tesseract OCR	Image-based PDFs, declassified docs
Plain Text	.txt	Direct read	UTF-8 encoding assumed
Markdown	.md	Direct read	Formatting stripped
Word	.docx	python-docx	Text and basic structure
JSONL	.jsonl	Pass-through	Pre-chunked format

OCR for Historical Documents

The Tesseract OCR integration enables ingestion of scanned documents, historical archives, and declassified materials. When a PDF contains image-based pages rather than selectable text, the system automatically invokes Tesseract to extract text content.

This capability is particularly valuable for:

- Declassified government documents (often released as scanned images)
- Historical newspaper archives
- Academic archives and special collections
- Legal document discovery sets

OCR quality depends on scan quality. For best results, use 300 DPI or higher resolution scans with good contrast.

3.2.2 Chunking Strategy

Documents are split into chunks for processing. The chunking algorithm balances several concerns:

- Chunk size: 500-1000 words optimal for LLM context
- Semantic boundaries: Prefer paragraph/section breaks
- Overlap: 50-word overlap prevents claim fragmentation
- Metadata preservation: Each chunk retains document provenance

The JSONL output format:

```
{
  "chunk_id": "doc123_chunk_0",
  "source_file": "/path/to/document.pdf",
  "chunk_index": 0,
  "chunk_text": "The extracted text content...",
  "doc_title": "Document Title",
  "doc_date": "2024-01-15"
}
```

3.3 The Seven Extractors

Each chunk passes through seven specialized extractors. The extractors run in sequence, and their outputs are merged into a comprehensive extraction record.

3.3.1 Entity Extractor

File: aegis_entity_extractor.py

The entity extractor identifies named entities mentioned in text and classifies them by type.

Entity Types

Type	Description	Examples
PERSON	Individual humans	Thomas Paine, Marie Curie
ORG	Organizations, companies, institutions	Congress, FDA, Reuters
PLACE	Geographic locations	Philadelphia, Lake Titicaca
CONCEPT	Abstract ideas, theories	Democracy, Relativity
EVENT	Historical events, incidents	Boston Tea Party, WWI

Extraction Prompt

The extractor uses a structured prompt that requests JSON output:

```
Extract all named entities from this text.
For each entity, provide:
- name: The entity name as it appears
- type: PERSON, ORG, PLACE, CONCEPT, or EVENT
- aliases: Any alternative names mentioned
Return as JSON array.
```

Alias Handling

The extractor identifies when the same entity is referenced by different names (e.g., 'General Butler', 'Smedley Butler', 'Gen. Butler'). These aliases are preserved and later used to create SAME_AS relationships in the graph.

3.3.2 Claim Extractor

File: aegis_claim_extractor.py

The claim extractor identifies factual assertions and classifies them by type and confidence.

Claim Types

Type	Description	Example
PRIMARY	Direct factual assertions	The ship exploded on February 15, 1898
SECONDARY	Supporting or contextual facts	Yellow journalism had a circulation of 800,000
CONTEXTUAL	Background information	Spain controlled Cuba at the time
META	Claims about other claims	Historians now dispute the original account

Confidence Scoring

Each claim receives a confidence score (0.0-1.0) based on:

- Hedging language ('allegedly', 'reportedly') reduces confidence
- Attribution to sources increases confidence
- Specificity (dates, numbers) increases confidence
- Subjective language reduces confidence

META Claim Significance

META claims are particularly significant for pattern analysis. A high ratio of META claims to PRIMARY claims often indicates contested or actively debated topics. The detection algorithms use META claim density as a signal.

3.3.3 Temporal Extractor

File: aegis_temporal_extractor.py

The temporal extractor identifies time references and normalizes them to machine-readable formats.

Temporal Data Structure

```
{
  "absolute_dates": ["1898-02-15", "1941-12-07"],
  "relative_dates": ["two weeks later", "the following year"],
  "periods": ["the 1890s", "post-war era"],
  "sequences": ["before the investigation", "after the explosion"]
}
```

Date Normalization

Various date formats are normalized to ISO 8601:

- 'February 15, 1898' → '1898-02-15'
- '15/2/1898' → '1898-02-15'
- 'the 1890s' → '1890-01-01' to '1899-12-31' (range)

3.3.4 Geographic Extractor

File: aegis_geographic_extractor.py

The geographic extractor identifies location references and attempts to geocode them.

