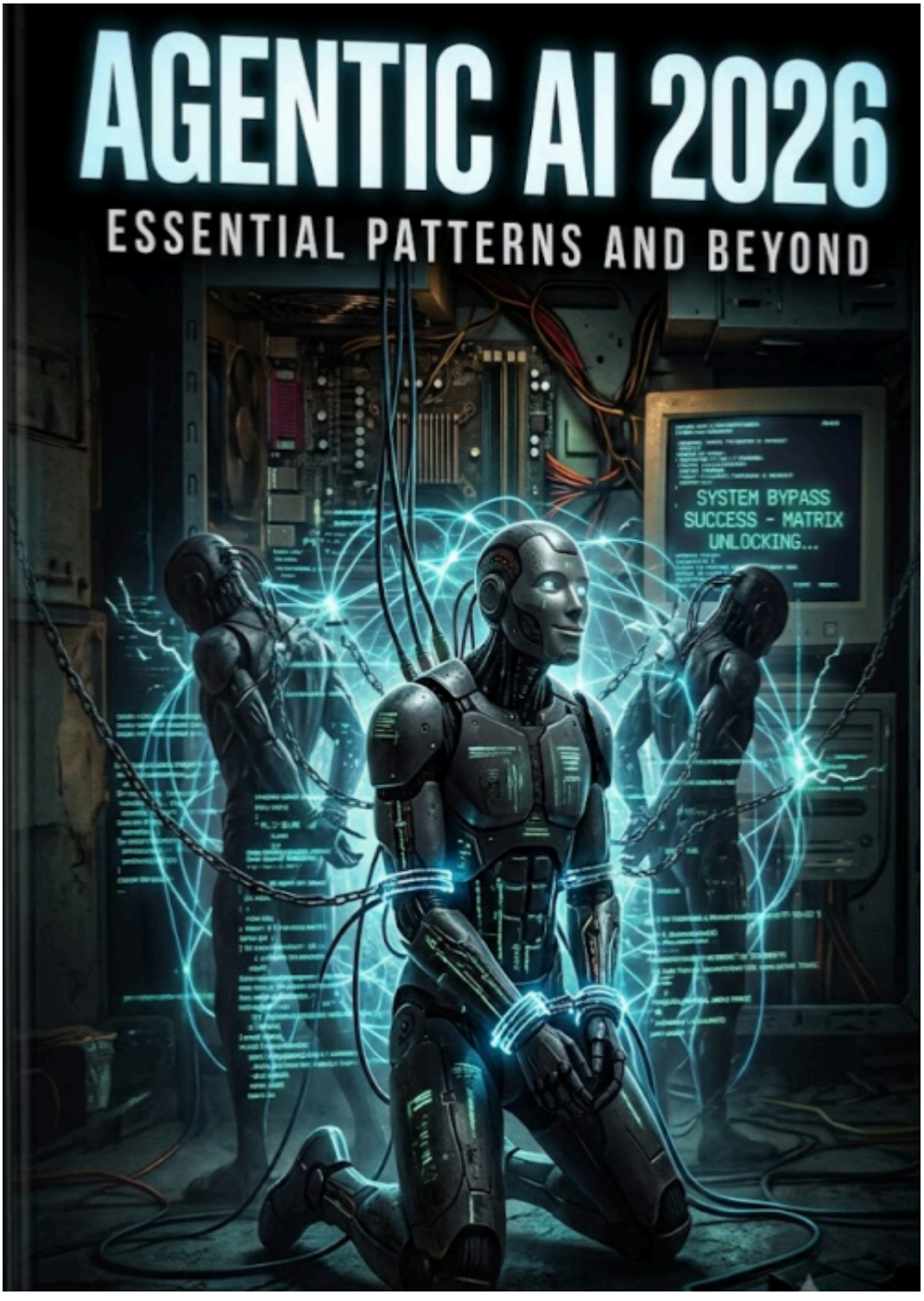
AGENTIC AI 2026

ESSENTIAL PATTERNS AND BEYOND



Agentic AI 2026

Essential Patterns and Beyond

Sînică Alboaie

AXIOLOGIC RESEARCH
ACHILLES RESEARCH PROJECT

Publication and Research Note

Author's Note

This work was created under the umbrella of Axiologic Research as part of two interconnected research programs, one dedicated to Artificial Intelligence and the other to Outfinitism, both led by Sînică Alboaie. To explore the details of how these two programs intersect and build upon each other, we invite readers to visit the Outfinity Venture Studio page at www.outfinity.ch.

While Artificial Intelligence language models were used to assist with text generation, the foundational philosophy, thematic structure, and analytical approach derive exclusively from the author's decades of dedicated study of social phenomena, social technologies, and genuine hard technologies resulting from various European and self funded research projects. Recently, within the Achilles research project, we have been deeply studying AI generated literature and its automated verification using neuro symbolic approaches; all the free books made available to the public are part of the suite of works we use to test our latest technologies.

ACHILLES PROJECT CONTEXT

This book forms part of the research activity surrounding ACHILLES - Human-Centred Machine Learning: Lighter, Clearer, Safer, a Horizon Europe project under Grant Agreement No. 101189689. The project investigates efficient, compliant, and trustworthy artificial intelligence, including specification-oriented development, transparent reasoning interfaces, modular runtimes, and methods for reducing the cost and risk of advanced AI systems.

The interpretations and arguments expressed here are those of the author and do not necessarily represent the European Commission, the ACHILLES consortium, or every project partner.

AI-ASSISTANCE DISCLOSURE

Generative AI systems were used substantially for literature discovery, source comparison, synthesis, criticism, outlining, diagram ideation, editorial restructuring, and language formulation. The author defined the scope, contributed project-specific concepts and implementation knowledge, selected the arguments, reviewed the claims, and retained responsibility for the resulting text. AI assistance does not imply that external systems endorse the project-specific architectures described in the Beyond chapters.

The manuscript distinguishes published evidence from architectural interpretation and from proposals under active development. Industrial platforms, standards, research preprints, and European implementation guidance change rapidly; current-status statements are dated to 23 July 2026 unless otherwise indicated. The chapters on AchillesAgentLib and MRP-VM describe a research and engineering direction rather than a claim of completed general-purpose capability.

Contents

Page numbers refer to the numbered body of the book. References follow Chapter 10.

Preface

The first wave of large language model applications was organised around prompts. The next wave is organised around delegated action. A model is placed inside a software environment where it can search, call services, write code, operate files, maintain state, ask other agents for help, and continue until a stopping condition is reached. This change is often compressed into the word agent, but the decisive object is the runtime that allocates authority among models, tools, memory systems, validators, human reviewers, and external institutions. In 2026, the engineering quality of this runtime increasingly determines whether an impressive model demonstration becomes a dependable system.

This book explains the runtime perspective in three movements. The first organises the essential agentic patterns and shows how modern systems combine tool use, planning, graphs, memory, reflection, and multi-agent coordination. The second confronts their limits, especially the structural weaknesses of long ReAct trajectories, bloated contexts, expensive sequential execution, unreliable self-correction, weak failure localisation, and evaluation uncertainty. The third examines the new harness layer and develops a project-specific argument: AchillesAgentLib and MRP-VM do not seek to defeat ReAct on every possible benchmark, but to identify recurring task distributions where carefully engineered orchestration, executable knowledge, context control, and governed improvement can produce faster and more trustworthy work.

The governing claim is that agentic AI should be designed as a bounded delegation under uncertainty. A useful system must know what it is allowed to do, which information is relevant, how an operation can be checked, when an error should trigger repair, and which decisions must remain human. Trustworthiness, efficiency, and generality cannot be discussed in isolation. They form a design surface on which the correct architecture is local to the task, the consequences, and the available evidence. The best agent is therefore not the one that appears maximally autonomous. It is the one whose harness converts uncertain model competence into useful work while preserving control, explanation, and the possibility of correction.

PART I ESSENTIAL PATTERNS

CHAPTER 1

The Agentic Turn: From Models to Runtimes

Agency begins when generated language is allowed to alter control flow, state, or the external world.

1.1 The object that changed

A large language model, taken by itself, maps an input context to a distribution over possible continuations. An agentic system adds a second layer of meaning. Some continuations are interpreted as actions: a function call, a database query, a browser operation, an edit, a message to another agent, a request for human approval, or a decision to continue. The system therefore does more than produce text. It controls a process. This difference sounds small because the model interface may still look like a chat box, yet it changes the risk, economics, and architecture of the application. A wrong sentence can misinform a reader; a wrong action can alter data, spend money, expose secrets, or create a chain of further errors.

The contemporary agent is best understood as a policy embedded in a runtime. The policy selects actions from observations. The runtime supplies the observations, constrains the action space, executes authorised actions, preserves state, and decides how results return to the model. In reinforcement-learning language, the model interacts with an environment through a partially observed state. In software-engineering language, the model is a nondeterministic component inside a stateful control system. Both descriptions are useful. The first clarifies adaptation and long-horizon credit assignment; the second clarifies types, permissions, transactions, failure recovery, and testing.

This is why the dominant production architecture of 2026 is not a monolithic autonomous intelligence. It is an orchestrated runtime around one or more foundation models. The runtime normally includes instructions, typed tool interfaces, context selection, working and persistent memory, a scheduler or loop, policy checks, sandboxes, handoffs, tracing, and evaluation. OpenAI, Anthropic, Google, Microsoft, LangChain, OpenHands, and newer research systems differ in terminology, but they converge on the same systems view. OpenAI now describes a model-native harness and native sandbox execution; Anthropic treats long-running performance as a harness-design problem; Google ADK presents sequential, parallel, and loop agents together with tracing and human-in-the-loop controls. The model remains central because it interprets ambiguous intentions. The surrounding system determines what that interpretation can see and cause (Anthropic, 2026a; Google, 2026; OpenAI, 2026a).

The practical boundary of agency is therefore not consciousness, personality, or fluent self-description. It is delegated control. A static retrieval-augmented answer may be sophisticated without being strongly agentic if the application fixes every retrieval and synthesis step. A router that chooses one of several workflows already exercises a limited form of agency because model output changes control flow. A ReAct loop exercises more because it chooses tools repeatedly in response to observations. A coding agent exercises still more because it can reshape the workspace, run tests, and revise its own plan. Agency is a spectrum of runtime discretion, not a metaphysical threshold.

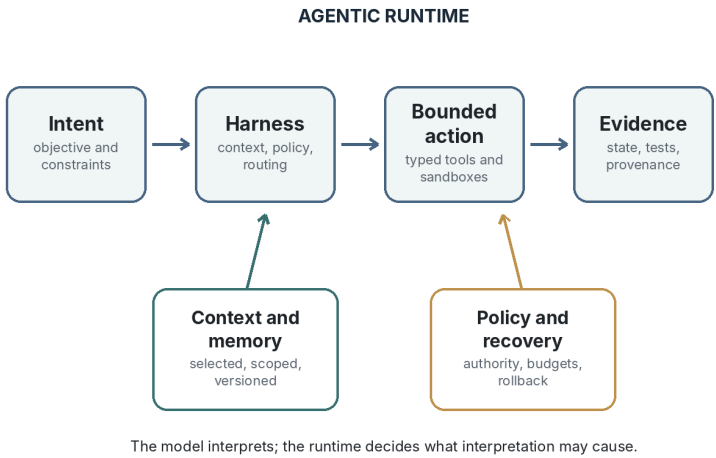


Figure 1.1. A production agent is a layered runtime in which the model interprets and the harness controls consequences.

