CANONICAL LIVING OVERVIEW GUIDE

AI Model Architecture and Capability Foundations

A foundation-to-frontier guide covering tensors, representation learning, transformers, attention, encoders, decoders, mixture-of-experts, diffusion, state-space and recurrent alternatives, multimodality, scaling, capability boundaries, and architecture evaluation.

PERMANENT ID
MYTHOS-FG-AI-0001

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

AI Model Architecture and Capability Foundations

Document ID
MYTHOS-FG-AI-0001

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

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

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

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-SWE; MYTHOS-DAT; MYTHOS-EMT
Audience	AI engineers, software architects, technical leaders, evaluators, infrastructure engineers, and advanced practitioners.
Prerequisites	Linear algebra, probability, software engineering, data structures, and basic machine-learning concepts.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Explain model computation from tensors and embeddings through attention, layers, logits, and decoding.

Differentiate encoder, decoder, encoder-decoder, diffusion, MoE, recurrent, and multimodal architectures.

Connect architecture choices to capability, context, latency, memory, training, and failure behavior.

Identify the difference between model capability and complete AI-system capability.

Evaluate architecture claims through controlled tasks, ablations, profiles, and limitations.

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

Mathematical and computational foundations

Build the minimum model of tensor computation, optimization, and representation.

Chapter operating position
Build the minimum model of tensor computation, optimization, and representation.

1.1 Tensors, shapes, and linear transforms

Tensors, shapes, and linear transforms must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Trace batches, sequence, hidden dimensions, heads, matrices, broadcasting, and device placement. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 tensors, shapes, and linear transforms.
- 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 tensors, shapes, and linear transforms 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 tensors, shapes, and linear transforms.
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 Embeddings and representations

Embeddings and representations must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Understand token, position, segment, modality, and learned representation spaces. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 embeddings and representations.
- 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

Treating embeddings and representations 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 embeddings and representations.
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 Probability and logits

Probability and logits must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Relate logits, softmax, likelihood, entropy, calibration, and sampling. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 probability and logits.

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 probability and logits 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 probability and logits.
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 Optimization and gradients

Optimization and gradients must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Explain loss, backpropagation, optimizers, schedules, regularization, precision, and distributed updates. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 optimization and gradients.

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 optimization and gradients 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 optimization and gradients.
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
Implement a small tensor classifier and inspect shapes, gradients, calibration, and failure under distribution shift.

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 low-precision training, learned optimizers, and alternative computational substrates.

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

Transformer and attention architecture

Understand the dominant sequence-model architecture at implementation level.

Chapter operating position
Understand the dominant sequence-model architecture at implementation level.

2.1 Self-attention

Self-attention must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Trace query, key, value projection, scores, masking, normalization, weighted values, and head composition. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 self-attention.
- 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 self-attention 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 self-attention.
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 Feed-forward and normalization blocks

Feed-forward and normalization blocks must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Understand residual streams, normalization placement, activations, gating, and depth. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 feed-forward and normalization blocks.
- 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

- Treating feed-forward and normalization blocks 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 feed-forward and normalization blocks.
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 Position and sequence representation

Position and sequence representation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Compare absolute, relative, rotary, bias-based, and recurrent context mechanisms. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

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

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

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for position and sequence representation.

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

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

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

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

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

Failure patterns

Treating position and sequence representation as a model capability claim disconnected from complete system behavior.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for position and sequence representation.
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 Efficient attention variants

Efficient attention variants must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Evaluate grouped-query, multi-query, sparse, sliding-window, linear, and IO-aware attention. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 efficient attention variants.

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 efficient attention variants 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 efficient attention variants.
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
Build a tiny transformer and compare full, causal, and local attention masks with profiling.

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 adaptive attention, recurrent memory, state-space models, and learned compute allocation.

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

Encoder, decoder, and sequence-to-sequence families

Match model family to representation, generation, and transformation tasks.

Chapter operating position
Match model family to representation, generation, and transformation tasks.

3.1 Encoder-only models

Encoder-only models must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use bidirectional context for classification, retrieval, extraction, and embedding. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 encoder-only models.
- 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 encoder-only models 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 encoder-only models.
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 Decoder-only models

Decoder-only models must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use causal prediction for generation, in-context learning, tools, and autoregressive multimodality. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 decoder-only models.
- 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

Treating decoder-only models 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 decoder-only models.
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 Encoder-decoder models

Encoder-decoder models must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Separate source representation from conditional generation for translation and transformation. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 encoder-decoder models.

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 encoder-decoder models 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 encoder-decoder models.
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 Prefix, infill, and structured generation

Prefix, infill, and structured generation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Support code completion, editing, constrained output, and bidirectional context 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 prefix, infill, and structured generation.

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

Chapter 03 laboratory and review

Practical laboratory
Compare tiny encoder, decoder, and encoder-decoder models on classification and sequence transformation.

