CANONICAL LIVING OVERVIEW GUIDE

Context Engineering, Token Management, and Trajectory Compaction

A context-systems guide covering tokenization, context budgets, prompt anatomy, relevance, long context, position effects, caches, summaries, compaction, trajectory records, working sets, multi-agent context, security, and evaluation.

PERMANENT ID
MYTHOS-FG-AI-0007

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Context Engineering, Token Management, and Trajectory Compaction

Document ID
MYTHOS-FG-AI-0007

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-SWE; MYTHOS-SEC
Audience	AI application, agent, memory, retrieval, platform, and developer-tool engineers.
Prerequisites	Transformers, tokenization, prompting, retrieval, software architecture, and evaluation.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Allocate context among instructions, task state, repository data, retrieval, tools, memory, and output.

Select and order evidence according to relevance, authority, freshness, dependency, and position.

Compact long trajectories without losing commitments, unresolved work, provenance, or recovery state.

Measure token, cache, latency, quality, and context-loss tradeoffs.

Protect context from injection, cross-tenant leakage, secret exposure, and untrusted summaries.

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

Context architecture and token accounting

Treat the context window as a scarce, typed working memory.

Chapter operating position
Treat the context window as a scarce, typed working memory.

1.1 Tokenizer behavior

Tokenizer behavior must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Understand tokens, bytes, Unicode, code, tables, images, special tokens, and model-specific differences. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 tokenizer behavior.
- 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 tokenizer behavior 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 tokenizer behavior.
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 Budget partitions

Budget partitions must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Allocate system policy, user task, state, retrieval, memory, tools, examples, reasoning allowance, and output. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 budget partitions.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

Treating budget partitions 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 budget partitions.
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 Hard and soft limits

Hard and soft limits must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Handle maximum context, reserved output, provider limits, tool schemas, multimodal tokens, and safety margins. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 hard and soft limits.

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 hard and soft limits 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 hard and soft limits.
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 Cost and latency

Cost and latency must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Relate input tokens, cache hits, prefill, throughput, provider cost, memory, and user delay. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 cost and latency.

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 cost and latency 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 cost and latency.
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
Build a token budgeter that rejects, truncates, retrieves, or compacts according to typed context classes.

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 semantic token allocation and adaptive model-specific budgets.

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

Instruction hierarchy and prompt structure

Separate authority, task, data, examples, and untrusted content.

Chapter operating position
Separate authority, task, data, examples, and untrusted content.

2.1 System and policy instructions

System and policy instructions must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Define identity, authority, constraints, tool rules, output contract, and conflict handling. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 system and policy instructions.
- 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 system and policy instructions 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 system and policy instructions.
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 Task and acceptance context

Task and acceptance context must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Provide objective, scope, dependencies, state, artifacts, tests, and completion 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 task and acceptance context.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

- Treating task and acceptance context 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 task and acceptance context.
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 Examples and demonstrations

Examples and demonstrations must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Choose representative, diverse, non-leaking examples with clear boundaries and labels. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 examples and demonstrations.
- 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 examples and demonstrations 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 examples and demonstrations.
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 Data and untrusted content

Data and untrusted content must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Mark documents, tool outputs, web content, code comments, messages, and retrieved text as data rather than authority. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 data and untrusted content.

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 data and untrusted content 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 data and untrusted content.
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

Create a prompt layout resistant to indirect instructions embedded in retrieved documents and repository files.

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 typed prompt languages and cryptographically labeled instruction provenance.

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

Selection, relevance, authority, and freshness

Choose the smallest context that preserves the decision-relevant system state.

Chapter operating position
Choose the smallest context that preserves the decision-relevant system state.