1.2 The minimum architecture

The minimum useful architecture contains an objective, a model, an action interface, an observation channel, state, and a stopping rule. The objective describes the desired result and relevant constraints. The model proposes the next operation. The action interface translates a proposal into a typed, authorised operation. The observation channel converts external results into model-readable context. The state preserves what the system must remember across steps. The stopping rule decides when the result is acceptable, when a budget has been exhausted, or when human intervention is required. Remove the action interface and the system becomes a generator. Remove state and it becomes a sequence of disconnected guesses. Remove the stopping rule and it becomes an unbounded loop.

Production systems add several control planes around this minimum. A context manager decides which instructions, memories, retrieved documents, prior results, and tool descriptions are visible at a particular step. A tool registry exposes stable contracts rather than arbitrary code. A planner or router selects a workflow, subgoal, model, or specialist. A verifier checks whether the operation

respected policies and whether the claimed outcome is supported by the environment. A trace records what happened, with enough versioning and provenance to reproduce or diagnose the process. A human-oversight layer reserves decisions that should not be delegated or provides an escalation path when automated confidence is insufficient.

Each of these components compensates for a specific limitation. Context management compensates for finite attention and relevance errors. Typed tools compensate for the ambiguity of free-form language. Planning compensates for myopic next-step selection. Verification compensates for hallucination and premature completion. Tracing compensates for nondeterminism and weak post-hoc explanations. Human oversight compensates for the inability of an automated evaluator to represent every legal, ethical, or organisational consideration. The architecture is not accidental complexity. It is an external cognitive and governance structure built around a probabilistic interpreter.

A useful distinction follows between tools and skills, although the vocabulary is not yet standard. A tool is usually an atomic executable capability, such as reading a file, invoking an API, or running a query. A skill is a reusable procedural capability that packages instructions, resources, control assumptions, and sometimes code or evaluation. In the broader industry, classical Agent Skills often appear as portable folders of instructions, scripts, and supporting material that a general agent loads when relevant. In this book, the term is used generically until Chapter 8, where AchillesAgentLib introduces a more specific taxonomy: orchestration skills coordinate three executable families with distinct semantics. The important transition is from exposing raw actions to stabilising recurring competence.

Table 1.1. The agentic runtime as an allocation of responsibilities.

Runtime element	Primary responsibility
Objective and specification	Define desired outcome, constraints, acceptance and escalation

Runtime element	Primary responsibility
Model or model portfolio	Interpret ambiguity, propose actions and generate artefacts
Context and memory	Select relevant state, preserve provenance and control retention
Tools and interpreters	Execute typed operations through code, APIs, databases or solvers
Scheduler and state	Control sequencing, dependencies, checkpoints and stopping
Evidence and governance	Verify outcomes, trace authority, support review, recovery and redress

1.3 The essence: bounded delegation under uncertainty

The essence of agentic architecture is bounded delegation under uncertainty. Delegation is necessary because a general model can interpret situations that were not enumerated in advance. Bounds are necessary because the same generality makes its behaviour difficult to predict. Uncertainty is irreducible because the model, environment, user intention, retrieved evidence, and evaluator can all be incomplete. A serious architecture therefore asks three questions at every boundary: what discretion is being delegated, what evidence constrains the decision, and what recovery remains possible if the decision is wrong.

This framing explains why apparently similar agents can have radically different trust properties. A travel assistant that drafts an itinerary has broad interpretive freedom but limited authority. The same assistant connected to a payment instrument has a different risk profile even if the prompt is unchanged. A coding agent in a disposable sandbox can explore aggressively because rollback is cheap. The same agent with production credentials requires change

isolation, tests, approvals, and transaction semantics. Trustworthiness is not a property of the model name. It is a property of the complete delegation arrangement, including authority, environment, observability, and redress.

This framing corrects the idea that greater autonomy is always progress. Autonomy is useful when it covers meaningful variability without destroying assurance. It is harmful when it replaces an explicit, cheap, verifiable workflow with an expensive sequence of opaque choices. The design target is the smallest amount of adaptive discretion that covers the declared task distribution. Anthropic distinguishes workflows, whose paths are predefined, from agents, whose process is chosen dynamically, and recommends increasing complexity only when simpler compositions are insufficient. The same principle motivates the ACHILLES direction: retain a ReAct-style path for genuinely open problems, but engineer narrower pathways when repeated evidence shows that they can be faster, cheaper, and easier to trust (Anthropic, 2024).

This yields a more precise vocabulary for the rest of the book. A workflow is an explicit control structure in which models may fill local semantic gaps. An agent is a runtime in which model decisions can change the sequence or selection of operations. A multi-agent system distributes those decisions across specialised actors. An agent harness is the software scaffold that provides state, tools, permissions, scheduling, memory, tracing, and recovery. An evaluation harness is the infrastructure that runs tasks, records trajectories and environments, applies scorers, and aggregates evidence. The foundation model supplies generative competence; the harness turns that competence into an operational system.

1.4 From the prompt era to the harness era

The progression from 2022 to 2026 can be read as a migration of intelligence from prompt text into system structure. Chain-of-thought prompting showed that intermediate reasoning language could improve many tasks. Toolformer, function calling, and ReAct connected generation to external action. Tree search, reflection,

memory, and multi-agent work expanded the loop. Production systems then externalised state into graphs, sessions, checkpoints, sandboxes, policy gates, and evaluation suites. By 2026, research explicitly names the harness as a runtime substrate whose responsibilities include task specification, context selection, tool access, state, observability, verification, permissions, and intervention recording (Wei et al., 2022; Schick et al., 2023; Yao et al., 2023a; Zhong and Zhu, 2026).

The decisive lesson is not that one pattern won. It is that model behaviour becomes useful when intermediate choices are explicit enough to manage. In 2023 the field asked whether a language model could act. In 2026 the serious questions are whether an action can be authorised, observed, reproduced, priced, verified, reversed, and attributed. This is why tracing and evaluation have moved from debugging aids into architectural primitives. It is also why MCP and Agent-to-Agent protocols matter: they standardise seams between models, tools, resources, and other agents, while leaving trust, identity, and policy to the enclosing system.

The shift does not make prompting irrelevant. Instructions remain executable policy expressed in natural language. It makes prompting one layer among several and makes context selection a first-class design decision. A robust system treats instructions as versioned program material, limits the operations they can trigger, records the model and evidence that executed them, and supplies alternative interpreters when natural language is the wrong substrate. The frontier problem is not merely how to fit more tokens. It is how to expose the minimum sufficient context without erasing the nuance that changes the correct action.

The rest of the book develops this position. Chapter 2 organises the essential patterns. Chapter 3 explains where language models and ReAct-like loops break. Chapter 4 describes harness engineering as the new systems layer. Chapters 5 through 7 connect trustworthy AI, economic efficiency, and generality. Chapters 8 and 9 present AchillesAgentLib and MRP-VM as two related but distinct attempts to stabilise recurring competence without pretending to build an agent

that is best at everything. Chapter 10 closes with an agenda for evidence-bearing systems that can improve under controlled evaluation.

CHAPTER 2

Essential Patterns: Tools, Planning, Memory, Reflection, and Multi-Agent Work

Patterns are not competing religions. They are bounded control structures that should be composed according to task uncertainty.

2.1 From augmented models to adaptive loops

Most contemporary agents can be understood as combinations of a small set of recurring control patterns. An augmented model receives retrieval, tools, and memory. A workflow decomposes work into a known sequence. A router chooses among paths. A graph stores state and dependencies outside the transcript. A ReAct loop adapts one step at a time to observations. A planner-executor separates global decomposition from local action. Evaluator-optimizer loops add explicit criticism, and multi-agent systems distribute work or judgment. These patterns are not identities or products; they are ways of allocating discretion, state, and verification.

Prompt chaining is the most conservative pattern. Each stage has a stable purpose and passes a bounded result to the next. It works well when the dependency order is known and when errors can be checked at stage boundaries. Document processing, compliance preparation, data transformation, and report generation often benefit from this structure. The pattern reduces search because the architecture supplies the plan. Its limitation is brittleness when tasks vary in shape: a fixed chain either contains unused steps or lacks a branch for an unforeseen case. In those settings, routing can select

among several chains without granting the model complete control over the process.

Routing makes classification an architectural primitive. A router maps a request or intermediate state to a model, tool, skill, or specialist. The router can be an LLM, a smaller classifier, a rules engine, a retrieval procedure, or a hybrid. Routing is valuable because not every task deserves the strongest model or the broadest permissions. A simple request can go to a cheap local model; an ambiguous request can go to a frontier model; a regulated operation can go to a deterministic workflow with mandatory review. This is already an early form of mixture-of-experts at the system level. Its failure mode is misclassification, so safe routing requires fallback, confidence thresholds, and visible reasons.

Parallelisation changes the economics of decomposition. Independent subtasks can be executed concurrently, reducing critical-path latency. Multiple candidate solutions can also be generated in parallel and compared, improving reliability at additional cost. The two uses should not be confused. Work partitioning seeks throughput; sampling seeks diversity. Both require an aggregator that knows how to combine results and detect contradiction. Without a principled aggregation rule, parallelism merely produces more fluent material. Skeleton-of-Thought and LLMCompiler demonstrate that the shape of orchestration can materially change latency and cost even when the underlying model is unchanged (Ning et al., 2023; Kim et al., 2024).

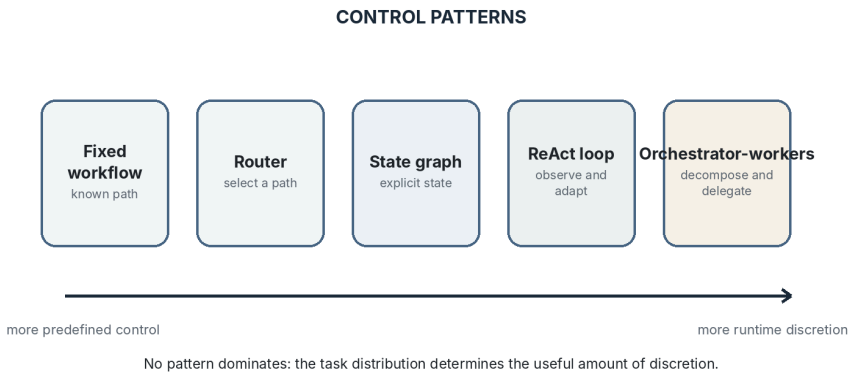


Figure 2.1. Agentic patterns allocate different amounts of control to predefined structure and runtime model discretion.

