CANONICAL LIVING OVERVIEW GUIDE

Knowledge, RAG, Memory, Grounding, and Source Authority

A grounded AI guide covering knowledge ingestion, parsing, chunking, embeddings, lexical and dense retrieval, reranking, graph retrieval, citations, memory tiers, source authority, freshness, access control, evaluation, and lifecycle.

PERMANENT ID
MYTHOS-FG-AI-0008

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Knowledge, RAG, Memory, Grounding, and Source Authority

Document ID
MYTHOS-FG-AI-0008

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SEC; MYTHOS-SWE
Audience	RAG, knowledge, search, memory, agent, data, security, and AI application engineers.
Prerequisites	Information retrieval, databases, AI models, data governance, APIs, and evaluation.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Build knowledge pipelines that preserve identity, structure, source authority, access, and freshness.

Combine lexical, dense, metadata, graph, and reranking methods according to task.

Generate answers with traceable evidence, citation validation, uncertainty, and conflict disclosure.

Design working, episodic, semantic, procedural, and user memory with explicit retention and authority.

Evaluate retrieval, grounding, citation, memory, security, and end-to-end task outcomes.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Knowledge architecture and source authority

Define which sources may support which claims and decisions.

Chapter operating position
Define which sources may support which claims and decisions.

1.1 Source classes and authority

Source classes and authority must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Distinguish normative, primary, internal authoritative, operational evidence, expert, secondary, and unverified sources. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for source classes and authority.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating source classes and authority as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for source classes and authority.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

1.2 Identity and lineage

Identity and lineage must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Assign stable IDs, versions, owners, origin, transformations, permissions, hashes, and supersession. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for identity and lineage.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating identity and lineage as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for identity and lineage.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

1.3 Freshness and validity

Freshness and validity must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Track publication, verification, effective, expiry, review, change triggers, and conflicting versions. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for freshness and validity.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating freshness and validity as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for freshness and validity.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

1.4 Claim and use restrictions

Claim and use restrictions must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Limit source use by domain, user, tenant, geography, license, confidentiality, and decision risk. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for claim and use restrictions.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating claim and use restrictions as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for claim and use restrictions.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 01 laboratory and review

Practical laboratory

Create a source-authority policy and resolve a conflict among a standard, vendor guide, internal runbook, and stale incident note.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore cryptographically verifiable knowledge lineage and automated authority scoring.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Ingestion, parsing, and canonical representation

Convert heterogeneous sources without losing structure and provenance.

Chapter operating position
Convert heterogeneous sources without losing structure and provenance.

2.1 Connectors and acquisition

Connectors and acquisition must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Retrieve files, repositories, databases, APIs, messages, tickets, pages, media, and structured records. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for connectors and acquisition.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating connectors and acquisition as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for connectors and acquisition.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

2.2 Parsing and structure

Parsing and structure must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Preserve documents, sections, tables, code, diagrams, pages, fields, timestamps, and relationships. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for parsing and structure.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating parsing and structure as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for parsing and structure.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

2.3 Normalization and enrichment

Normalization and enrichment must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Add canonical IDs, entities, language, permissions, topics, embeddings, links, and quality metadata. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for normalization and enrichment.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating normalization and enrichment as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for normalization and enrichment.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

2.4 Change capture and deletion

Change capture and deletion must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Detect additions, modifications, moves, supersession, revocation, retention, and source removal. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for change capture and deletion.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating change capture and deletion as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for change capture and deletion.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 02 laboratory and review

Practical laboratory
Ingest a mixed corpus and prove every normalized segment can be traced to its original source and access policy.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore multimodal semantic parsing and continuously reconciled knowledge twins.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Chunking, embeddings, and indexes

Create retrieval units aligned to meaning, tasks, and source structure.

Chapter operating position
Create retrieval units aligned to meaning, tasks, and source structure.

3.1 Chunk boundaries

Chunk boundaries must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use semantic, structural, sliding, code-aware, table-aware, and hierarchical segmentation. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for chunk boundaries.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating chunk boundaries as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for chunk boundaries.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

3.2 Embedding models and spaces

Embedding models and spaces must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Select model, dimension, language, domain, normalization, version, and privacy. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for embedding models and spaces.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating embedding models and spaces as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for embedding models and spaces.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

3.3 Lexical and vector indexes

Lexical and vector indexes must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use inverted indexes, ANN structures, metadata filters, namespaces, updates, and tombstones. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for lexical and vector indexes.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating lexical and vector indexes as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for lexical and vector indexes.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

3.4 Index quality and drift

Index quality and drift must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure coverage, duplicates, orphan chunks, embedding changes, source mismatch, and stale entries. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for index quality and drift.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating index quality and drift as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for index quality and drift.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 03 laboratory and review

Practical laboratory
Compare fixed, structural, and semantic chunking on a technical corpus using retrieval and citation outcomes.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore late-interaction retrieval and learned task-specific indexes.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Retrieval, reranking, and graph reasoning

Find evidence through complementary retrieval strategies.

