CANONICAL LIVING OVERVIEW GUIDE

Fine-Tuning, Alignment, Synthetic Data, and Training Governance

A training and post-training guide covering objectives, datasets, SFT, PEFT, LoRA, QLoRA, preferences, DPO, reward models, synthetic data, curriculum, distributed training, evaluation, safety, provenance, release, and rollback.

PERMANENT ID
MYTHOS-FG-AI-0005

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Fine-Tuning, Alignment, Synthetic Data, and Training Governance

Document ID
MYTHOS-FG-AI-0005

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 Foundational through frontier	FIELD GUIDE Practical and educational	QUARTERLY Interim updates as needed	SOURCE-BOUND Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SEC; MYTHOS-SWE
Audience	Model engineers, data teams, AI researchers, evaluators, product teams, and governance leaders.
Prerequisites	Model architecture, optimization, data engineering, statistics, evaluation, and ML operations.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

- Choose adaptation methods according to capability gap, data, compute, risk, and deployment constraints.
- Govern training data, synthetic data, labels, licenses, privacy, provenance, and contamination.
- Implement SFT, LoRA-style adaptation, preference optimization, and evaluation safely.
- Detect overfitting, forgetting, reward hacking, alignment tradeoffs, and distribution shift.
- Release adapters or models with reproducible training records, evaluation, approval, and rollback.

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

Training objective and adaptation strategy

Define the capability change and select the least risky effective method.

Chapter operating position
Define the capability change and select the least risky effective method.

1.1 Capability gap and target behavior

Capability gap and target behavior must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Specify tasks, users, languages, domains, formats, safety, abstention, and failure costs. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 capability gap and target 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 capability gap and target 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 capability gap and target 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 Prompting, retrieval, tools, or training

Prompting, retrieval, tools, or training must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Determine whether behavior belongs in weights, context, data, tools, policy, or interface. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 prompting, retrieval, tools, or training.
- 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

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

Treating prompting, retrieval, tools, or training 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 prompting, retrieval, tools, or training.
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 Full and parameter-efficient tuning

Full and parameter-efficient tuning must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Compare full updates, adapters, LoRA, QLoRA, prompt tuning, and selective layers. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 full and parameter-efficient tuning.

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 full and parameter-efficient tuning 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 full and parameter-efficient tuning.
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 Success and rollback criteria

Success and rollback criteria must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Define evaluation, resource, safety, compatibility, deployment, and retirement gates. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 success and rollback criteria.
- 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 success and rollback criteria 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 success and rollback criteria.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 01 laboratory and review

Practical laboratory
Create an adaptation decision for a domain assistant and justify training versus retrieval, tools, and prompting.

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 modular skill adapters and dynamically composed model capabilities.

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

Dataset design, provenance, and governance

Build training material that is lawful, representative, and traceable.

Chapter operating position
Build training material that is lawful, representative, and traceable.

2.1 Sources and licensing

Sources and licensing must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Record origin, permission, terms, ownership, consent, restrictions, and downstream obligations. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 sources and licensing.

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 sources and licensing 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 sources and licensing.
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 Schema and examples

Schema and examples must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Define instructions, inputs, context, outputs, reasoning policy, tools, metadata, and quality 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 schema and examples.
- 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

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

Treating schema and examples 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 schema and examples.
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 Cleaning and deduplication

Cleaning and deduplication must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Remove corruption, secrets, personal data, duplicates, benchmark leakage, unsafe content, and low-value patterns. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 cleaning and deduplication.

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 cleaning and deduplication 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 cleaning and deduplication.
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 Splits and versioning

Splits and versioning must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Separate train, validation, holdout, safety, and regression data with immutable manifests and lineage. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 splits and versioning.
- 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 splits and versioning 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 splits and versioning.
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 versioned training-data manifest and identify licensing, privacy, leakage, quality, and representativeness issues.

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 data trusts, machine-readable consent, and cryptographically verifiable dataset lineage.

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

Supervised fine-tuning and PEFT

Teach desired response and task patterns efficiently.

Chapter operating position
Teach desired response and task patterns efficiently.

3.1 Tokenization and packing

Tokenization and packing must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Apply templates, labels, masking, truncation, sequence packing, loss boundaries, and special tokens. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 tokenization and packing.
- 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 tokenization and packing 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 tokenization and packing.
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 LoRA and low-rank adaptation

LoRA and low-rank adaptation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Select target modules, rank, scaling, dropout, initialization, merge, and adapter lifecycle. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 lora and low-rank adaptation.
- 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

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

Treating lora and low-rank adaptation 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 lora and low-rank adaptation.
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 Quantized fine-tuning

Quantized fine-tuning must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Combine low-bit base weights with higher-precision adapters, optimizer state, calibration, and hardware. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 quantized fine-tuning.