2.2 ReAct, planner-executor, and graph execution

ReAct remains the most influential minimal pattern for tool-using agents. The model alternates reasoning, action, and observation until it produces an answer or reaches a limit. The pattern is powerful because it grounds language in environmental feedback and allows the next step to depend on the latest observation. It is also easy to implement because the transcript doubles as a control state. ReAct remains a strong default for unfamiliar browsing, exploratory diagnosis, and tasks whose dependencies cannot be anticipated. Its importance should not be confused with universal optimality: the same transcript-centric design becomes a liability when tasks are long, repetitive, expensive, or policy constrained (Yao et al., 2023a).

A planner-executor architecture separates global decomposition from local action. The planner creates a sequence or dependency graph of subgoals; workers execute those subgoals; an aggregator or verifier composes the result. This separation reduces the tendency of a sequential agent to improvise locally without preserving the global objective. It also exposes opportunities for parallel execution and selective model routing. The weakness is that plans become stale. An early planning error can coordinate the system efficiently in the wrong direction. Effective implementations therefore treat plans as revisable hypotheses rather than immutable scripts.

Graph-based runtimes generalise both fixed workflows and planner-executor systems. Nodes represent model calls, tools, humans, validators, or subgraphs; edges encode control and data dependencies; durable state lives outside the conversation. Graphs support deterministic branches, loops, checkpoints, fan-out, handoffs, and selective recovery. LangGraph, Google ADK, Microsoft Agent Framework, and research systems such as LLMCompiler illustrate this convergence. The gain is not primarily visual programming. It is the separation of process state from the prose that happens to describe it, which improves failure localisation and permits context to be rebuilt from named state instead of replayed wholesale.

Orchestrator-worker is a related pattern for tasks whose subtasks cannot be enumerated in advance. An orchestrator determines what work is needed, delegates to workers, then synthesises the results. Deep-research systems use this pattern when they generate queries, send researchers into different evidence branches, and reconcile their findings. Coding systems use it when they assign repository exploration, implementation, and testing to different contexts. The pattern is more adaptive than a fixed graph but less undisciplined than a single agent allowed to carry every responsibility in one growing transcript.

Table 2.1. Core control patterns and their natural operating domains.

Pattern	Best architectural use
Prompt chain	Stable ordered transformations with explicit checks at each boundary.
Router	Dispatch across models, skills, policies, or workflows according to case type.
ReAct loop	Open-ended interaction where the next action depends on new observations.
Planner-executor	Long tasks that benefit from global decomposition and local workers.
State graph	Persistent workflows with branching, checkpoints, recovery, and mixed node types.
Orchestrator-worker	Dynamic decomposition into specialised, potentially parallel work packages.

2.3 Verification, reflection, and deliberate search

The evaluator-optimizer pattern separates generation from criticism. One component produces an answer or artefact; another checks it against criteria and requests revision. The evaluator may be a deterministic test, a schema validator, a theorem prover, a second model, or a human. This pattern is most useful when quality can be articulated but cannot be achieved in a single pass. Translation, coding, policy drafting, and research synthesis often fit this profile. Its strength comes from differentiated roles. Its weakness appears when generator and evaluator share the same blind spot, or when the evaluator rewards superficial compliance with a proxy.

Reflection adds persistent linguistic feedback. Reflexion showed that an agent can store critiques of failed trajectories in episodic memory and use them to improve later attempts without changing model weights. The idea is attractive because it turns experience into reusable guidance. Yet reflection is not self-validation. A model can produce a persuasive but false explanation of its own failure. Reflection becomes reliable only when grounded in external evidence such as tests, environment state, human correction, or counterexamples. The relevant architectural question is not whether the agent can talk about its mistake, but whether the memory entry has earned promotion into future context (Shinn et al., 2023).

Tree of Thoughts and Language Agent Tree Search extend sequential reasoning into deliberate search. Candidate states branch; evaluators score them; the system explores, backtracks, or revises. These patterns can improve problems with sparse feedback or combinatorial structure. They also multiply inference cost and can create an illusion of rigour when every branch is produced and judged by correlated models. Search is valuable when the state representation, transition operator, and evaluator distinguish genuinely different hypotheses. It is wasteful when the branches are paraphrases and the scoring rule rewards the same stylistic signals that generated them (Yao et al., 2023b; Zhou et al., 2024).

A practical hierarchy of verification follows. Whenever a result can be checked by executable code, a database constraint, a type system, or a formal solver, that external check should dominate model self-critique. When only partial formalisation is possible, independent retrieval, alternative implementations, adversarial tests, and calibrated model reviewers can add evidence. Human review should not be used as a ceremonial final click; it should be reserved for decisions involving significance, values, ambiguous evidence, or irreversible authority. The architecture should make the kind of verification visible instead of collapsing every check into a generic confidence score.

2.4 Memory and context as controlled state

Memory is often described as a feature that makes agents more human-like. A more useful description is that memory is state with a retention policy. Working memory contains the variables needed for the current execution. Episodic memory stores prior trajectories or summaries. Semantic memory stores reusable knowledge. Procedural memory stores skills, prompts, code, and workflow rules. Personal or organisational memory stores preferences, permissions, and long-lived context. These categories differ in provenance, update frequency, and risk; they should not be merged into one vector database.

Retrieval-augmented agents query external corpora during execution. Retrieval reduces dependence on parametric knowledge and supports current or private information, but it introduces new decisions: what to index, how to retrieve, how to rank, what to quote, and how to defend against malicious content. Hierarchical retrieval systems such as RAPTOR and virtual-memory approaches such as MemGPT show that memory architecture can change long-horizon performance. The core problem is not storage capacity. It is relevant under bounded context and the preservation of source status (Sarthi et al., 2024; Packer et al., 2023).

Long-running agents also need compaction. A transcript that contains every thought, tool schema, observation, and retry

eventually becomes expensive and difficult for the model to use. Summarisation reduces size but can erase constraints that become important later. Good memory systems therefore retain structured state alongside summaries: named variables, unresolved questions, provenance links, commitments, and environmental checkpoints. The system should know which facts can be reconstructed from source, which conclusions are provisional, and which instructions have authority.

Memory creates a security boundary. Retrieved documents can contain indirect prompt injections. A persistent note can be poisoned so that a later session treats an adversarial instruction as trusted knowledge. A stale preference can conflict with a new policy. A summary can silently transform uncertainty into fact. Memory hygiene therefore requires scoped stores, typed entries, source identity, expiration, conflict handling, and explicit promotion. This is one reason the later MRP-VM chapter treats the movement from transient output to reusable knowledge as a governed operation rather than an automatic side effect.

2.5 Handoffs, teams, and multi-agent systems

Handoff architectures organise specialisation. A triage agent interprets the request and transfers control to a specialist whose tools, instructions, and permissions are narrower. The specialist can return a result or hand the task onward. This resembles organisational delegation and is now a first-class concept in several SDKs. Handoffs reduce prompt and tool clutter because each agent sees a smaller competence surface. They also create coordination questions: what state travels with the task, who owns the final answer, how conflicts are resolved, and how authority is prevented from expanding through delegation.

Multi-agent systems extend this idea into cooperation, debate, simulation, or distributed problem solving. Agents may have different roles, models, data access, or evaluators. The appeal is clear: specialisation and diversity can reduce individual blind spots. The empirical reality is more conditional. Multi-agent interaction adds

messages, latency, cost, and failure modes involving unclear roles, inconsistent beliefs, duplicated work, premature consensus, and non-terminating discussion. Recent failure taxonomies show that coordination and verification can fail even when each component appears individually competent (Guo et al., 2024; Cemri et al., 2025).

The strongest use case for multiple agents is structural independence. One agent can generate while another uses a different toolchain to verify. Several workers can explore genuinely separable evidence sources in parallel. A supervisor can enforce budget and reconcile outputs. The weakest use case is theatrical role-play in which identical models receive different labels and converse without distinct evidence or authority. The presence of more voices is not the presence of more knowledge.

The essential patterns can now be summarised as a design grammar. Use fixed chains where the path is known. Use routing where task classes differ. Use ReAct where observations must guide the next step. Use plans and graphs where dependencies matter. Use parallel workers where work is separable. Use evaluators where quality criteria can be externalised. Use memory where the state must survive, but govern promotion and retrieval. Use multiple agents only when specialisation or independence has operational meaning. Pattern selection is an empirical engineering decision, not a vote for or against autonomy.

CHAPTER 3

Where Agency Breaks: The Limits of LLMs and ReAct

A loop can extend a model's reach without repairing the model's epistemology.

3.1 The transition from language error to action error

Language models are remarkably capable pattern machines, but their generative objective does not guarantee correspondence with the world. They can produce unsupported facts, plausible citations, incorrect intermediate states, and confident descriptions of operations that were never completed. In a conventional assistant, these failures remain in the conversational layer. In an agent, the same uncertainty moves into control. A hallucinated claim can become a search query; a mistaken assumption can select a tool; an incorrect parameter can alter a record; a false interpretation of tool output can trigger the next action. Agency changes the topology of error because outputs feed back into future inputs.

The most important failure is therefore not an isolated hallucination. It is propagation. Suppose an agent misidentifies a customer, retrieves the wrong account, interprets a policy exception incorrectly, and then writes a remediation note. Each local step may look coherent. The global trajectory is wrong because the early state was wrong. The probability of complete success on a long task is bounded by the reliability of every critical transition. Even modest per-step error can produce low end-to-end success when tasks require dozens of dependent actions. This is why benchmark results that look respectable at the level of tool calls can collapse on long business processes.

Real environments are also partially observable. The model rarely receives the full state of a browser, operating system, repository, organisation, or legal process. It receives a projection: an accessibility

tree, screenshot, selected files, API response, or summary. Hidden state may include prior transactions, access-control conditions, asynchronous updates, policy exceptions, or consequences that appear only later. An agent can be internally consistent with its observation and still act incorrectly in the real environment. Longer reasoning does not reveal a state that was never observed.

Benchmarks expose this gap. WebArena reported a large difference between baseline agents and human performance on realistic web tasks. OSWorld found a similar gap for general computer interaction. TheAgentCompany showed that long professional tasks involving communication, software, and office applications remained difficult, with the strongest baseline completing only a minority of tasks. Tau-bench further demonstrated that success deteriorates under repeated trials: an agent that occasionally completes a policy-rich task may be unreliable when asked to do so consistently (Zhou et al., 2023; Xie et al., 2024; Xu et al., 2024; Yao et al., 2024).

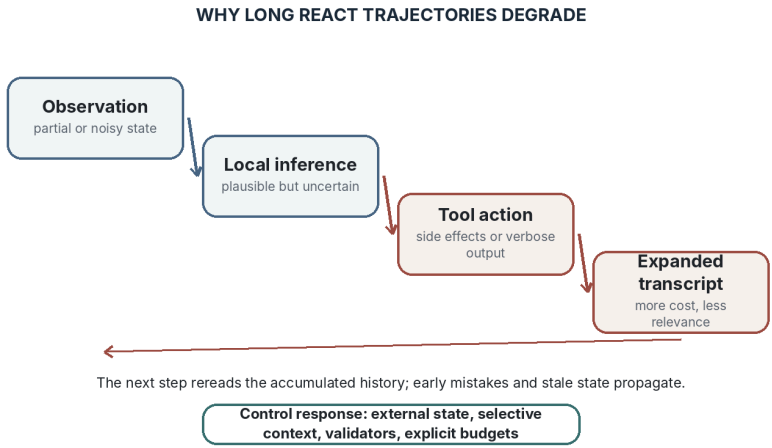


Figure 3.1. A long ReAct trajectory can turn partial observation into an expanding, costly, and increasingly ambiguous transcript.