Geographic Data Structure

```
{
  "locations": [
    {"name": "Havana Harbor", "type": "harbor",
     "coordinates": {"lat": 23.1136, "lon": -82.3666}},
    {"name": "Cuba", "type": "country",
     "coordinates": {"lat": 21.5218, "lon": -77.7812}}
  ],
  "spatial_relations": ["near the harbor", "off the coast"]
}
```

Geocoding

The extractor uses the geopy library to resolve location names to coordinates. Ambiguous locations are flagged for manual review.

3.3.5 Citation Extractor

File: aegis_citation_extractor.py

The citation extractor identifies references to other sources, documents, or claims.

Citation Data Structure

```
{
  "citations": [
    {"source": "Naval Court of Inquiry",
     "type": "official_document",
     "year": 1898},
    {"source": "Admiral Rickover study",
     "type": "academic",
     "year": 1976}
  ],
  "attribution_phrases": ["according to", "as reported by"]
}
```

Citation Types

Type	Description
academic	Peer-reviewed papers, studies
official_document	Government reports, court records
news	Newspaper or media citations
primary_source	Original documents, testimony
secondary	Books, encyclopedias, summaries

3.3.6 Emotion Extractor

File: aegis_emotion_extractor.py

The emotion extractor analyzes the emotional content and rhetorical tone of text.

Emotional Data Structure

```
{
  "primary_emotion": "anger",
  "emotion_scores": {
    "anger": 0.7,
    "fear": 0.3,
    "sadness": 0.1,
    "joy": 0.0,
    "surprise": 0.2
  },
  "urgency": 0.8,
  "sentiment": -0.6
}
```

Emotion Analysis Applications

Emotional data enables several analytical capabilities:

- Sentiment tracking across documents or time periods
- Identifying emotionally charged claims
- Detecting synchronized emotional patterns (coordination signal)
- Comparing emotional tone between sources

3.3.7 Authority Domain Analyzer

File: aegis_authority_domain_analyzer.py

The authority analyzer assesses domain expertise indicators and credibility markers.

Authority Data Structure

```
{
  "domains": ["military history", "naval warfare"],
  "expertise_indicators": ["Admiral", "historian", "researcher"],
  "credibility_markers": ["peer-reviewed", "official inquiry"],
  "authority_score": 0.75
}
```

Domain Classification

The analyzer classifies content into knowledge domains:

Domain	Indicator Terms
medicine	clinical, treatment, diagnosis, physician
law	court, statute, ruling, attorney

Domain	Indicator Terms
military	general, admiral, operation, combat
science	study, research, experiment, peer-reviewed
history	historian, archive, period, century

3.4 Orchestration

The extraction orchestrator coordinates extractor execution and manages the processing pipeline.

3.4.1 Extraction Orchestrator

The V3 orchestrator (aegis_extraction_orchestrator_v3.py) is the standard for both web UI and CLI imports. It provides:

- All seven domain extractors running in sequence
- Coreference resolution for entity consistency
- Enhanced metadata preservation
- Checkpoint/resume capability for long jobs
- Parallel processing support for multi-GPU systems

The orchestrator automatically detects available GPU resources and configures batch sizes accordingly.

3.4.2 Processing Flow

1. Load JSONL chunk from input file
2. Run entity extractor → entities[]
3. Run claim extractor → claims[]
4. Run temporal extractor → temporal_data
5. Run geographic extractor → geographic_data
6. Run citation extractor → citation_data
7. Run emotion extractor → emotional_data
8. Run authority analyzer → authority_data
9. Merge all outputs into extraction record
10. Write to checkpoint file

3.4.3 Checkpoint and Resume

The orchestrator supports checkpoint/resume for long-running jobs:

- Progress saved after each chunk
- Automatic resume from last checkpoint on restart
- Checkpoint files stored in ~/.aegis/jobs/<job_id>/checkpoints/

3.5 Graph Building

The graph builder (aegis_graph_builder.py) loads extracted data into Neo4j.

3.5.1 Node Creation

For each extraction record, the builder creates:

- One Document node (if not exists)
- One Chunk node
- Multiple Claim nodes (one per extracted claim)
- Multiple Entity nodes (deduplicated)
- Topic nodes based on document subject matter

3.5.2 Relationship Creation

Relationships are created based on extraction results:

- Document -[:CONTAINS]-> Chunk
- Chunk -[:CONTAINS_CLAIM]-> Claim
- Claim -[:MENTIONS]-> Entity (for each entity in claim)
- Document -[:ABOUT]-> Topic

3.5.3 Entity Deduplication

Entity names are normalized and deduplicated:

- Case normalization (lowercase comparison)
- Whitespace normalization
- Alias matching via SAME_AS relationships
- Fuzzy matching for near-duplicates

3.6 Embedding Generation

The embedding generator (generate_embeddings.py) creates vector representations for semantic search.