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 universal architectures that dynamically select encoding and generation modes.

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

Mixture-of-experts and conditional computation

Understand sparse capacity, routing, and operational tradeoffs.

Chapter operating position
Understand sparse capacity, routing, and operational tradeoffs.

4.1 Experts and routers

Experts and routers must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Trace token routing, top-k selection, expert computation, residual paths, and output combination. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 experts and routers.
- 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 experts and routers 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 experts and routers.
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 Load balancing and specialization

Load balancing and specialization must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Control expert collapse, capacity, dropped tokens, auxiliary losses, and specialization. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 load balancing and specialization.
- 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

Treating load balancing and specialization 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 load balancing and specialization.
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 Distributed execution

Distributed execution must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Place experts across devices, manage all-to-all communication, memory, latency, and failures. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 distributed execution.

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 distributed execution 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 distributed execution.
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 Capability and evaluation

Capability and evaluation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Distinguish parameter count, active parameters, routing quality, and domain performance. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 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 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 evaluation.
Observe	Requests, selected context, trajectories, tool calls, model outputs, citations, caches, telemetry, resources, and user outcomes.
Test	Expected, prohibited, adversarial, malformed, long-context, overloaded, failed, recovery, and rollback cases.
Measure	Task quality, calibration, grounding, safety, latency, throughput, memory, tokens, cost, energy, and operator effort.
Reconcile	Intended system behavior against evaluated model, data, runtime, tools, sources, deployment, and residual risk.

Chapter 04 laboratory and review

Practical laboratory
Implement a small routed expert layer and measure balance, specialization, latency, and failure when one expert is unavailable.

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 hierarchical experts, dynamic depth, modular skill composition, and expert marketplaces.

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

Generative architectures beyond text transformers

Connect diffusion, autoregressive, masked, and latent generative systems.

Chapter operating position
Connect diffusion, autoregressive, masked, and latent generative systems.

5.1 Diffusion and denoising

Diffusion and denoising must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Understand forward corruption, reverse denoising, score prediction, schedules, guidance, and sampling. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 diffusion and denoising.
- 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 diffusion and denoising 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 diffusion and denoising.
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 Autoregressive media generation

Autoregressive media generation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Generate image, audio, video, or actions as discrete or continuous sequences. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 autoregressive media generation.
- 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

Treating autoregressive media generation 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 autoregressive media generation.
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 Latent representations

Latent representations must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Use autoencoders, tokenizers, compressed spaces, and modality-specific decoders. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 latent representations.

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 latent representations 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 latent representations.
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 Consistency, flow, and accelerated sampling

Consistency, flow, and accelerated sampling must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Evaluate fewer-step generation, distillation, flow matching, and quality tradeoffs. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

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

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

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for consistency, flow, and accelerated sampling.

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

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

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

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

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

Failure patterns

Treating consistency, flow, and accelerated sampling as a model capability claim disconnected from complete system behavior.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Requirements, authority, model, data, prompts, tools, memory, schemas, versions, and assumptions for consistency, flow, and accelerated sampling.
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

Implement a one-dimensional diffusion toy model and compare sampling steps, noise schedules, and reconstruction error.

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 unified world models, continuous-time models, and multimodal generative simulators.

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

Multimodal and embodied architectures

Fuse text, image, audio, video, sensor, and action representations.

Chapter operating position
Fuse text, image, audio, video, sensor, and action representations.

6.1 Modality encoders and tokenization

Modality encoders and tokenization must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Represent patches, frames, spectrograms, waveforms, sensors, and actions. The implementation should name authoritative inputs, data and model versions, identities, state transitions, permissions, resource limits, telemetry, failure behavior, uncertainty, rollback, and acceptance criteria.

This guide traces the subject through the input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 modality encoders and tokenization.
- 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 modality encoders and tokenization 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 modality encoders and tokenization.
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 Fusion and cross-attention

Fusion and cross-attention must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Compare early, late, shared-space, cross-attention, and adapter-based fusion. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 fusion and cross-attention.
- 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

Treating fusion and cross-attention 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 fusion and cross-attention.
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 Grounding and temporal understanding

Grounding and temporal understanding must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Link language to regions, timestamps, speakers, objects, actions, and environment 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

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

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

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for grounding and temporal understanding.

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

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

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

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

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

Failure patterns

Treating grounding and temporal understanding as a model capability claim disconnected from complete system behavior.

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

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

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

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

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

Validation and evidence

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

6.4 Embodied and world-model loops

Embodied and world-model loops must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Connect perception, memory, planning, control, feedback, and safety. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 embodied and world-model loops.

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 embodied and world-model loops 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 embodied and world-model loops.
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
Build a small paired image-text embedding system and evaluate retrieval, hard negatives, and calibration.

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 realtime world models, spatial intelligence, and physical AI foundation 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 07

Scaling, emergence, and capability boundaries

Interpret scaling evidence without treating model size as a complete capability measure.