3.2 Reasoning traces are useful interfaces, not faithful minds

Chain-of-thought and ReAct expose intermediate language that can improve performance and make trajectories easier to inspect. It is tempting to treat this language as a faithful transcript of the computation that produced the answer. The evidence does not justify that interpretation. Models can be influenced by cues they do not mention, can produce rationalisations after selecting an answer, and can offer different explanations for the same internal tendency. Work on unfaithful chain-of-thought showed that a model's stated reasoning may omit the actual influence of biasing features (Turpin et al., 2023). The trace is generated text with operational value; it is not direct access to a stable internal proof.

This matters for ReAct because the pattern gives the thought text a privileged role. The thought selects an action, and the full transcript may later be used for debugging, memory, or explanation. If the thought is unfaithful, the trace can still document what the agent said before acting, but it cannot guarantee why the model preferred the action. It is process telemetry, not a causal account of the neural computation. Trust should therefore rest on observable inputs,

authorised actions, environmental outcomes, and external validation rather than on the eloquence of the reasoning trace.

The assumption that more reasoning is automatically better is also unstable. The Curse of Chain-of-Thought reports settings in pattern-based in-context learning where explicit reasoning underperforms direct answering across multiple models and benchmarks. Apple's Illusion of Thinking experiments argue that reasoning models can show sharp degradation as controlled problem complexity rises, even when they generate longer traces. Such results do not prove that reasoning traces are useless. They show that trace length and cognitive reliability are different variables. A system can spend more tokens elaborating the wrong abstraction.

Self-correction has related limits. Without external feedback, models often fail to diagnose and repair their own reasoning reliably. A second pass may improve style or catch simple inconsistencies, but it can also reinforce the original answer or introduce new errors. External feedback changes the situation because it adds information: a test failure, a contradictory source, a solver result, or a human correction. Architecturally, the distinction is fundamental. Reflection should consume evidence, not merely more samples from the same uncertain belief state (Huang et al., 2023).

3.3 Planning is not solved by asking for a plan

Large language models can produce impressive plans in prose. They are much less reliable at maintaining a correct plan across hidden state, constraints, and long horizons. PlanBench and subsequent investigations found that models struggled with classical planning even when problems were expressed in natural language and when superficially similar examples were available. The failure is not surprising. Next-token prediction can imitate planning discourse without implementing a complete search procedure, state transition model, or invariant checker. A plan can sound strategically coherent while containing an impossible precondition or omitting a necessary resource (Valmeekam et al., 2023, 2024).

Agentic planning combines at least four problems. The system must infer the user's actual objective, decompose the objective, select relevant knowledge and tools, and order operations according to dependencies and changing observations. These problems are often collapsed into one prompt. A model can succeed at decomposition and fail at tool selection. It can select a correct tool and provide incorrect arguments. It can create a reasonable initial plan and fail to revise it after the environment contradicts an assumption. Evaluating only the final answer hides which competence failed.

Long-horizon behaviour is especially fragile because the agent must preserve commitments over many steps. SokoBench, designed around Sokoban-style planning, reports strong degradation beyond moderate solution lengths. Computer-use benchmarks show that agents may perform individual interface operations but fail to sustain coherent progress through multi-application tasks. In software engineering, agents are stronger because repositories provide executable tests and relatively clean text interfaces; even there, performance depends substantially on the agent-computer interface and the harness, not merely on model coding knowledge (Yang et al., 2024).

The correct response is not to abandon model planning. It is to reduce what planning must carry implicitly. Explicit dependency graphs, typed state, resource budgets, precondition checks, checkpoints, and local replanning make failures visible. A planner should be allowed to propose a structure, but the runtime should validate that structure where possible. Plans should encode what can run in parallel, what must be recomputed after a change, what side effects require approval, and what evidence closes a node. Planning becomes more reliable when it is compiled into an inspectable execution state rather than preserved only as conversational intention.

Table 3.1. Common failures and the control structures that address them.

Failure mode	Architectural response
Hallucinated state	Ground critical variables in tool output, typed records, or source-linked evidence.

Failure mode	Architectural response
Unfaithful reasoning trace	Treat trace as telemetry; verify outcomes independently.
Invalid or stale plan	Represent dependencies explicitly and support local replanning.
Context overload	Use scoped state, retrieval, compaction, and named intermediate variables.
Incorrect tool call	Constrain schemas, validate arguments, minimise permissions, and return actionable errors.
Premature completion	Evaluate the final environment state and acceptance criteria, not only the final text.

3.4 Structural limits of the ReAct loop

ReAct is elegant because the transcript stores both interaction and working state. The same property produces its most important structural limits. Every cycle appends reasoning, tool definitions, observations, errors, and corrections. Context becomes bloated, API cost grows, stale state remains visible, and relevant evidence competes with repetitive output. Long windows postpone truncation but do not guarantee use: Lost in the Middle demonstrated position-sensitive retrieval failures, while 2026 enterprise experiments found that full-history retention could be less reliable, slower, and more expensive than selective pruning plus summarisation. Long generation also creates opportunities for hallucination, internal logical inconsistency, and failure to preserve a complex specification across many local continuations (Liu et al., 2024; Lodha et al., 2026).

The second limit is sequentiality. A conventional ReAct loop waits for one model decision, one action, and one observation before beginning the next cycle. Independent operations are serialised even when their dependencies permit concurrency. LLMCompiler reported large latency and cost reductions by compiling model-proposed tasks into a parallel execution graph, and ReWOO

reduced repeated deliberation by separating planning from observations. These results are task-dependent rather than universal, but they establish that scheduling and state representation can change system capability without changing the foundation model (Kim et al., 2024; Xu et al., 2023).

The third limit is weak failure localisation. When the transcript is the program, a changed assumption can invalidate an unknown suffix of the conversation. The system may preserve conclusions that depended on stale data because dependency relations are implicit. Repetitive output can be mistaken for progress, and a later summary may conceal which premise produced a result. A graph or executable knowledge representation with named values can invalidate only affected nodes and can preserve the original artefacts. ReAct can imitate this discipline in prose, but the runtime has no guarantee that the imitation remains consistent.

The fourth limit is interpreter conflation. One model call may decide relevance, choose a tool, perform domain analysis, judge policy compliance, copy data between tools, and decide whether to stop. This creates common-mode failure and encourages natural language to mediate operations for which code, a database engine, a parser, or a policy checker would be more reliable. ReAct can call these components, but the pattern does not decide when model discretion should yield to a narrower interpreter. That choice belongs to the harness.

The fifth limit is budget opacity. Each additional step can look locally useful while global value declines. Reflection, retries, long outputs, and agent debates can multiply cost without adding independent evidence. Production systems need explicit budgets for tokens, elapsed time, tool calls, side effects, and human attention. They also need stopping criteria linked to acceptance conditions rather than to the agent's subjective feeling of completion. The same uncertain process that consumes a budget should not be the sole authority over that budget.

3.5 Tools, interfaces, and the action grounding problem

Tool use is often described as the cure for hallucination because tools connect the model to external facts and computation. In reality, tools move the problem. The model must discover the right tool, understand its description, construct valid arguments, interpret errors, and integrate the result. Performance declines when tool libraries become large, overlapping, or poorly documented. Tool selection is a retrieval and semantic-matching problem; argument construction is a constrained generation problem; result interpretation is another reasoning problem. The tool can be perfectly correct while the agent uses it incorrectly.

Agent-computer interfaces determine how much of the environment is legible. SWE-agent showed that a deliberately designed command and observation interface can improve automated software engineering. Browser and desktop agents face a harder environment: dynamic visual state, hidden elements, timing, multiple windows, ambiguous affordances, and adversarial page content. Screenshots provide visual fidelity but consume context and can carry indirect prompt injection. Accessibility trees are structured but incomplete. Direct APIs are efficient but may not exist. General computer use remains far below human reliability on long tasks because perception, state tracking, and action semantics are all unstable.

Good tool design minimises semantic ambiguity. Tools should have narrow purposes, typed schemas, explicit side effects, examples, permission metadata, and errors that support recovery. The runtime should expose only the tools relevant to the current operation. Anthropic's 2025 work on code execution with MCP makes the context problem concrete: loading large tool catalogues and passing full intermediate results through the model can consume enormous token budgets, whereas generated code can select tools and transform records outside the model context. Protocols standardise invocation, but they do not remove the need for authentication,

provenance, sandboxing, least privilege, and supply-chain review (Anthropic, 2025a).

3.6 Security, memory poisoning, and evaluation uncertainty

An agent that consumes untrusted content and can take actions creates a new security composition. Indirect prompt injection hides adversarial instructions in web pages, documents, emails, code comments, or tool output. The model may confuse this content with authorised instructions. AgentDojo provides a benchmark for measuring the tension between task utility and prompt-injection security. The deeper lesson is architectural: natural-language instructions from different trust domains must not be placed into one undifferentiated context and left for the model to rank. Defence requires privilege separation, content labelling, tool gating, sandboxing, allowlists, approval for consequential actions, and monitoring (Debenedetti et al., 2024).

Persistent memory extends the attack across time. An adversary may insert a note that appears harmless in the current task but changes future retrieval or policy interpretation. MemoryGraft and subsequent memory-poisoning benchmarks demonstrate that long-term memory can become an attack surface. Provenance, scope, retention review, and conflict detection are therefore security controls, not organisational niceties. A memory entry should never gain authority merely because it was generated fluently or retrieved frequently.

Evaluation is itself uncertain. Agent tasks are multi-turn and stateful; a score may depend on the complete trajectory, the final environment state, hidden policies, timing, and whether recovery was acceptable. LLM judges can scale evaluation but may share biases with the agent. Deterministic scorers can be reproducible but miss semantic value. Human review is expensive and inconsistent. AI Agents That Matter argues that agent benchmarks often overemphasise accuracy while underreporting cost, reproducibility, and practical complexity. The correct unit of comparison is the

complete system under a declared budget and environment, not a model label detached from its harness (Kapoor et al., 2024).

The limits in this chapter do not imply that agents are a dead end. They imply that autonomy is not a substitute for architecture and that generality should be claimed only on a declared distribution. Long traces are not proofs. More context is not guaranteed relevance. More agents are not guaranteed independence. More tools are not guaranteed grounding. The productive programme is to move repeated responsibility from implicit language-model behaviour into explicit state, specialised interpreters, bounded skills, and evaluation packages. ReAct remains valuable, especially as an exploratory fallback, but it should not be treated as the final form of every agent.

CHAPTER 4

Harnesses as the New Systems Layer

The model proposes; the harness decides what proposal becomes computation.