3.6.1 Embedding Process

1. Query Neo4j for claims without embeddings
2. Batch claims (100 per batch for efficiency)
3. Send to Ollama embedding endpoint
4. Store embeddings in PostgreSQL
5. Update progress tracking

3.6.2 Embedding Model

The system uses the nomic-embed-text model through Ollama:

- Dimension: 1024
- Context: 8192 tokens
- Speed: ~100 embeddings/second on GPU

4. Pattern Detection Algorithms

4.1 Detection Philosophy

The pattern detection system identifies structural features in the knowledge graph that may warrant investigation. It does not make truth claims—it identifies patterns that deviate from baseline expectations.

Each detector analyzes multiple signals and produces a composite score. The score represents pattern strength, not certainty. High scores indicate patterns worth examining; they do not prove any particular conclusion.

4.2 Suppression Pattern Detection

File: aegis_suppression_detector_v2.py

The suppression detector identifies patterns suggesting that information about a topic may have been systematically diminished in visibility or credibility.

4.2.1 Signal Analysis

The detector analyzes five signal categories:

Suppression Narrative (40% weight)

Detects language patterns associated with dismissal or marginalization:

Category	Pattern Examples
Dismissal Language	debunked, conspiracy, fringe, pseudoscience, discredited
Institutional Actions	banned, censored, suppressed, seized, revoked, fired
Suppression Experience	silenced, marginalized, persecuted, blacklisted
Memory-Holing	forgotten, erased, memory-holed, expunged, buried

Evidence Avoidance (20% weight)

Measures META claims that dismiss PRIMARY claims without engaging with evidence:

- META claims without citations
- Dismissal without substantive rebuttal
- Appeal to authority without evidence

Meta Claim Density (15% weight)

Calculates the ratio of META claims to PRIMARY claims:

```
meta_density = count(META claims) / count(PRIMARY claims)
```

High density suggests contested or actively disputed topics.

Network Isolation (15% weight)

Analyzes citation network connectivity:

- Citation ratio compared to similar topics
- Cross-citation patterns between sources

- Isolation from mainstream citation networks

Authority Mismatch (10% weight)

Detects patterns where expertise is dismissed:

- Credentialed sources being dismissed
- Non-expert dismissal of expert claims
- Appeal to institutional authority over evidence

4.2.2 Goldfinger Scoring

The detector uses 'Goldfinger scoring'—named after the James Bond principle: 'Once is happenstance, twice is coincidence, three times is enemy action.'

Single indicators produce low scores. Multiple independent indicators compound:

Indicators	Base Score	Interpretation
1	0.10	Single indicator, likely noise
2	0.30	Two indicators, possible pattern
3	0.50	Three indicators, notable pattern
4	0.60	Four indicators, strong pattern
5+	0.70+	Multiple indicators, significant pattern

4.2.3 LLM Directionality Filtering

Raw pattern matches are filtered through an LLM to assess directionality:

A claim containing 'censored' might describe:

- The topic being censored (positive indicator)
- The topic supporting censorship of something else (negative indicator)
- General discussion of censorship (neutral)

The LLM classifies each match, and only positive indicators contribute to the score.

4.2.4 Score Interpretation

Score	Level	Interpretation
0.00-0.25	LOW	Minimal pattern indicators
0.25-0.50	MODERATE	Some indicators present
0.50-0.75	HIGH	Strong pattern indicators
0.75-1.00	CRITICAL	Multiple strong indicators

4.3 Coordination Pattern Detection

File: aegis_coordination_detector.py

The coordination detector identifies patterns suggesting artificially synchronized narratives.

4.3.1 Signal Analysis

Temporal Clustering

Measures claim concentration in narrow time windows:

```
clustering_score = claims_in_window / expected_claims
```

High clustering suggests coordinated release timing.

Language Similarity

Uses embedding cosine similarity to detect unusual phrasing overlap:

```
similarity = cosine(embedding_a, embedding_b)
```

Scores claims by average similarity to peers. High similarity across ostensibly independent sources is a coordination signal.

Source Centralization

Measures source distribution using Gini coefficient:

- Gini = 0: Claims evenly distributed across sources
- Gini = 1: All claims from single source

High centralization indicates few sources dominating the narrative.