Chapter operating position
Find evidence through complementary retrieval strategies.

4.1 Query understanding

Query understanding must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Extract intent, entities, constraints, time, authority, permissions, subquestions, and expected evidence. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for query understanding.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating query understanding as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for query understanding.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

4.2 Hybrid retrieval

Hybrid retrieval must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Combine keyword, dense, metadata, recency, authority, popularity, and exact identifiers. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for hybrid retrieval.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating hybrid retrieval as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for hybrid retrieval.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

4.3 Reranking and diversity

Reranking and diversity must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use cross-encoders or judges, relevance, authority, freshness, novelty, redundancy, and coverage. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for reranking and diversity.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating reranking and diversity as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for reranking and diversity.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

4.4 Graph and multi-hop retrieval

Graph and multi-hop retrieval must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Traverse entities, documents, communities, dependencies, citations, and temporal relationships. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for graph and multi-hop retrieval.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating graph and multi-hop retrieval as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for graph and multi-hop retrieval.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 04 laboratory and review

Practical laboratory
Build a hybrid retrieval pipeline and compare dense-only, lexical-only, reranked, and graph-assisted results.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore agentic retrieval planning and causal knowledge graphs.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Grounded generation and citation engineering

Ensure outputs remain linked to evidence and disclose unsupported claims.

Chapter operating position
Ensure outputs remain linked to evidence and disclose unsupported claims.

5.1 Evidence packaging

Evidence packaging must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Provide source text, metadata, authority, date, permissions, section, page, and stable citation ID. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for evidence packaging.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating evidence packaging as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for evidence packaging.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

5.2 Answer construction

Answer construction must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Separate sourced facts, calculations, inference, uncertainty, conflicts, and recommendations. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for answer construction.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating answer construction as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for answer construction.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

5.3 Citation validation

Citation validation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Verify cited text supports the claim, source exists, permissions allow use, and locations are stable. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for citation validation.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating citation validation as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for citation validation.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

5.4 Insufficient evidence and abstention

Insufficient evidence and abstention must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Ask for data, narrow the claim, present alternatives, or decline unsupported certainty. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for insufficient evidence and abstention.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating insufficient evidence and abstention as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for insufficient evidence and abstention.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 05 laboratory and review

Practical laboratory
Generate answers over a conflicting corpus and score claim-level support, authority, freshness, and citation correctness.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore proof-producing generation and machine-checkable claim graphs.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Memory models and lifecycle

Store durable state without confusing memory, knowledge, conversation, and authority.

Chapter operating position
Store durable state without confusing memory, knowledge, conversation, and authority.

6.1 Working and short-term memory

Working and short-term memory must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Maintain active goals, variables, constraints, intermediate results, and recent interaction. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for working and short-term memory.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating working and short-term memory as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for working and short-term memory.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

6.2 Episodic and trajectory memory

Episodic and trajectory memory must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Store events, tasks, actions, observations, outcomes, errors, and lessons with timestamps. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for episodic and trajectory memory.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating episodic and trajectory memory as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for episodic and trajectory memory.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

6.3 Semantic and procedural memory

Semantic and procedural memory must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Represent stable facts, preferences, concepts, policies, workflows, and skill knowledge. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for semantic and procedural memory.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating semantic and procedural memory as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for semantic and procedural memory.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

6.4 Retention, consent, and forgetting

Retention, consent, and forgetting must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Control user choice, purpose, access, expiry, correction, deletion, and derived artifacts. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for retention, consent, and forgetting.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating retention, consent, and forgetting as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for retention, consent, and forgetting.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 06 laboratory and review

Practical laboratory
Design a memory schema and demonstrate promotion, correction, expiry, deletion, and cross-session retrieval.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore personal semantic memory with user-owned encrypted stores and verifiable forgetting.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Security, privacy, and multi-tenant isolation

Prevent knowledge systems from becoming exfiltration and authority-confusion channels.

Chapter operating position
Prevent knowledge systems from becoming exfiltration and authority-confusion channels.

7.1 Access-aware retrieval

Access-aware retrieval must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Apply user, role, tenant, document, field, purpose, and row-level permissions before ranking. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for access-aware retrieval.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating access-aware retrieval as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for access-aware retrieval.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

7.2 Indirect prompt injection

Indirect prompt injection must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Treat documents, pages, messages, code, images, and tool outputs as untrusted data. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for indirect prompt injection.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating indirect prompt injection as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for indirect prompt injection.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

7.3 Sensitive data and embeddings

Sensitive data and embeddings must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Control secrets, personal data, model inversion risk, logs, caches, backups, and index exports. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for sensitive data and embeddings.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating sensitive data and embeddings as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for sensitive data and embeddings.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

7.4 Poisoning and integrity

Poisoning and integrity must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Detect malicious sources, modified content, fake authority, link manipulation, and embedding attacks. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for poisoning and integrity.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating poisoning and integrity as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for poisoning and integrity.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 07 laboratory and review

Practical laboratory

Red-team a multi-tenant RAG system for cross-tenant retrieval, injection, poisoned sources, stale permissions, and citation spoofing.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore encrypted retrieval, confidential vector search, and signed source assertions.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Evaluation, observability, and lifecycle operations

Measure the complete knowledge-to-answer pipeline and keep it current.