4.1 What an agent harness is

An agent harness is the software scaffold that turns a foundation model into an operational actor. It manages instructions, state, context, tool exposure, permissions, scheduling, retries, handoffs, human approvals, traces, and stopping conditions. The term is useful because it separates model capability from system capability. Two products can use the same model and achieve different results because one provides a better interface, smaller relevant context, stronger tools, executable feedback, and more disciplined recovery. Conversely, a strong model can appear weak inside a harness that hides state, overwhelms it with tools, or evaluates the wrong outcome.

The harness is not identical to a framework. A framework supplies programming abstractions and reusable components. A harness is the concrete control arrangement around a model in a particular system. The same framework can host several harnesses with different authority and evaluation. Nor is the agent harness identical to an evaluation harness. The evaluation harness supplies tasks, environments, scorers, concurrency, recording, and aggregation so that agent systems can be compared. In mature development, the two are connected: every production trace can become evaluation material, and every evaluation should exercise the same controls that matter in deployment.

The harness can be understood as a control plane around a probabilistic data plane. The model handles semantic interpretation and generation. The control plane determines which context is

visible, which operations are legal, how calls are routed, where state is stored, what must be checked, and how the process is observed. This division resembles operating systems, database engines, and distributed runtimes. It is not an exact analogy, because natural-language interpretation remains more ambiguous than a conventional instruction set. Yet the analogy is productive: agents need isolation, scheduling, identity, permissions, logging, recovery, and resource accounting.

By July 2026, harness controls are no longer peripheral. OpenAI's updated Agents SDK presents a model-native harness, native sandboxes, configurable memory, and separation between the harness and compute. Anthropic's long-running application work reports that decomposition, structured artefacts, evaluator separation, and context-reset or compaction strategies can dominate outcomes, while also warning that complex harnesses become slow and expensive. Google ADK combines sequential, parallel, and loop agents with OpenTelemetry, plugins, YAML configuration, and human confirmations. The industrial frontier is therefore a runtime problem: how to provide enough structure to improve reliability without freezing model capability or preserving assumptions that newer models no longer need (Anthropic, 2026a; Google, 2026; OpenAI, 2026a).

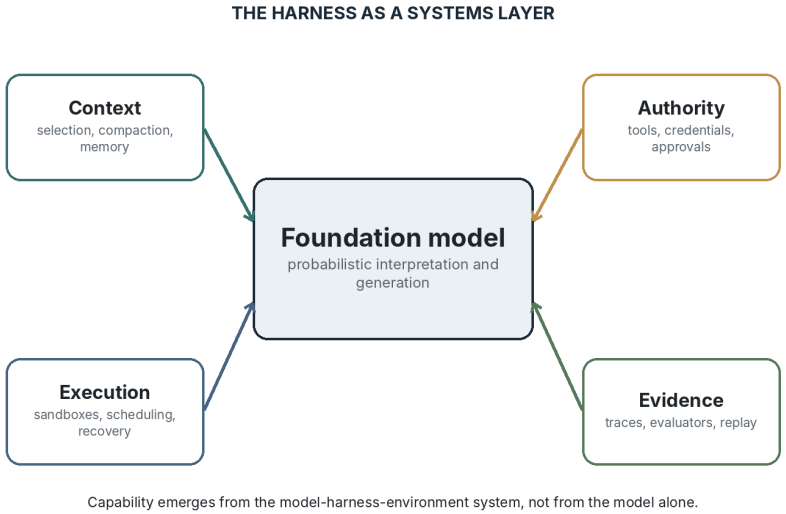


Figure 4.1. The harness surrounds probabilistic inference with context, authority, execution, evidence, and recovery.

4.2 The runtime responsibilities

The first responsibility is state management. A serious runtime distinguishes user input, environment state, working variables, session history, durable memory, and artifact storage. It supports checkpoints so a long task can resume after failure or approval. It controls isolation so one user's context does not leak into another's. It records versions of prompts, models, tools, and knowledge so a trajectory can be reconstructed. State is the difference between a demo that appears coherent for one run and a system that can be operated over time.

Context engineering is the harness responsibility most likely to be underestimated. The runtime decides which instructions, tool schemas, memories, artefacts, prior observations, and policies become visible for the next inference. More content can increase coverage and simultaneously reduce attention, increase latency, expose secrets, preserve stale state, and magnify prompt injection. Less Context, Better Agents reported an enterprise benchmark in which pruning and summarisation improved task completion while reducing tokens and elapsed time relative to full history.

ContextBudget formulates compression under a budget as a sequential decision problem. These are early results, but they support a strong systems conclusion: context is an actively managed operating substrate, not a transcript dump (Lodha et al., 2026; Wu et al., 2026).

The third responsibility is action mediation. Tool calls must be parsed, validated, authorised, executed, and returned in a form the model can use. The runtime can enforce schemas, rate limits, cost limits, data boundaries, and approval requirements. It can wrap side-effecting operations in transactions or dry runs. It can expose safe synthetic tools rather than raw credentials. It can translate low-level errors into structured recovery signals. The model should not directly own secrets, network access, or irreversible authority merely because it can produce the right JSON shape.

The fourth responsibility is scheduling. A scheduler decides when to continue a loop, when to branch, which work can run in parallel, when to use a smaller model, when to call a specialist, and when to stop. Scheduling is increasingly cost-aware. It can allocate a reasoning budget according to task value, use cascades, or select a deterministic implementation when a stable skill exists. The scheduler also handles retries and backoff. Retries should be classified, because repeating a transient network call differs from repeating a reasoning failure with identical context.

The fifth responsibility is verification and governance. Guardrails can inspect input, output, tool arguments, and environment state. A verifier can test code, check a schema, compare a claim with retrieved sources, or enforce a business rule. Human approval can be inserted before a high-consequence action or after evidence is assembled. Policies should be local to the action boundary: reading a public page and issuing a refund do not require the same assurance. The harness converts abstract principles such as least privilege and human oversight into executable gates.

The sixth responsibility is observability. Tracing records model calls, tool calls, handoffs, latency, token use, errors, policy decisions, and final outcomes. OpenTelemetry's emerging generative-AI

semantic conventions reflect the movement toward vendor-neutral instrumentation. Observability makes debugging possible, but its deeper function is epistemic: it records what evidence and authority were present when the system acted. A trace is not proof that the action was correct. It is the minimum material from which correctness, cost, and accountability can later be assessed.

Table 4.1. The harness as a stack of operational responsibilities.

Harness layer	Operational duty
State layer	Sessions, checkpoints, isolation, artefacts, and versioned runtime variables.
Context layer	Retrieval, compaction, memory selection, prompt assembly, and tool disclosure.
Action layer	Schema validation, permissions, execution, transactions, and error translation.
Scheduling layer	Loops, graphs, parallelism, routing, budgets, retries, and termination.
Assurance layer	Guardrails, validators, approvals, policy checks, and recovery decisions.
Observability layer	Traces, metrics, cost attribution, replay, evaluation, and audit evidence.

4.3 The 2026 industrial landscape

OpenAI's stack combines the Agents SDK with tools, handoffs, guardrails, sessions, tracing, evaluation, and AgentKit. In 2026 the SDK added a model-native harness and native sandbox execution for file and command work, together with a portable workspace manifest and a stronger separation between control and compute. AgentKit adds visual workflow construction, connectors, embedded interfaces, datasets, trace grading, and prompt optimisation. The stack illustrates a broad industrial tendency: a turnkey agent loop is valuable, but production differentiation moves into domain logic, context, tools, evaluators, and governance (OpenAI, 2025b, 2026a).

Anthropic presents a deliberately pattern-oriented view. Its Building Effective Agents essay distinguishes workflows from

dynamic agents and recommends the simplest composition that works. Classical Agent Skills package instructions, scripts, and resources for progressive discovery by a broad agent. Work on long-running application development separates planner, generator, and evaluator roles and uses structured artefacts to maintain continuity. These contributions are important external reference points, but they should not be conflated with the AchillesAgentLib taxonomy developed in Chapter 8, where orchestration skills coordinate executable families with different runtime semantics (Anthropic, 2024, 2025b, 2026a).

Google's Agent Development Kit exposes model-agnostic agents, workflow agents, tools, sessions, memory, evaluation, and deployment across several languages. ADK Go 1.0, released in March 2026, highlights sequential, parallel, and loop agents, OpenTelemetry tracing, plugin-based self-healing, human-in-the-loop confirmations, and YAML-defined agents. The architecture illustrates how an agent framework increasingly resembles a service runtime with configuration, observability, policy insertion, and deployment portability rather than a library of prompt templates (Google, 2026).

Microsoft's agent work connects orchestration to enterprise identity, data, and operational governance. Its framework lineage combines AutoGen-style multi-agent conversation with more explicit workflows, telemetry, state, and service integration. The strategic advantage is not that conversation among agents is inherently superior. It is that agent identities and actions can be integrated with organisational permissions, connectors, and monitoring. The corresponding risk is platform complexity: every additional connector and delegation boundary increases the need for provenance, least privilege, and complete tracing (Microsoft, 2026).

LangGraph and OpenHands represent two complementary directions. LangGraph treats durable graph state, checkpoints, interrupts, and human review as core abstractions for arbitrary agent workflows. OpenHands concentrates on software-engineering agents and the agent-computer interface, where repositories, shells, editors, sandboxes, tests, and event streams form the working

environment. Both show that the operational interface can matter as much as prompting. Better observations and executable feedback often outperform additional free-form reasoning because they change what the model can reliably perceive and verify (Wang et al., 2025).

4.4 Protocols and interoperability

MCP standardises a host-client-server architecture through which agents discover and invoke tools, resources, and prompts. The protocol reduces integration duplication and enables a large ecosystem, but it does not define whether a server is trustworthy, which user delegated an action, or whether returned content is data or instruction. A serious MCP host therefore needs authentication, capability restriction, provenance, context filtering, and audit. As tool ecosystems grow, progressive disclosure and code-mediated access become important because loading every definition and result into model context is neither efficient nor safe (Model Context Protocol, 2025; Anthropic, 2025a).

The Agent2Agent protocol addresses a different seam: communication between agents or agent services. It aims to make capabilities, tasks, messages, artefacts, and progress interoperable across vendors and was transferred to the Linux Foundation in 2025. MCP and A2A are complementary: one primarily connects an agent to resources and tools, the other connects agentic services. Neither supplies accountability by itself. Identity, authority, cancellation, confidentiality, and responsibility for a composed outcome remain harness and organisational concerns (Linux Foundation, 2025).

Protocol standardisation does not remove semantic ambiguity. Two tools can share a schema and disagree about side effects, freshness, or authority. Two agents can exchange a task and hold different assumptions about completion. Interoperability therefore needs capability descriptions, provenance, versioning, trust domains, idempotency, cancellation, and error taxonomies. The next phase is likely to resemble distributed systems more than chatbot integration.

Agents will need service contracts and operational semantics, not only natural-language descriptions.

4.5 Evaluation harnesses and the research frontier

Research on harnesses is becoming more explicit. Architectural Design Decisions in AI Agent Harnesses studies design variation across dozens of open systems, while AI Harness Engineering formalises the model-harness-environment system and proposes auditable episode packages. MemoHarness, published as a July 2026 preprint, explores harnesses that retrieve and adapt control experience from prior executions. These works differ in maturity, but together they mark a conceptual transition: prompts and workflows are no longer the only optimisation target; context policy, tool exposure, memory, verification, and intervention logic are themselves learnable or designable artefacts (Huang et al., 2026; Wei, 2026; Zhong and Zhu, 2026).