Emotion Synchronization

Compares emotional signatures across claims:

- Extract emotion vectors from emotional_data
- Calculate cross-claim emotion correlation
- High correlation suggests coordinated messaging

Citation Cartel Detection

Analyzes citation patterns for circular or coordinated citing:

- Sources citing each other disproportionately
- Coordinated citation of specific claims
- Unusual citation timing patterns

4.3.2 Cluster Visualization

The coordination detector outputs cluster assignments that the UI uses for visualization:

- Detected nodes highlighted with orange border
- 'Cluster Detected Nodes' groups them visually
- Timeline view shows temporal distribution

4.4 Anomaly Detection

File: aegis_anomaly_detector.py

The anomaly detector identifies statistical outliers in claim patterns.

Note: This detector is marked as experimental. Results may be unreliable.

4.4.1 Anomaly Types

- Distribution anomalies: Claim patterns deviating from expected distributions
- Relationship anomalies: Unusual entity co-occurrence patterns
- Temporal anomalies: Unexpected gaps or bursts in claim timing
- Citation anomalies: Unusual citation network topology

4.5 Calibration Profiles

Detection algorithms can be tuned using calibration profiles stored as JSON files.

4.5.1 Profile Location

Profiles are stored in: data/calibration_profiles/

Profile	Use Case
default.json	General-purpose detection
state_suppression.json	Government/institutional patterns
academic_gatekeeping.json	Scientific establishment patterns
media_coordination.json	News/media patterns

4.5.2 Profile Structure

```
{
  "name": "custom_profile",
  "description": "Custom detection profile",
  "suppression": {
    "narrative_weight": 0.40,
    "evidence_weight": 0.20,
    "meta_density_weight": 0.15,
    "isolation_weight": 0.15,
    "authority_weight": 0.10,
    "custom_patterns": ["pattern1", "pattern2"]
  },
  "coordination": {
    "temporal_window_hours": 24,
    "similarity_threshold": 0.85
  }
}
```

4.5.3 Creating Custom Profiles

To create a custom detection profile:

6. Copy an existing profile as a template
7. Adjust weights for your use case
8. Add domain-specific patterns if needed
9. Save with a descriptive filename
10. Select the profile in the UI calibration panel

5. Web Interface Guide

5.1 Interface Overview

The web interface provides interactive visualization and control of all system capabilities. It is organized into tabs for different functional areas.

5.1.1 Main Tabs

Tab	Function
Knowledge Graph	Primary visualization and search interface
Data Loading	Import documents into the system
Admin	System status, configuration, maintenance

5.2 Knowledge Graph Tab

The Knowledge Graph tab is the primary interface for exploring and analyzing content.

5.2.1 Search Panel

The search panel on the left side provides:

Search Input

Enter search terms to query the knowledge graph. The system uses semantic similarity, so conceptually related content will be found even without exact keyword matches.

Mode Selector

Choose the analysis mode:

Mode	Description
Standard	Basic search and visualization
Detect Suppression	Analyze for suppression patterns
Detect Coordination	Analyze for coordination patterns
Detect Anomalies	Analyze for statistical anomalies (experimental)

Results Panel

Detection results display:

- Overall score (0.00-1.00)
- Score level (LOW/MODERATE/HIGH/CRITICAL)
- Number of claims analyzed
- Individual signal breakdowns
- Affected claims list

5.2.2 Graph Visualization

The central area displays an interactive force-directed graph.

Node Types and Colors

Node Type	Color	Shape
Claim	Gold/Orange	Circle
Entity	Blue	Circle

Node Type	Color	Shape
Detected (highlighted)	Orange border, pulsing	Circle

Interactions

Action	Effect
Click node	Select and show details in side panel
Drag node	Reposition node (temporarily pins it)
Scroll/pinch	Zoom in/out
Click background	Deselect current node
Double-click node	Expand connections

5.2.3 Filter Controls

The toolbar above the graph provides filtering options:

Node Limit

Slider controlling maximum displayed nodes (10-500). Lower values improve performance; higher values show more context.

Cluster Detected Nodes

When enabled, detection-relevant nodes are grouped on the left side of the visualization for easier analysis.

Narrative Clustering

Groups nodes by stance (pro/anti/neutral) on the detected topic.

Timeline View

Arranges nodes chronologically based on temporal data. Useful for understanding how narratives evolved over time.

5.2.4 Side Panel

Clicking a node displays its details in the right side panel:

For Claims:

- Full claim text
- Claim type (PRIMARY/SECONDARY/CONTEXTUAL/META)
- Confidence score
- Source document
- Temporal data (if extracted)
- Geographic data (if extracted)
- Emotional analysis
- Connected entities

For Entities:

- Entity name and type
- Known aliases
- Connected claims

- Co-occurring entities

5.3 Data Loading Tab

The Data Loading tab provides a wizard interface for importing documents.