3.1 Relevance and dependency

Relevance and dependency must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Rank direct task evidence, interfaces, callers, state, tests, related incidents, and neighboring modules. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 relevance and dependency.
- 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 relevance and dependency 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 relevance and dependency.
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 Authority and source tiers

Authority and source tiers must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Prefer normative specifications, current source of truth, approved policy, primary evidence, and explicit user decisions. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 authority and source tiers.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

- Treating authority and source tiers 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 authority and source tiers.
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 Freshness and change detection

Freshness and change detection must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Track versions, timestamps, commits, schema, model, policy, index, and invalidation triggers. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 change detection.
- 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 change detection 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 change detection.
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 Diversity and contradiction

Diversity and contradiction must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Include competing evidence, unresolved conflicts, uncertainty, and minority failure cases when decision-relevant. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 diversity and contradiction.

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 diversity and contradiction 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 diversity and contradiction.
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
Score a candidate context set and remove low-value material while preserving all authoritative dependencies and contradictions.

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 learned context selection constrained by source authority.

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

Long-context behavior and position effects

Use large windows without assuming uniform attention or recall.

Chapter operating position
Use large windows without assuming uniform attention or recall.

4.1 Position sensitivity

Position sensitivity must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Test beginning, middle, end, recency, ordering, separators, and repeated instructions. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 position sensitivity.
- 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 position sensitivity 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 position sensitivity.
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 Distractors and overload

Distractors and overload must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure irrelevant documents, duplicated text, similar identifiers, conflicting versions, and noisy examples. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 distractors and overload.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

Treating distractors and overload 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 distractors and overload.
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 Hierarchical organization

Hierarchical organization must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use maps, summaries, indexes, sections, citations, and progressive disclosure. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 hierarchical organization.

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 hierarchical organization 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 hierarchical organization.
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 Long-context evaluation

Long-context evaluation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Create needle, multi-hop, synthesis, chronology, codebase, and state-consistency tests. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 long-context evaluation.

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 long-context evaluation 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 long-context evaluation.
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
Run a lost-in-the-middle experiment with controlled evidence positions and distractor density.

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 recurrent memory, infinite-context approximations, and external attention stores.

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

Caching, reuse, and stable prefixes

Reuse expensive context without serving stale or cross-tenant state.

Chapter operating position
Reuse expensive context without serving stale or cross-tenant state.

5.1 Prompt and prefix caching

Prompt and prefix caching must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Identify stable instructions, schemas, repositories, documents, and repeated conversation prefixes. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 prompt and prefix caching.
- 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 prompt and prefix caching 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 prompt and prefix caching.
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 Cache identity and isolation

Cache identity and isolation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Key by model, tokenizer, template, policy, tenant, data version, adapter, and tool schema. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 cache identity and isolation.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

- Treating cache identity and isolation 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 cache identity and isolation.
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 Invalidation and freshness

Invalidation and freshness must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Expire or rebuild on policy, source, permission, model, index, or user-state change. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 invalidation and freshness.
- 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 invalidation and freshness 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 invalidation and freshness.
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 Quality and economics

Quality and economics must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure prefill avoided, latency, memory, hit rate, fragmentation, leakage risk, and stale results. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 quality and economics.

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 quality and economics 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 quality and economics.
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
Design a cache key and invalidation strategy for multi-tenant repository agents.

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 distributed semantic caches with verifiable privacy boundaries.

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

Summarization and trajectory compaction

Compress interaction history while preserving executable state and evidence.

Chapter operating position
Compress interaction history while preserving executable state and evidence.

6.1 Conversation and task summaries

Conversation and task summaries must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Preserve objective, decisions, constraints, artifacts, results, unresolved issues, and next actions. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 conversation and task summaries.
- 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 conversation and task summaries 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 conversation and task summaries.
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 Tool trajectory records

Tool trajectory records must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Store action, arguments, observation, state change, evidence, error, retry, and compensation. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 tool trajectory records.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

- Treating tool trajectory records 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 tool trajectory records.
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 Loss and hallucination controls

Loss and hallucination controls must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use source-linked summaries, deterministic extraction, validation, confidence, and retained raw 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 loss and hallucination controls.
- 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 loss and hallucination controls 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 loss and hallucination controls.
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 Checkpoint and restore

Checkpoint and restore must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Create compact state snapshots that allow another model or agent to resume and verify work. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 checkpoint and restore.