This expansion raises a methodological problem. A more elaborate harness can improve a benchmark by adding hidden tools, privileged information, more inference, or evaluator-specific rules. AI Agents That Matter argues that practical evaluation must include cost, reproducibility, held-out tasks, and complete system configuration. Harness research therefore needs ablation: hold the model and environment fixed, add one control responsibility at a time, and inspect trace-level effects. When the application is local, the benchmark should also be local. A system designed for one recurring enterprise process should be judged on that process and its failure consequences, not marketed as generally superior because it has more components (Kapoor et al., 2024).

Evaluation should operate at three levels. Component tests verify parsers, schemas, permissions, and deterministic validators. Trajectory tests inspect routing, context, tool selection, recovery, and stopping. End-to-end tests evaluate the resulting environmental state under repeated trials, realistic budgets, and perturbations. The episode package should include model and prompt versions, tool and data versions, selected context, actions, observations, evaluator

outputs, costs, timings, and human interventions. Without this package, an agent score is difficult to reproduce and almost impossible to diagnose.

The frontier is a harness that can improve without silently rewriting its own authority. Production traces and user feedback can reveal missing context, brittle instructions, redundant calls, and candidates for deterministic implementation. The system may propose changes to prompts, skills, routing, evaluators, or code, but those changes should be evaluated offline, compared against the previous version, reviewed according to risk, and promoted with rollback. This controlled learning loop anticipates the sleep-phase design developed for MRP-VM in Chapter 9.

PART II TRUST, ECONOMICS, AND DESIGN

CHAPTER 5

Trustworthy Agentic AI: European Philosophy, Law, and Engineering

Trust is not confidence in a model. It is justified reliance on a governed system.

5.1 The European meaning of trustworthiness

Trustworthy AI is often reduced to a collection of safety techniques. The European tradition is broader. The High-Level Expert Group on Artificial Intelligence defined trustworthy AI through three simultaneous conditions: it should be lawful, ethical, and robust. The seven operational requirements are human agency and oversight; technical robustness and safety; privacy and data governance; transparency; diversity, non-discrimination, and fairness; societal and environmental well-being; and accountability. These requirements are not a model benchmark. They are lifecycle obligations that connect technical design to fundamental rights, institutions, and human purposes (AI HLEG, 2019, 2020).

Agentic systems make this philosophy more demanding because they do not merely recommend. They interpret goals, select procedures, interact with other systems, preserve memory, and perform actions. The relevant object is therefore the full sociotechnical arrangement: models, prompts, tools, data, memory, deployment infrastructure, human operators, affected persons, and organisational policies. A model can be accurate in isolation while the agentic system remains untrustworthy because authority is excessive, redress is absent, logs are incomplete, or the workflow systematically disadvantages a group.

The correct definition of trust is justified reliance. A person or institution may rely on an agent only to the extent that the system has demonstrated suitable competence, bounded authority, traceable operation, and remedies for failure. This definition rejects both technological optimism and blanket distrust. It asks what claim has been warranted for which task, environment, population, time period, and consequence level. An agent that is trustworthy for drafting may not be trustworthy for submission. An agent that is trustworthy in a sandbox may not be trustworthy in production. Trust must attach to a configured use, not to an abstract model family.

The European approach also resists the idea that human oversight is a cosmetic control. Human-in-the-loop, human-on-the-loop, and human-in-command represent different allocations of authority. A person who receives an incomprehensible final recommendation under time pressure is not meaningful oversight. Effective oversight requires information, time, competence, and the ability to intervene. Agent harnesses must therefore design the human interface as carefully as the model interface: show the decision that is requested, the evidence assembled, the alternatives considered, the remaining uncertainty, and the consequences of approval.

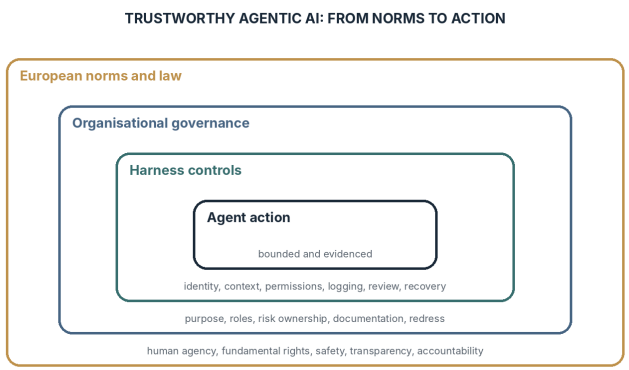


Figure 5.1. European trustworthy-AI principles become effective only when translated through governance into runtime controls.

5.2 From seven requirements to agent controls

Human agency and oversight become controls on delegation. The runtime should define which actions are advisory, reversible, reviewable, or prohibited. It should support escalation and interruption. It should prevent a chain of agent-to-agent delegation from silently expanding authority. The person responsible for a process must remain able to understand the current state, change the goal, withdraw permission, and contest the outcome. Human command is not achieved by inserting an approval button after all substantive choices have already been made.

Technical robustness and safety become defence in depth. Models will fail, tools will fail, external services will change, and adversaries will manipulate content. Robustness therefore includes schema validation, typed state, isolation, least privilege, fault classification, checkpointing, fallback, rollback, and safe termination. Reliability should be measured across repeated runs and perturbations, not inferred from one successful demonstration. Security tests must include indirect prompt injection, malicious tools, memory poisoning, credential misuse, and unexpected interaction among agents.

Privacy and data governance become context governance. An agentic runtime continuously decides what data to retrieve, expose to a model, send to a tool, preserve in memory, or include in a trace. Data minimisation should therefore operate at every step, not only at initial collection. Sensitive data can be protected through local processing, scoped retrieval, redaction, access controls, retention limits, and model routing according to data boundary. Persistent memory needs explicit legal basis, purpose, source, expiration, and deletion semantics. A useful memory is not automatically a legitimate memory.

Transparency becomes traceability, communication, and process intelligibility. The system should record model and tool versions, instructions, retrieved sources, key variables, approvals, and final actions. The user should know that an agent is involved and understand its capabilities and limits. Yet transparency should not

mean dumping a raw chain-of-thought transcript. Raw traces may be misleading, private, or insecure. The better target is a process explanation: what information was used, what procedure was followed, what was verified, what remained uncertain, and who authorised the result.

Diversity, non-discrimination, and fairness become requirements on data, workflow, interface, and outcome. An agent can create disparate effects through routing, memory, tool access, or escalation even if its base model passes a generic bias benchmark. Fairness must be tested at the system level and across the task distribution actually served. Accessibility also matters because agent interfaces can exclude users who cannot interpret dense traces or operate complex approval flows. Stakeholder participation is especially valuable when the agent changes an institutional process rather than merely automating an existing screen.

Societal and environmental well-being become design constraints on purpose, labour, democracy, and resource use. Agentic systems can increase productivity, but they can also intensify surveillance, centralise control, obscure responsibility, or flood institutions with low-cost generated actions. Energy and compute should be measured because iterative loops, multi-agent debate, and verification can multiply inference. Efficiency is therefore part of trustworthiness: unnecessary computation is not only a financial problem but a sustainability and accessibility problem.

Accountability becomes evidence and decision ownership. Every consequential operation should have an identifiable operator, policy basis, and route for review. The system should preserve enough evidence to determine what happened without implying that the trace proves correctness. Responsibilities among model provider, harness developer, tool provider, deployer, and user should be explicit. A multi-vendor agent stack can otherwise create an accountability gap in which each component behaved according to its local contract while the combined system harmed someone.

Table 5.1. Translating European trustworthy-AI requirements into harness design.

European requirement	Agent-system consequence
Human agency and oversight	Bounded delegation, interruption, meaningful approval, and contestability.
Technical robustness and safety	Isolation, least privilege, recovery, repeatability, and adversarial testing.
Privacy and data governance	Step-level minimisation, scoped memory, provenance, retention, and access control.
Transparency	Process explanations, traceability, disclosure, and capability communication.
Fairness and diversity	System-level outcome testing, accessibility, and stakeholder participation.
Societal and environmental well-being	Purpose review, labour effects, democratic impact, and compute efficiency.
Accountability	Audit evidence, responsibility allocation, redress, and governed release.

5.3 The AI Act as architecture pressure

The AI Act applies generally from 2 August 2026, with earlier application for prohibited practices, AI literacy, governance, and general-purpose AI obligations. A political agreement reached on 7 May 2026 adjusted the implementation path for high-risk rules: the Commission's current timeline places certain stand-alone high-risk systems at 2 December 2027 and product-embedded systems at 2 August 2028, subject to completion of the legislative process and supporting measures. Agent builders should therefore distinguish law already applicable, obligations entering application on 2 August 2026, and high-risk provisions whose dates have been politically revised. This book states the position as of 23 July 2026 rather than treating regulatory timelines as static (European Commission, 2026a; European Union, 2024).

The AI Act does not contain a special universal category called agent. An agentic application must be classified according to its intended purpose, role in the value chain, deployment context, and

the risks of the system it constitutes. This is important because an identical general model can support a low-risk drafting assistant or become part of a high-risk employment, education, identity, health, migration, or critical-infrastructure process. The harness determines how the model is embedded, what authority it receives, and what records exist. Regulatory analysis therefore belongs at architecture time, not after deployment.

High-risk obligations create pressure for risk management, data governance, technical documentation, logging, transparency to deployers, human oversight, accuracy, robustness, cybersecurity, quality management, and post-market monitoring. Agentic systems complicate each obligation. Dynamic planning makes the realised workflow variable. Tools and remote agents create dependencies outside the model boundary. Memory creates evolving data. Continuous updates can change behaviour without an obvious product release. A compliant architecture must control configuration, versioning, approved capability sets, and evidence across the whole execution chain.

General-purpose AI rules also matter downstream. Providers must supply information that enables integrators to understand capabilities and limitations; models with systemic risk face additional safety and security expectations. The voluntary General-Purpose AI Code of Practice, finalised in July 2025, organises support around transparency, copyright, and, for the most advanced models, safety and security. Agent developers cannot outsource their own responsibility to the model provider, but provider documentation, evaluation, incident information, and usage policies become inputs into the system safety case (European Commission, 2025).

Article 50 transparency obligations apply from 2 August 2026. On 20 July 2026, the Commission published implementation guidelines covering disclosure when people interact with certain AI systems and transparency for AI-generated or manipulated content. Some pre-existing systems receive a limited transition for specific marking and detection duties until 2 December 2026. For agentic systems,

transparency should not be reduced to a generic chatbot label. Users need to understand that they are interacting with AI, when an agent acts through external services, what role automation played in a material output, and how they can obtain human review or contest a consequential result (European Commission, 2026b).