5.3.1 Import Wizard Steps

Step 1: Select Files

Choose files to import:

- Drag and drop files into the upload area
- Click to browse and select files
- Select from inbox folder (pre-staged files)

Step 2: Configure Options

Review import settings:

- Chunk size (default: 500 words)
- Overlap size (default: 50 words)
- Processing priority

Step 3: Monitor Progress

Watch the three-phase progress:

- Phase 1: Converting documents to chunks
- Phase 2: Extracting knowledge (longest phase)
- Phase 3: Generating embeddings

Step 4: Review Results

After completion, view statistics:

- Documents processed
- Entities extracted
- Claims extracted
- Processing duration

5.3.2 Processing Time Estimates

Document Size	CPU Only	With GPU
10 pages	5-10 minutes	2-3 minutes
100 pages	30-60 minutes	10-15 minutes
500 pages	3-5 hours	45-90 minutes

5.4 Admin Tab

The Admin tab provides system management capabilities.

5.4.1 System Status

Displays health of all system components:

Component	Status Indicators
Neo4j	Connection status, node/relationship counts
PostgreSQL	Connection status, embedding count
Ollama	Model availability, GPU status
API Server	Uptime, request metrics

5.4.2 Detection Infrastructure

Manage detection system components:

- Initialize/rebuild detection indexes
- Update materialized views
- Refresh calibration profiles

5.4.3 Data Management

Manage imported content:

View Sources

List all imported document sources with claim counts.

Exclude Claims

Soft-delete claims by marking them as excluded. Excluded claims are hidden from search and detection but can be restored.

Remove Sources

Permanently remove a document source and all its claims.

6. Command Line Interface

6.1 CLI Overview

Aegis Insight provides command-line tools for batch processing and automation.

6.2 Import Wizard CLI

File: aegis_import_wizard.py

Interactive command-line wizard for document import.

6.2.1 Starting the Wizard

```
python aegis_import_wizard.py
```

6.2.2 Wizard Flow

The wizard guides you through:

11. Select input directory or files
12. Configure processing options
13. Review and confirm settings
14. Monitor extraction progress
15. View completion statistics

6.2.3 Non-Interactive Mode

For scripted imports:

```
python aegis_import_wizard.py --input /path/to/docs --auto
```

6.3 Extraction Pipeline

File: run_extraction_pipeline_v3.py

Direct pipeline execution for advanced users.

6.3.1 Basic Usage

```
python run_extraction_pipeline_v3.py \
  --input /path/to/chunks.jsonl \
  --output /path/to/extracted.jsonl
```

6.3.2 Options

Option	Description	Default
--input	Input JSONL file	Required
--output	Output JSONL file	Required
--model	Ollama model name	mistral-nemo:12b
--batch-size	Chunks per batch	10
--checkpoint	Enable checkpointing	True

6.4 Graph Builder

File: aegis_graph_builder.py

Load extracted data into Neo4j.

6.4.1 Basic Usage

```
python aegis_graph_builder.py \
  --input /path/to/extracted.jsonl
```

6.5 Embedding Generator

File: generate_embeddings.py

Generate vector embeddings for claims.

6.5.1 Basic Usage

```
python generate_embeddings.py
```

Processes all claims in Neo4j that don't have embeddings.

6.5.2 Options

Option	Description	Default
--batch-size	Claims per batch	100
--model	Embedding model	nomic-embed-text
--limit	Maximum claims to process	None (all)

6.6 Detection CLI

Run detectors from command line for batch analysis.

6.6.1 Suppression Detection

```
curl -X POST http://localhost:8001/api/detect/suppression \
  -H 'Content-Type: application/json' \
  -d '{"topic": "Thomas Paine"}'
```

6.6.2 Coordination Detection

```
curl -X POST http://localhost:8001/api/detect/coordination \
  -H 'Content-Type: application/json' \
  -d '{"topic": "Remember the Maine"}'
```

7. API Reference

7.1 API Overview

The REST API runs on port 8001 by default and provides programmatic access to all system capabilities.

7.2 Search Endpoints

7.2.1 Pattern Search

POST /api/pattern-search

Semantic search with LLM synthesis.