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 quantized fine-tuning 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 quantized fine-tuning.
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 Training stability

Training stability must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Control learning rate, batch, gradient accumulation, clipping, precision, checkpointing, and validation. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 training stability.

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 training stability 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 training stability.
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
Fine-tune a tiny PyTorch language model or adapter on a controlled dataset and inspect overfit, loss, and held-out behavior.

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 adapter routing, sparse updates, and continual low-rank learning.

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

Preference optimization and alignment

Optimize comparative preferences without hiding evaluator assumptions.

Chapter operating position
Optimize comparative preferences without hiding evaluator assumptions.

4.1 Preference data

Preference data must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Create chosen and rejected pairs, rankings, critiques, constitutions, and uncertainty 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 preference data.
- 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 preference data 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 preference data.
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 Reward models and feedback

Reward models and feedback must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Train and validate preference models, detect shortcuts, disagreement, distribution shift, and reward hacking. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 reward models and feedback.
- 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

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

- Treating reward models and feedback 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 reward models and feedback.
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 DPO and related objectives

DPO and related objectives must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Understand reference policy, preference likelihood, regularization, beta, and optimization behavior. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 dpo and related objectives.

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 dpo and related objectives 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 dpo and related objectives.
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 Safety and capability tradeoffs

Safety and capability tradeoffs must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure refusal, helpfulness, truthfulness, calibration, style, and task performance by slice. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 safety and capability tradeoffs.

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

Chapter 04 laboratory and review

Practical laboratory
Build a small preference dataset and compare rule-based ranking, reward scoring, and DPO-style objectives conceptually or experimentally.

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 pluralistic alignment and personalized policies with explicit user 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 05

Synthetic data and curriculum engineering

Generate useful data without amplifying model errors and artifacts.

Chapter operating position

Generate useful data without amplifying model errors and artifacts.

5.1 Generation pipelines

Generation pipelines must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use seed tasks, templates, models, tools, simulators, retrieval, and self-play to create 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 generation pipelines.
- 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 generation pipelines 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 generation pipelines.
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 Filtering and verification

Filtering and verification must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Apply schemas, deterministic checks, execution, experts, judges, diversity, novelty, and confidence. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 filtering and verification.
- 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

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

- Treating filtering and verification 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 filtering and verification.
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 Curriculum and difficulty

Curriculum and difficulty must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Sequence concepts, task complexity, counterexamples, rare cases, and recovery behavior. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 curriculum and difficulty.
- 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 curriculum and difficulty 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 curriculum and difficulty.
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 Feedback loops and contamination

Feedback loops and contamination must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Prevent self-reinforcing errors, benchmark leakage, homogenization, synthetic signatures, and hidden provenance. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 feedback loops and contamination.

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

Chapter 05 laboratory and review

Practical laboratory

Generate a synthetic training set with deterministic validation and measure diversity, error rate, leakage, and marginal value.

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 simulator-generated trajectories and adversarial co-evolution of data and models.

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

Training systems, efficiency, and checkpoints

Operate training as a reliable distributed computation.

Chapter operating position

Operate training as a reliable distributed computation.

6.1 Data and model parallelism

Data and model parallelism must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Partition batches, parameters, gradients, optimizer state, experts, and pipeline stages. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 model parallelism.
- 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 model parallelism 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 model parallelism.
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 Memory and precision

Memory and precision must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use mixed precision, checkpointing, offload, sharding, accumulation, and sequence packing. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 memory and precision.
- 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

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

- Treating memory and precision 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 memory and precision.
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 Checkpoint and resume

Checkpoint and resume must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Record model, optimizer, scheduler, scaler, RNG, data position, code, config, and hardware state. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 resume.
- 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 resume 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 resume.
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 Failure and cost control

Failure and cost control must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Handle worker loss, NaNs, divergence, data errors, quota, preemption, and budget. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 failure and cost control.

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 failure and cost control 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 failure and cost control.
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

Run a tiny training job with checkpoint, interruption, deterministic resume, and artifact comparison.

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 elastic training, distributed checkpoint fabrics, and energy-aware scheduling.

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

Evaluation, safety, and release qualification

Prove that adaptation improved the intended behavior without unacceptable regression.

Chapter operating position

Prove that adaptation improved the intended behavior without unacceptable regression.

7.1 Capability and slice evaluation

Capability and slice evaluation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Measure target task, general capability, languages, formats, domains, and user populations. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 capability and slice 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 capability and slice 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 capability and slice 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.