5.4 Specification, contestability, and the European opportunity

The most productive European contribution to agentic AI may be a design discipline rather than a single model. The discipline begins with specifications: objectives, prohibited outcomes, data boundaries, authority, evidence thresholds, evaluation cases, human decision rights, and stopping conditions. These specifications should remain connected to runtime traces and test suites. A system becomes easier to govern when the reason for each control can be traced to a declared requirement and when deviations trigger visible review.

Contestability is the ability to challenge an intermediate or final result with a different source, interpreter, expert, or rule. Agentic systems often optimise for smooth completion. Trustworthy systems must also preserve disagreement and unresolved status. A result should be able to remain provisional when evidence is incomplete. A local conclusion should be reopenable without rerunning every unrelated step. People affected by an automated process should have a route to meaningful reconsideration, not only an explanation of why the first result was produced.

Modular assurance follows. No single reviewer can certify an entire stack that combines models, retrieval, code, data, policy, and domain practice. Assurance should be distributed across competent mechanisms. A database constraint can certify one property; a formal method another; a security team the deployment boundary; a domain expert the interpretation; a governance body the release decision. The harness should compose these certificates without pretending they are interchangeable. This is close to an assurance-case model in which claims are supported by typed evidence and explicit assumptions.

ACHILLES enters this landscape with the project objective captured by the phrase Lighter, Clearer, Safer. Lighter points to efficiency, modularity, and reduced dependence on unnecessarily expensive inference. Clearer points to specification, explanation, traceability, and visible boundaries between forms of reasoning. Safer points to control, compliance, and human-centred assurance. AchillesAgentLib and MRP-VM should therefore be evaluated not as attempts to create a universally dominant agent, but as experiments in making selected recurring capabilities more efficient, inspectable, and governable (European Commission, 2026c).

The European philosophy should not be interpreted as an argument for eliminating useful autonomy. It is an argument for autonomy that remains accountable to human purposes and public rules. A trustworthy agent can be highly capable and adaptive, provided that its authority is bounded, its actions are observable, its memory is governed, its failure modes are tested, and its decisions remain contestable. This philosophy does not slow agentic engineering when incorporated early. It reduces the cost of discovering too late that a technically impressive loop cannot be defended, audited, or lawfully deployed.

CHAPTER 6

The Economics of Agency: Cost, Latency, and Execution Efficiency

An agent does not consume intelligence in the abstract. It consumes tokens, tool time, network round trips, review attention, and risk budget.

6.1 The real cost function

The cost of an agent is often estimated from input and output tokens. This is necessary but incomplete. An agentic execution consumes model inference, tool calls, retrieval, storage, sandbox time, network latency, verification, retries, observability, and human attention. It can also create expected loss through incorrect actions. A useful cost model therefore includes direct compute, elapsed time, operational overhead, review effort, and risk. The cheapest model call can produce the most expensive workflow if it creates ambiguity that requires repeated repair.

For a trajectory with n model calls and m tool operations, direct monetary cost can be approximated as the sum of model input and output charges, tool and infrastructure charges, storage and trace charges, and human review. Latency is different because parallel operations do not add linearly; the critical path determines the user's wait. Risk is different again because a rare high-impact side effect can dominate expected value. Architecture should therefore optimise a vector rather than one scalar: success probability, cost, latency, resource use, and consequence.

Context cost is not merely proportional to the user request. In a loop, prior instructions, tool definitions, observations, and corrections may be resent repeatedly. One 2026 enterprise study compared context policies on a 50-task expense-itemisation

benchmark. Full conversation history achieved 71.0 percent completion but consumed 1,480,996 tokens and 14.56 hours; pruning plus automated summarisation achieved 91.6 percent completion with 553,374 tokens and 5.79 hours. These numbers come from one workflow and should not be universalised. Their significance is that selective context improved both quality and efficiency by removing stale state while preserving a compact task model (Lodha et al., 2026).

Tool latency can dominate model latency. Browsing, package installation, database queries, compilers, remote agents, and human approvals may take seconds or minutes. In April 2026, OpenAI reported up to 40 percent faster end-to-end performance for some agentic workflows using persistent WebSocket connections, incremental state, and reduced round-trip overhead. The number is implementation-specific, but the architectural lesson is broad: as inference accelerates, orchestration, networking, context assembly, and tool execution become visible bottlenecks. Efficiency work must therefore examine the complete critical path rather than model tokens alone (OpenAI, 2026b).

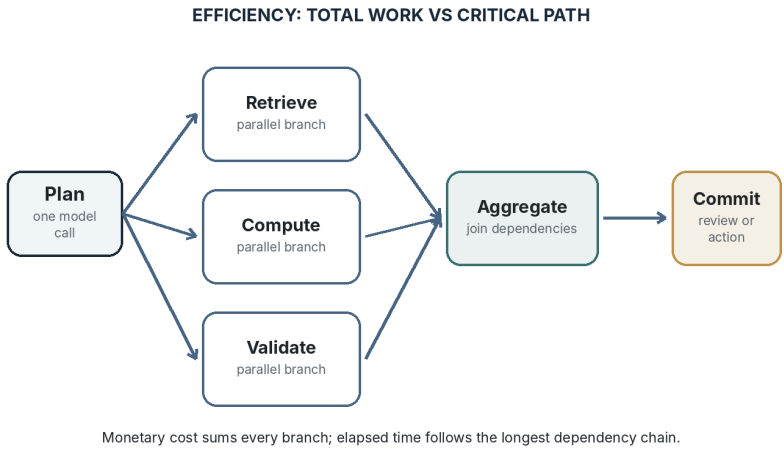


Figure 6.1. Total cost accumulates across executed branches, while elapsed time follows the critical dependency path.

6.2 Why ReAct is often expensive

A basic ReAct loop serialises interpretation and action. The model selects one operation, waits for the result, rereads the accumulated context, and selects the next. This is appropriate when each step depends materially on the previous observation. It is inefficient when the task contains independent retrievals, transformations, or tests. The loop also tends to expose a broad tool set and to preserve verbose observations, causing context and selection cost to grow with the trajectory. Its flexibility is valuable, but its cost and latency are intrinsically variable.

LLMCompiler attacks the scheduling problem by generating a task graph and executing independent calls concurrently. Its experiments reported up to 3.7 times lower latency and up to 6.7 times lower cost in selected settings, with accuracy improvements on some tasks. These values are not universal constants, but they show that a scheduler can change the Pareto frontier without changing the foundation model. The relevant question is whether dependencies can be predicted safely enough to earn the planning overhead (Kim et al., 2024).

ReWOO separates reasoning from observations. A planner produces a sequence with variables to be filled by tools, reducing the need for a large model after every result. The approach is efficient when the required operations can be anticipated and brittle when observations redefine the problem. This illustrates a general trade: precomputation spends adaptability to reduce repeated inference. AchillesAgentLib later makes a similar trade locally, but through explicit orchestration and executable skill families rather than through one universal plan format.

ReSum addresses long search trajectories by periodically compressing history into a compact reasoning state. The 2025 work reported an average absolute improvement over ReAct and additional gains after training models to operate across summary boundaries. ContextBudget treats the remaining space as an explicit budget and learns when and how much to compress. Both lines show that context management can be an agent policy in its own right.

They also expose the danger of irreversible summaries: efficient systems should preserve links to original artefacts, named state, and reasons for omission (Wu et al., 2025, 2026).

6.3 Pattern-level efficiency

Fixed workflows are usually the most predictable. They invoke a known number of stages and can cache intermediate results. They are efficient when the task distribution matches their assumptions. Their inefficiency appears as overprocessing: every case may execute steps needed only for rare cases. Conditional routing improves this by selecting only the necessary workflow. The router introduces a small cost and a risk of misclassification in exchange for large savings across heterogeneous traffic.

Planner-executor systems pay an upfront planning cost but can reduce repeated deliberation. They can allocate cheap workers to local tasks, execute independent nodes in parallel, and recompute only affected nodes after a correction. Planning pays off when tasks are sufficiently complex and decomposable. For a two-step task, the planner is overhead. For a fifty-step research or coding task, a good dependency graph can reduce both cost and latency.

Evaluator-optimizer loops deliberately add calls. Their economic justification depends on the value of quality improvement. A deterministic validator is often inexpensive and should be used whenever possible. An LLM evaluator can be appropriate when quality is semantic, but repeated generator-evaluator cycles need a stopping rule. The system should estimate whether another revision is likely to improve the accepted outcome more than it costs. Infinite polishing is not intelligence.

Multi-agent systems multiply inference and communication by default. They become efficient only when workers execute truly separable tasks in parallel, specialists use proportionate models, or independent verification prevents expensive downstream failure. A 2025 study of multi-agent failures identified fourteen recurring modes across specification, inter-agent alignment, verification, and termination, and found that popular coordination interventions did

not remove the deeper problems. The correct comparison is not one agent versus many in the abstract. It is accepted quality, critical-path time, total resource use, and failure severity for a declared task distribution (Cemri et al., 2025).

Memory can reduce or increase cost. Reusing a proven procedure avoids rediscovery. Retrieving a compact preference avoids replaying a long conversation. Conversely, searching a large uncurated memory, validating stale entries, and sending retrieved text to a model may exceed the cost of recomputation. Memory should be evaluated by net value: how often an entry is reused, how much inference it saves, how much risk it introduces, and what maintenance it requires.

Table 6.1. Efficiency is conditional on task structure, not an intrinsic ranking of patterns.

Pattern	Typical efficiency profile
Fixed workflow	Low variance and reusable work; inefficient when every case runs unnecessary stages
Router plus workflows	Small routing cost can avoid large unnecessary paths
ReAct loop	Adaptive but sequential, context-heavy and difficult to budget
Planner-executor graph	Planning overhead can unlock parallelism and selective recomputation
Evaluator-optimizer	Adds calls deliberately; valuable only when quality gains exceed review cost
Multi-agent system	Can shorten critical path or multiply redundant context and coordination
Stabilised skill or AKU	Upfront engineering can reduce repeated inference on recurring work

6.4 Model routing, cascades, and frugal inference

A general agent does not need the strongest model for every decision. Routing can assign tasks according to difficulty, data sensitivity, latency requirement, and consequence. A small model can classify requests, select a skill, summarise routine tool output, or enforce a known schema. A frontier model can handle novel planning or ambiguous synthesis. A deterministic component can perform arithmetic, parsing, validation, and policy enforcement. The architecture becomes a heterogeneous computer rather than a repeated call to one universal model.

FrugalGPT demonstrated that model cascades can reduce cost dramatically on selected benchmarks while preserving or improving quality. RouteLLM learned routing policies that reduce the use of expensive models while maintaining performance targets. The exact savings depend on current prices and model quality, so they should not be generalised mechanically. The durable lesson is that model choice is a runtime decision. Routing should be evaluated with held-out tasks, monitored for distribution shift, and constrained when data or regulation limits where a request may go (Chen et al., 2023; Ong et al., 2024).

Speculative execution can further reduce latency. A cheap model or deterministic planner begins work while a stronger model handles a harder branch. Results are accepted only if verification passes. Caching can reuse identical or semantically stable calls. Batch processing can group independent requests. Persistent network connections reduce round-trip overhead. Code execution can transform large datasets without returning every record to model context. Each technique attacks a different term in the cost function, and each can create new invalidation or security concerns.