Request Body:

```
{
  "query": "search terms",
  "limit": 100,
  "include_synthesis": true
}
```

Response:

```
{
  "claims": [...],
  "claim_ids": [...],
  "synthesis": "LLM-generated summary...",
  "total_found": 150
}
```

7.2.2 Graph Search

GET /api/graph/search?query=<term>&limit=<n>

Basic graph search returning nodes and edges.

Response:

```
{
  "nodes": [...],
  "edges": [...],
  "total_nodes": 100
}
```

7.3 Detection Endpoints

7.3.1 Suppression Detection

POST /api/detect/suppression

Request Body:

```
{
  "topic": "topic to analyze",
  "profile": "default"
}
```

Response:

```
{
  "score": 0.78,
  "level": "CRITICAL",
  "signals": {
    "suppression_narrative": {"score": 0.85, "indicators": [...]},
    "evidence_avoidance": {"score": 0.60, "indicators": [...]},
    "meta_claim_density": {"score": 0.45, "ratio": 0.35},
    "network_isolation": {"score": 0.30, "metrics": {...}},
    "authority_mismatch": {"score": 0.20, "indicators": [...]}
  },
  "affected_claims": [...],
  "claims_analyzed": 496
}
```

7.3.2 Coordination Detection

POST /api/detect/coordination

Request Body:

```
{
  "topic": "topic to analyze"
}
```

Response:

```
{
  "score": 0.65,
  "level": "HIGH",
  "signals": {
    "temporal_clustering": {"score": 0.70, ...},
    "language_similarity": {"score": 0.60, ...},
    "source_centralization": {"score": 0.55, ...},
    "emotion_synchronization": {"score": 0.40, ...}
  },
  "clusters": [...],
  "affected_claims": [...]
}
```

7.4 Data Management Endpoints

7.4.1 List Sources

GET /api/data-management/sources

Returns list of imported document sources.

7.4.2 Exclude Claims

POST /api/data-management/exclude

Request Body:

```
{
  "claim_ids": ["claim_1", "claim_2"]
}
```

7.4.3 Remove Source

DELETE /api/data-management/sources/<source_id>

Permanently removes a document source and all claims.

7.5 Admin Endpoints

7.5.1 System Status

GET /api/system/status

Returns health and statistics for all components.

7.5.2 Detection Setup

POST /api/admin/detection-setup

Initializes or rebuilds detection infrastructure.

8. Model Context Protocol (MCP) Server

8.1 MCP Overview

The Model Context Protocol (MCP) is an emerging standard for connecting AI assistants to external data sources and tools. Aegis Insight implements an MCP server that enables AI applications to query the knowledge graph and access multi-dimensional claim data.

Through MCP integration, AI systems can ground their responses in structured knowledge while maintaining awareness of contested topics, multiple perspectives, and epistemic uncertainty.

8.2 MCP Architecture

The MCP server (aegis_mcp_server.py) implements the MCP specification using stdio transport:

Component	Description
Transport	stdio (standard input/output)
Protocol	JSON-RPC 2.0
Authentication	None (local deployment)
Capabilities	tools, resources, prompts

8.3 Available MCP Tools

The MCP server exposes several tools for AI applications:

8.3.1 search_claims

Search for claims related to a topic using semantic similarity.

Parameters:

```
{
  "query": "topic to search",
  "limit": 50,
  "include_metadata": true
}
```

Returns:

- Matching claims with full text
- Claim type (PRIMARY/SECONDARY/CONTEXTUAL/META)
- Confidence scores
- Source document references
- Multi-dimensional metadata (temporal, geographic, emotional)

8.3.2 detect_patterns

Run pattern detection on a topic and return detection scores.

Parameters:

```
{
  "topic": "topic to analyze",
```

```
  "detection_type": "suppression|coordination|anomaly"
}
```

Returns:

- Detection score (0.0-1.0)
- Score level (LOW/MODERATE/HIGH/CRITICAL)
- Signal breakdowns
- Affected claim IDs

8.3.3 get_entity_context

Retrieve claims and relationships for a specific entity.

Parameters:

```
{
  "entity_name": "entity to query",
  "include_aliases": true
}
```

Returns:

- Entity details and known aliases
- Claims mentioning the entity
- Co-occurring entities
- Document sources

8.3.4 get_multiple_perspectives

Retrieve claims representing different perspectives on a topic.