7.2 Safety and robustness

Safety and robustness must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Test harmful use, prompt injection, privacy, memorization, refusal, over-compliance, and distribution shift. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 safety and robustness.

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

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

Treating safety and robustness 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 safety and robustness.
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 Base versus tuned comparison

Base versus tuned comparison must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use paired outputs, blinded review, deterministic tests, significance, and critical regressions. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 base versus tuned comparison.

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 base versus tuned comparison 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 base versus tuned comparison.
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 Model card and release evidence

Model card and release evidence must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Publish data, method, hyperparameters, compute, evaluation, limitations, license, risks, and intended use. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 model card and release evidence.

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 model card and release evidence 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 model card and release evidence.
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
Create a release report comparing base and tuned models with critical-failure gates and rollback criteria.

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 continuous post-training evaluation and automatically generated assurance cases.

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

Lifecycle, governance, and continual adaptation

Control adapters and model variants after deployment.

Chapter operating position

Control adapters and model variants after deployment.

8.1 Variant registry

Variant registry must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Track base, adapter, tokenizer, template, data, code, evaluation, compatibility, owner, and status. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 variant registry.
- 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 variant registry 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 variant registry.
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 Monitoring and drift

Monitoring and drift must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Observe task quality, user feedback, safety, data change, tools, retrieval, latency, and incidents. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 monitoring and drift.
- Separate model behavior from retrieval, tools, memory, policy, interface, and human oversight.
- Version prompts, schemas, data, model artifacts, runtime configuration, evaluations, and release evidence.
- Test nominal, negative, adversarial, long-context, degraded, overloaded, recovery, and rollback conditions.
- Retain source, trajectory, tool, citation, metric, profile, decision, and user-outcome evidence.
- Prefer least privilege, typed contracts, bounded budgets, staged rollout, abstention, and reversibility.

Implementation sketch

```
training_run:
  base_model_digest: sha256:...
  dataset_manifest: data/v3/manifest.json
  method: lora
  targets: [q_proj, v_proj]
  seed: 42
  checkpoints:
    include: [model, optimizer, scheduler, rng, data_position]
  release_gates:
    - target-task-improved
    - safety-regression-zero
    - heldout-contamination-check-pass
```

Failure patterns

- Treating monitoring and drift as a model capability claim disconnected from complete system behavior.

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

Validation and evidence

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

8.3 Update and rollback

Update and rollback must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use staged rollout, canaries, A/B tests, adapter switch, model fallback, and retained prior artifacts. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 update and rollback.
- 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 update and rollback 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 update and rollback.
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 Retirement and data obligations

Retirement and data obligations must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Remove unsupported variants, revoke access, archive evidence, respect deletion, and update downstream users. 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 objective, dataset, tokenizer, model, optimizer, checkpoint, evaluation, release, deployment, and monitoring 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 retirement and data obligations.

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 retirement and data obligations 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 retirement and data obligations.
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
Design a registry and rollout process for multiple adapters sharing one base model across teams.

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 online adaptation with bounded memory, user-specific adapters, and verifiable unlearning.

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. Hugging Face - Transformers Documentation
Role: Model architecture, training, inference, and multimodal framework documentation.
Verified: 2026-07-26
Official source: https://huggingface.co/docs/transformers/index

02. Hugging Face - PEFT Documentation
Role: Parameter-efficient fine-tuning framework and methods.
Verified: 2026-07-26
Official source: https://huggingface.co/docs/peft/index

03. Hugging Face - TRL Documentation
Role: Post-training and alignment framework documentation.
Verified: 2026-07-26
Official source: https://huggingface.co/docs/trl/index

04. Microsoft Research - LoRA: Low-Rank Adaptation of Large Language Models
Role: Low-rank parameter-efficient adaptation.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2106.09685

05. University of Washington - QLoRA: Efficient Finetuning of Quantized LLMs
Role: Quantized parameter-efficient fine-tuning.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2305.14314

06. Stanford University - Direct Preference Optimization
Role: Preference optimization without an explicit reward-model training loop.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2305.18290

07. 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

08. NIST - SP 800-218A - Secure Software Development Practices for Generative AI and Dual-Use Foundation Models
Role: AI-specific secure development profile.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/218/a/final

09. NIST - Artificial Intelligence Risk Management Framework 1.0
Role: Govern, Map, Measure, and Manage framework for trustworthy AI risk.
Verified: 2026-07-26
Official source: https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

Authority application and refresh triggers

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

Limitations, freshness, and revision control

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

Revision record

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

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SEC; MYTHOS-SWE
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	fine tuning, alignment, LoRA, QLoRA, DPO, synthetic data, SFT, training governance, preference optimization, model release
Canonical route	/library/field-guides/mythos-fg-ai-0005/

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