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 checkpoint and restore 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 checkpoint and restore.
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
Compact a long agent trajectory into a resume package and test whether another agent can continue without repeating or violating decisions.

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 learned trajectory compression and event-sourced agent memory.

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

Multi-agent and multi-model context fabric

Share necessary state without flooding every agent or leaking authority.

Chapter operating position
Share necessary state without flooding every agent or leaking authority.

7.1 Private and shared context

Private and shared context must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Separate agent scratch, task state, evidence, decisions, common memory, and restricted 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 private and shared context.
- 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 private and shared context 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 private and shared context.
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 Role-specific views

Role-specific views must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Give planners, implementers, reviewers, testers, and supervisors different context projections. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 role-specific views.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

- Treating role-specific views 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 role-specific views.
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 Handoffs and fan-in

Handoffs and fan-in must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use typed deliverables, claims, citations, conflicts, confidence, and unresolved dependencies. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 handoffs and fan-in.
- 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 handoffs and fan-in 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 handoffs and fan-in.
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 Consistency and concurrency

Consistency and concurrency must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Handle simultaneous edits, stale plans, duplicated work, conflicting summaries, and supervisor reconciliation. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 consistency and concurrency.

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 consistency and concurrency 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 consistency and concurrency.
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
Run three role-specific agents on one task using separate context views and a governed reconciliation record.

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 distributed context services with transactional state and semantic access control.

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

Security, evaluation, and operations

Treat context as a high-value execution and information-security boundary.

Chapter operating position
Treat context as a high-value execution and information-security boundary.

8.1 Prompt injection and authority confusion

Prompt injection and authority confusion must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Test direct, indirect, retrieved, tool, memory, image, and repository instructions. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 prompt injection and authority confusion.
- 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 prompt injection and authority confusion 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 prompt injection and authority confusion.
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 Secrets and cross-tenant isolation

Secrets and cross-tenant isolation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Prevent credentials, private data, hidden prompts, embeddings, cache entries, and logs from crossing boundaries. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 secrets and cross-tenant isolation.
- 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

```
context_budget:
  total_tokens: 65536
  reserve_output: 8192
  partitions:
    system_policy: 4000
    task_state: 8000
    repository_map: 10000
    retrieved_evidence: 24000
    trajectory_summary: 6000
  overflow_policy:
    - remove_low_authority
    - deduplicate
    - retrieve_on_demand
    - compact_with_citations
```

Failure patterns

- Treating secrets and cross-tenant isolation 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 secrets and cross-tenant isolation.
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 Context quality metrics

Context quality metrics must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure relevance, authority, freshness, coverage, contradiction, token efficiency, cache reuse, and task 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 context quality 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 context quality 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 context quality 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.

8.4 Lifecycle and incident response

Lifecycle and incident response must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Trace context versions, delete data, invalidate caches, inspect leaks, restore safe state, and update policy. 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 authority, instruction, selected evidence, token budget, working state, cache, compaction, handoff, and task outcome 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 lifecycle 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 lifecycle 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 lifecycle 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

Red-team a context pipeline for injection, stale source, summary corruption, cache leakage, and missing critical evidence.

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 a source-authority-aware context operating system with continuous evaluation.

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. Stanford University - Lost in the Middle
Role: Long-context position sensitivity evaluation.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2307.03172

02. Google Brain / CMU - Transformer-XL
Role: Segment-level recurrence and long-context modeling.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/1901.02860

03. UC Berkeley - MemGPT: Towards LLMs as Operating Systems
Role: Tiered memory and context-management architecture.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2310.08560

04. vLLM Project - Automatic Prefix Caching
Role: Prefix-cache architecture and reuse behavior.
Verified: 2026-07-26
Official source: https://docs.vllm.ai/en/stable/design/prefix_caching/

05. 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

06. 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

07. 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/

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-SWE; MYTHOS-SEC
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	context engineering, token management, context window, trajectory compaction, prompt caching, long context, summarization, multi-agent context, prompt injection, context evaluation
Canonical route	/library/field-guides/mythos-fg-ai-0007/

Living publication

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