Parameters:

```
{
  "topic": "topic to query",
  "max_perspectives": 3
}
```

Returns:

- Claims grouped by stance/perspective
- Source diversity metrics
- Temporal distribution of perspectives
- Detection flags if topic appears contested

8.4 MCP Configuration

To configure an AI application to use the Aegis MCP server:

8.4.1 Claude Desktop Configuration

Add to claude_desktop_config.json:

```
{
  "mcpServers": {
    "aegis-insight": {
      "command": "python",
      "args": ["aegis_mcp_server.py"],
      "cwd": "/path/to/aegis-insight"
    }
  }
}
```

8.4.2 Starting the MCP Server Manually

```
python aegis_mcp_server.py
```

The server communicates via stdio - it reads JSON-RPC requests from stdin and writes responses to stdout.

8.5 Use Cases for MCP Integration

Research Assistants

AI assistants can query Aegis Insight to provide multi-perspective answers with appropriate citations and caveats when topics are contested.

Fact-Checking Workflows

Before presenting claims as facts, AI systems can check detection scores to identify potentially contested or coordinated topics.

Knowledge Synthesis

AI applications can retrieve structured claims across multiple sources, enabling synthesis with full provenance tracking.

Epistemic Transparency

By surfacing detection scores and claim types, AI systems can communicate uncertainty more effectively to users.

9. Installation and Deployment

9.1 System Requirements

8.1.1 Minimum Requirements

Component	Minimum	Recommended
CPU	4 cores	8+ cores
RAM	16 GB	32+ GB
Storage	50 GB SSD	200+ GB SSD
GPU	None (CPU mode)	NVIDIA 8GB+ VRAM
OS	Linux, macOS, Windows 10+	Ubuntu 22.04 LTS

8.1.2 Software Prerequisites

- Docker 20.10+
- Docker Compose 2.0+
- Ollama (for LLM inference)
- NVIDIA Container Toolkit (for GPU support)

9.2 Docker Installation

8.2.1 Quick Start

```
# Clone the repository
git clone https://github.com/your-org/aegis-insight.git
cd aegis-insight

# Start services
docker-compose up -d

# Verify services
docker-compose ps
```

8.2.2 Docker Compose Configuration

The docker-compose.yml defines all services:

```
services:
  neo4j:
    image: neo4j:5.15.0-community
    ports:
      - '7474:7474'
      - '7687:7687'
    environment:
      NEO4J_AUTH: neo4j/aegistrusted

  postgres:
```

```
    image: pgvector/pgvector:pg16
    ports:
      - '5432:5432'
    environment:
      POSTGRES_USER: aegis
      POSTGRES_PASSWORD: aegis_trusted_2025
      POSTGRES_DB: aegis_insight

  api:
    build: .
    ports:
      - '8001:8001'
    depends_on:
      - neo4j
      - postgres
```

9.3 Ollama Setup

Ollama runs on the host machine for GPU access.

8.3.1 Installation

```
# Linux/macOS
curl -fsSL https://ollama.com/install.sh | sh

# Windows
# Download from https://ollama.com/download
```

8.3.2 Model Installation

```
# Required model
ollama pull mistral-nemo:12b

# Embedding model
ollama pull nomic-embed-text
```

8.3.3 Verify Installation

```
ollama list
# Should show mistral-nemo:12b and nomic-embed-text
```

9.4 Database Configuration

8.4.1 Neo4j Credentials

Setting	Value
Bolt URL	bolt://localhost:7687
HTTP URL	http://localhost:7474

Setting	Value
Username	neo4j
Password	aegistrusted

8.4.2 PostgreSQL Credentials

Setting	Value
Host	localhost
Port	5432
Database	aegis_insight
Username	aegis
Password	aegis_trusted_2025

9.5 Verification

After installation, verify the system:

16. Open http://localhost:8001 in your browser
17. Navigate to the Admin tab
18. Verify all status indicators show green/ready
19. Try a search to confirm graph connectivity

10. Troubleshooting

10.1 Common Issues

9.1.1 Services Not Starting

Symptom: Docker containers exit immediately

Solutions:

- Check Docker logs: docker-compose logs <service>
- Verify ports are available (7474, 7687, 5432, 8001)
- Ensure sufficient disk space (50GB+ free)
- Check memory availability (16GB+ recommended)

9.1.2 Ollama Connection Failed

Symptom: Extraction fails with connection error

Solutions:

- Verify Ollama is running: ollama list
- Check Ollama port: curl http://localhost:11434/api/tags
- Restart Ollama: systemctl restart ollama (Linux)

9.1.3 Low Detection Scores

Symptom: Detection returns unexpectedly low scores

Solutions:

- Verify topic has sufficient claims in database
- Check embedding generation completed
- Try alternative search terms
- Review calibration profile settings