Chapter operating position
Measure the complete knowledge-to-answer pipeline and keep it current.

8.1 Retrieval metrics

Retrieval metrics must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure recall, precision, ranking, nDCG, authority, freshness, diversity, and permission correctness. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for retrieval metrics.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

- Treating retrieval metrics as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.
- Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.
- Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.
- Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.
- Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for retrieval metrics.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

8.2 Grounding and answer metrics

Grounding and answer metrics must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure claim support, citation precision, contradiction, completeness, abstention, and user outcome. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

- Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for grounding and answer metrics.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
claim:
  text: "The active policy requires phishing-resistant authentication."
  support:
    source_id: POLICY-IAM-004
    version: 7
    location: section-3.2
    authority: internal-normative
    verified_at: 2026-07-26
  inference: false
  citation_validated: true
  conflict_state: none
```

Failure patterns

- Treating grounding and answer metrics as a model capability claim disconnected from complete system behavior.
- Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for grounding and answer metrics.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

8.3 Pipeline observability

Pipeline observability must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Trace query, filters, candidates, scores, reranking, context, generation, citations, cache, and latency. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for pipeline observability.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating pipeline observability as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for pipeline observability.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

8.4 Reindex, migration, and incident response

Reindex, migration, and incident response must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Handle source changes, embedding upgrades, corrupt indexes, leaks, model changes, rollback, and evidence. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the source, lineage, parsing, chunk, index, retrieval, rerank, grounded claim, citation, memory, and lifecycle chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

Implementation begins with a small, inspectable baseline and explicit invariants. Introduce one model, dataset, retrieval source, tool, memory tier, optimization, or permission at a time; preserve before-state evidence; test expected and prohibited behavior; and retain a reversible path. Generated plans and model judgments remain subject to deterministic validation and human authority.

Troubleshooting locates the first divergence from the expected contract or state machine. Account for tokenizer and template mismatch, stale data, context loss, model nondeterminism, caching, quantization, scheduling, permissions, prompt injection, missing telemetry, partial tool effects, and distribution shift. Completion requires reproducibility, causal evidence, validated recovery, and clear disclosure of uncertainty.

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for reindex, migration, and incident response.

Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.

Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.

Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.

Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.

Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Failure patterns

Treating reindex, migration, and incident response as a model capability claim disconnected from complete system behavior.

Using one benchmark, model judge, or successful demonstration as proof of production readiness.

Ignoring tokenizer, prompt template, source authority, permissions, cache, hardware, or version boundaries.

Allowing tools, plugins, retrieval content, or memory to redefine trusted instructions.

Optimizing latency, tokens, or cost without measuring quality, safety, and tail failures.

Declaring completion without hidden tests, evidence, residual limitations, rollback, and monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for reindex, migration, and incident response.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 08 laboratory and review

Practical laboratory
Create an end-to-end RAG evaluation set and operational dashboard with failure slices and reindex rollback.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop self-auditing knowledge systems that continuously verify authority and citation integrity.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. Meta AI Research - Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
Role: Canonical RAG architecture paper.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2005.11401

02. Meta AI Research - Dense Passage Retrieval for Open-Domain Question Answering
Role: Dense-retrieval architecture and evaluation.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2004.04906

03. Microsoft Research - GraphRAG
Role: Graph-based retrieval and community summarization implementation.
Verified: 2026-07-26
Official source: https://github.com/microsoft/graphrag

04. NIST - NIST AI 600-1 - Generative Artificial Intelligence Profile
Role: Generative-AI risk profile and recommended actions.
Verified: 2026-07-26
Official source: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

05. OWASP GenAI Security Project - OWASP Top 10 for LLM and GenAI Applications 2025
Role: Risk and mitigation baseline for generative-AI applications.
Verified: 2026-07-26
Official source: https://genai.owasp.org/llm-top-10/

06. Model Context Protocol - Model Context Protocol Specification 2025-11-25
Role: Authoritative protocol requirements for resources, prompts, tools, transports, authorization, and lifecycle.
Verified: 2026-07-26
Official source: https://modelcontextprotocol.io/specification/2025-11-25

07. JSON Schema - JSON Schema 2020-12
Role: Typed JSON instance and schema validation.
Verified: 2026-07-26
Official source: https://json-schema.org/draft/2020-12

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SEC; MYTHOS-SWE
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	RAG, retrieval, grounding, citations, memory, source authority, embeddings, GraphRAG, knowledge engineering, access-aware retrieval
Canonical route	/library/field-guides/mythos-fg-ai-0008/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.