Chapter operating position
Interpret scaling evidence without treating model size as a complete capability measure.

7.1 Scaling variables

Scaling variables must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Relate parameters, data, compute, tokens, architecture, optimizer, precision, and training quality. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 scaling variables.
- 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 scaling variables 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 scaling variables.
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 In-context learning and adaptation

In-context learning and adaptation must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Distinguish memorization, pattern induction, retrieval, reasoning, and tool-mediated 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 in-context learning and 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

- Treating in-context learning and 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 in-context learning and 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.

7.3 Capability discontinuities and measurement

Capability discontinuities and measurement must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Investigate threshold effects, benchmark artifacts, contamination, prompting, 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 discontinuities and measurement.

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 discontinuities and measurement 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 discontinuities and measurement.
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 System capability versus model capability

System capability versus model capability must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Separate weights from retrieval, tools, memory, orchestration, policy, and user 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation chain. A fluent response, high aggregate score, successful tool call, or green service dashboard is not sufficient proof. Intended behavior must be reconciled with hidden tests, source evidence, negative and adversarial cases, latency and resource measurements, user-visible results, and residual limitations.

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

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

Engineering practices

Define the objective, authority, data, model, owner, constraints, risks, and acceptance criteria for system capability versus model capability.

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

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

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

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

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

Failure patterns

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

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

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

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

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

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

Validation and evidence

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

Chapter 07 laboratory and review

Practical laboratory
Run scaling experiments on small models and identify which apparent capability gains are data, prompt, or metric artifacts.

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 test-time compute, adaptive reasoning, modular networks, and self-improving systems.

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

Architecture selection and assurance

Choose model architecture according to workload, evidence, and constraints.

Chapter operating position
Choose model architecture according to workload, evidence, and constraints.

8.1 Workload and capability matrix

Workload and capability matrix must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Map tasks, modalities, context, latency, throughput, privacy, adaptation, and deployment. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 workload and capability matrix.
- 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 workload and capability matrix 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 workload and capability matrix.
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 Resource and lifecycle model

Resource and lifecycle model must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Estimate training, inference, memory, storage, networking, updates, evaluation, and operations. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 resource and lifecycle model.
- 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

```
class TinyAttention(torch.nn.Module):
    def __init__(self, width: int, heads: int):
        super().__init__()
        assert width % heads == 0
        self.heads = heads
        self.qkv = torch.nn.Linear(width, width * 3)
        self.out = torch.nn.Linear(width, width)

    def forward(self, x, causal_mask):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # reshape -> attention -> merge; see reference project
        return self.out(attend(q, k, v, causal_mask, self.heads))
```

Failure patterns

Treating resource and lifecycle model 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 resource and lifecycle model.
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 Risk and failure analysis

Risk and failure analysis must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Assess hallucination, bias, brittleness, privacy, security, misuse, uncertainty, and unsafe autonomy. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 risk and failure analysis.

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 risk and failure analysis 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 risk and failure analysis.
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 Architecture decision record

Architecture decision record must be treated as an observable AI-system mechanism rather than a model label, benchmark claim, prompt trick, or framework feature. Document alternatives, evidence, assumptions, limitations, selection, review triggers, and exit. 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 input representation, model block, state, output distribution, decoding, system integration, evaluation, and limitation 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 architecture decision record.

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

Chapter 08 laboratory and review

Practical laboratory

Create an architecture decision for an enterprise assistant using at least four model families and explicit system components.

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 architecture search guided by safety, energy, sovereignty, and measurable system outcomes.

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. Google Research - Attention Is All You Need
Role: Transformer architecture foundation.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/1706.03762

02. Google Research - BERT: Pre-training of Deep Bidirectional Transformers
Role: Encoder pretraining and transfer-learning foundation.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/1810.04805

03. Google Research - Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer
Role: Text-to-text architecture and scaling study.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/1910.10683

04. Google Research - Switch Transformers
Role: Sparse mixture-of-experts architecture and routing.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2101.03961

05. Google Research - An Image is Worth 16x16 Words
Role: Vision Transformer foundation.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2010.11929

06. OpenAI Research - Learning Transferable Visual Models From Natural Language Supervision
Role: Contrastive image-text representation foundation.
Verified: 2026-07-26
Official source: https://arxiv.org/abs/2103.00020

07. PyTorch - PyTorch Documentation
Role: Official tensor, autograd, neural-network, distributed, and compiler documentation.
Verified: 2026-07-26
Official source: https://pytorch.org/docs/stable/index.html

08. 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

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-SWE; MYTHOS-DAT; MYTHOS-EMT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	AI architecture, transformers, attention, mixture of experts, diffusion, multimodal, embeddings, scaling, world models, model selection
Canonical route	/library/field-guides/mythos-fg-ai-0001/

Living publication

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