9.1.4 Graph Visualization Issues

Symptom: Graph displays incorrectly or errors

Solutions:

- Hard refresh browser (Ctrl+Shift+R)
- Check browser console for JavaScript errors
- Reduce node limit if displaying many nodes
- Clear browser cache

9.1.5 Import Failures

Symptom: Document import fails or hangs

Solutions:

- Check Ollama is running with correct model
- Verify input file format is supported
- Check disk space for temporary files
- Review API server logs for errors

10.2 Log Locations

Component	Log Location
API Server	Console output or api_server.log
Import Jobs	~/.aegis/jobs/<job_id>/import.log
Neo4j	docker logs aegis-neo4j
PostgreSQL	docker logs aegis-postgres
Ollama	~/.ollama/logs/ or journalctl -u ollama

10.3 Performance Tuning

9.3.1 Neo4j Memory

For large datasets, increase Neo4j heap size:

```
# In docker-compose.yml
environment:
  NEO4J_dbms_memory_heap_max__size: 4G
  NEO4J_dbms_memory_pagecache_size: 2G
```

9.3.2 PostgreSQL Tuning

```
# Connect to PostgreSQL
docker exec -it aegis-postgres psql -U aegis

# Increase work memory for complex queries
SET work_mem = '256MB';
```

9.3.3 Batch Processing

For large imports, adjust batch sizes:

- Smaller batches (10-20) for memory-constrained systems
- Larger batches (50-100) for faster processing

Appendix A: Keyboard Shortcuts

Shortcut	Action
Enter	Execute search
Escape	Clear selection / close panel
Ctrl+R / Cmd+R	Refresh graph
Scroll / Pinch	Zoom in/out
+/-	Zoom in/out
Arrow keys	Pan view

Appendix B: Glossary

Claim

A factual assertion extracted from a document. The atomic unit of knowledge in Aegis Insight.

Claim Type

Classification of a claim as PRIMARY (direct assertion), SECONDARY (supporting), CONTEXTUAL (background), or META (about other claims).

Embedding

A vector representation of text that captures semantic meaning, enabling similarity search.

Entity

A named thing mentioned in documents: person, organization, place, concept, or event.

Epistemic Topology

The structural relationships between claims, sources, and narratives in a knowledge graph.

Extraction

The process of identifying structured knowledge (claims, entities, metadata) from unstructured text.

Goldfinger Scoring

A scoring method where multiple independent indicators compound the score, based on the principle that coincident patterns suggest non-random causes.

Knowledge Graph

A database structure representing entities and their relationships as nodes and edges.

META Claim

A claim about other claims, often indicating contested knowledge or narrative framing.

Pattern Detection

Algorithms that identify structural features in the knowledge graph that deviate from baseline expectations.

Semantic Similarity

A measure of conceptual relatedness between texts, computed via embedding vector cosine distance.

Topic Traversal

A graph query pattern that finds claims by traversing from Topic nodes through Documents to Claims.

Appendix C: File Reference

Core Application Files

File	Purpose
api_server.py	Main FastAPI server
aegis_graph_builder.py	Neo4j graph construction
generate_embeddings.py	Vector embedding generation
pattern_search_llm.py	Semantic search with LLM
aegis_import_wizard.py	CLI import wizard

Extractors

File	Purpose
aegis_entity_extractor.py	Entity extraction
aegis_claim_extractor.py	Claim extraction
aegis_temporal_extractor.py	Temporal data
aegis_geographic_extractor.py	Geographic data
aegis_citation_extractor.py	Citation data
aegis_emotion_extractor.py	Emotional analysis
aegis_authority_domain_analyzer.py	Authority analysis

Detectors

File	Purpose
aegis_suppression_detector_v2.py	Suppression detection
aegis_coordination_detector.py	Coordination detection
aegis_anomaly_detector.py	Anomaly detection

Web Interface

Path	Purpose
web/index.html	Main HTML
web/js/graph/GraphRenderer.js	D3 visualization
web/js/detection/DetectionControls.js	Detection UI
web/js/wizard/DataWizard.js	Import wizard UI

Document Information

Title: Aegis Insight Technical Reference Manual

Version: 3.0

Release Date: December 2025

Document Status: Production

This manual provides comprehensive technical documentation for Aegis Insight version 3.0. Features and interfaces may change in future versions. For the latest documentation, refer to the project repository.

Acknowledgments

Aegis Insight builds upon excellent open-source projects including Neo4j, PostgreSQL, pgvector, Ollama, FastAPI, and D3.js. We gratefully acknowledge these communities.