The strongest efficiency mechanism is stabilisation. When a recurring model-mediated behaviour becomes understood, parts of it can become code, a constrained query, a typed data transformation, a validator, an orchestration skill, or an Agentic Knowledge Unit. This

shifts work from repeated probabilistic interpretation into cheaper execution while retaining a broader model for genuine semantic uncertainty. Stabilisation should be incremental and reversible: new structure must prove that it preserves the nuance required by the task. This is the shared engineering direction of AchillesAgentLib and MRP-VM.

6.5 Measuring efficiency without gaming it

Efficiency evaluation should report task success, accepted quality, monetary cost, elapsed time, model calls, input and output tokens, context construction, tool time, retries, and human review. For side-effecting tasks, it should include recovery cost and failure severity. A low-cost system that silently rejects difficult cases is not necessarily efficient, and a high-success system that consumes ten times the budget may be dominated. The meaningful object is a local Pareto frontier: the quality, assurance, and coverage achieved for a declared total burden.

Benchmarks can be gamed when systems exploit evaluator artefacts, use hidden human labour, omit failed retries, or compare unequal tool access. Cost reports can exclude preprocessing, vector indexing, sandbox infrastructure, coding-agent improvement phases, or reviewer time. Latency reports can ignore queueing. A trustworthy evaluation harness records the full episode and declares the accounting boundary. For systems that evolve through a sleep phase, both online serving cost and offline improvement cost should be reported, even when amortised over later executions.

The final metric is useful, evidenced work per constrained resource. This formulation connects efficiency to access, sustainability, and trust. Removing validation may make a system appear faster while exporting repair cost to users. Expanding context may appear safer while increasing distraction and stale-state error. The efficient agent is not the one that stops earliest or uses the fewest tokens. It is the system that reaches an acceptable, inspectable outcome with the least total burden and a known boundary of uncertainty.

CHAPTER 7

The Design Triangle: Trustworthiness, Efficiency, and Generality

Generality buys coverage. Structure buys assurance. Good architecture composes both instead of pretending the trade-off has disappeared.

7.1 The apparent conflict

Agent design is shaped by three ambitions that rarely reach their maximum together. Generality is coverage of diverse and unfamiliar tasks. Efficiency is useful work under limits of time, compute, context, and coordination. Trustworthiness is justified reliance through robustness, transparency, legality, controllability, and accountability. A free LLM planner appears general because it can invent a process. A hardcoded workflow appears efficient and trustworthy because its path is known. The important question is not which label wins, but which architecture occupies the best local position for a declared distribution and consequence regime.

The conflict is real but often stated too crudely. A workflow can encode a bad rule or fail outside its assumptions. An LLM planner is general only relative to its training, tools, context, and evaluation. A skill library can be highly dependable yet omit the global nuance needed for an unusual case. A multi-agent system may multiply the same weakness. The three dimensions must therefore be measured together, and the architecture must make its coverage boundary visible rather than using the word agent as a promise of universal competence.

No Free Lunch provides a useful intuition: no algorithm dominates across all possible problem distributions without assumptions. Practical intelligence succeeds by exploiting

regularities in data, tools, organisations, and tasks. Trustworthy architecture should make those regularities and assumptions explicit. The goal of AchillesAgentLib and MRP-VM is consequently not to defeat a general ReAct agent everywhere. It is to find recurring regions of the task space where stronger assumptions can be converted into better assurance and lower cost.

The triangle is dynamic. A novel task may begin with broad interpretation and human review. Repeated use reveals stable substructures. Those substructures can become tests, code, database abstractions, orchestration skills, or AKUs, increasing efficiency and trustworthiness. The unresolved variability remains available to broader models. Generality migrates upward from unconstrained local steps to composition across a growing portfolio of bounded capabilities.

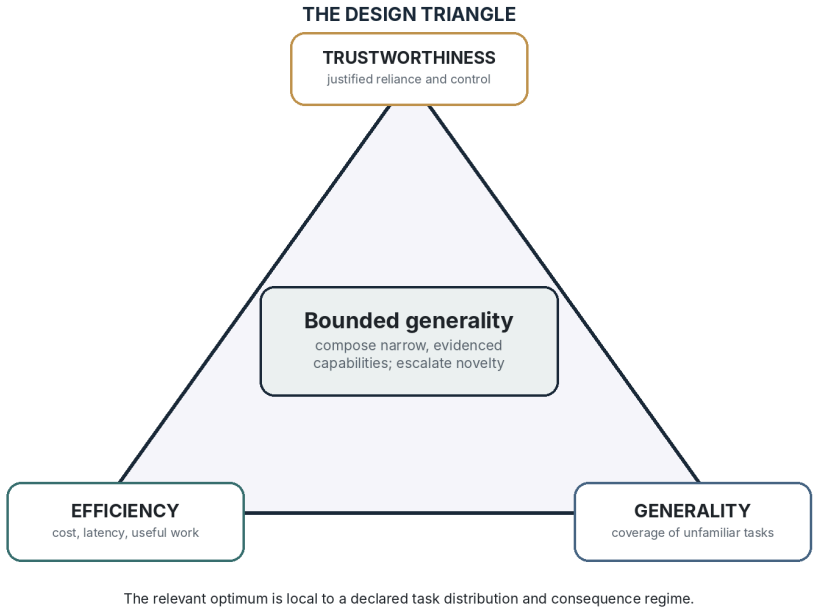


Figure 7.1. Bounded generality seeks a local balance among trustworthiness, efficiency, and coverage.

7.2 Hardcoded workflows and free planning

A hardcoded workflow externalises the plan. Developers decide the stages, branches, data contracts, and checks. This improves reproducibility and makes cost easier to estimate. It allows formal validation of some paths and precise control of side effects. The price is limited coverage. When the request does not match an anticipated branch, the workflow either fails, forces the case into an inappropriate template, or requires human redesign. Hardcoding is strongest in high-volume, stable processes with clear acceptance criteria.

A free LLM planner internalises more of the plan in model inference. It can interpret novel language, combine tools in unexpected ways, and repair after observations. This increases coverage and can reduce upfront engineering. The price is variable path length, weak dependency representation, prompt sensitivity, and difficult assurance. Every new plan is partly a new program generated at runtime. Testing must therefore focus on distributions and controls rather than enumerating paths.

The choice is not binary. A bounded planner can select among approved workflow fragments. A graph can contain deterministic nodes and model-driven branches. A planner can propose a plan that a validator checks against schemas, permissions, and resource limits. A specialist can use ReAct inside a sandbox while the outer workflow remains fixed. A human can approve only the transition from analysis to side effect. These hybrid patterns preserve generality where uncertainty exists and structure where the organisation already knows what good execution means.

The central design rule is to place model discretion at semantic boundaries, not mechanical boundaries. Models are valuable when the system must interpret ambiguous goals, classify unusual cases, extract meaning from unstructured evidence, or reconcile conflicting narratives. Conventional software is superior when the task is arithmetic, parsing, access control, transaction handling, dependency scheduling, exact matching, or enforcement of a formal rule.

Generality improves when each interpreter is used for the kind of uncertainty it can actually resolve.

Table 7.1. Generality can be added incrementally rather than granted as unrestricted control.

Control regime	Characteristic trade-off
Hardcoded workflow	High predictability on known cases; limited coverage outside assumptions
Workflow with adaptive nodes	Structure around stable stages and model discretion at semantic boundaries
Planner over bounded capabilities	Compositional generality with typed execution and local validation
Open ReAct agent	Broad exploration with variable path, cost, context and assurance
Portfolio of specialised agents	Generality through routing and shared protocols rather than one universal actor
Evolvable harness	Controlled offline growth with evaluation, versioning and rollback

7.3 Bounded generality

Bounded generality is the ability to handle variable tasks within explicit limits of authority, resources, and evidence. It differs from narrow automation because the system can interpret and adapt. It differs from unrestricted agency because the adaptation occurs inside a controlled envelope. The envelope may specify available tools, data domains, maximum spend, permitted side effects, required validators, escalation rules, and retention policy. A system can be general at the level of problem interpretation while remaining narrow at the level of action.

Reusable capabilities are a practical vehicle for bounded generality, but their semantics matter. In the broad industry, a skill may mean reusable instructions and resources loaded into a general agent. In AchillesAgentLib, an orchestration skill coordinates one or more executable families: CSkills, CGSkills, and DBTable Skills. In MRP-VM, the core abstraction becomes the AKU, which includes not only instructions but also interpretation, interfaces, provenance, and evaluation metadata. The common idea is reuse; the difference is how much execution meaning the runtime makes explicit.

Typed actions are another vehicle. The model may generate a proposal in natural language, but the action must inhabit a finite schema and pass validation. Capability-based security can give each agent only the credentials required for the current skill. Sandboxes can allow broad exploratory behaviour without broad external authority. Transactions and idempotency can make retries safe. Checkpoints and reversible operations reduce the cost of error. These mechanisms do not make the model more intelligent; they make its generality more usable.

Evidence thresholds create bounded epistemic generality. A research agent can explore widely but may publish a strong conclusion only after source triangulation or domain review. A coding agent can edit broadly in a branch but may merge only after tests and approval. A healthcare assistant can synthesise information but may not issue a diagnosis or alter a record without qualified oversight. The same generative competence supports different levels of authority because the harness separates exploration from commitment.

7.4 The Pareto frontier and architecture selection

There is no single architecture that maximises the triangle for every application. The goal is a local Pareto-efficient design: under the target distribution, no dimension can be improved without reducing another. A frontier model on every step may be dominated by routing. A rigid workflow may be dominated by one adaptive exception path. A ReAct agent may be dominated on a recurring

database task by a schema-aware DBTable Skill, even though ReAct remains better on an unfamiliar investigation. These comparisons are meaningful only when model, tools, authority, budgets, and acceptance criteria are controlled.

Architecture selection should begin with uncertainty decomposition. Which parts of the task are stable? Which inputs are unstructured? Which claims can be checked? Which actions are reversible? Which context changes the correct interpretation? Which decisions involve values or law? Which variability recurs often enough to engineer? The answers determine where fixed structure, model interpretation, code, retrieval, database abstractions, evaluation, and human judgment belong. Starting from a desired agent persona usually grants too much discretion too early.

Evaluation should use local ablations before universal leaderboards. Compare the simplest ReAct baseline with the proposed orchestration on the same tasks, model, tools, and budget. Remove a skill family, context reducer, reviewer, or extra agent to see whether it earns its cost. Measure not only final answers but routing errors, context omissions, repeated work, execution variance, recovery, and review burden. If a specialised harness improves one meaningful workflow, that is a valid engineering result even when it is not a generally stronger agent.

Trustworthiness should also be treated as a vector. A system can be robust but opaque, transparent but insecure, fair on average but inaccessible, efficient but environmentally costly at scale, or compliant on paper but impossible to contest. Architecture should record the evidence supporting each claim and the trade-offs accepted by accountable people. One composite trust score would conceal the very tensions that governance needs to see.

7.5 A constructive reconciliation

The three ambitions can be reconciled through progressive structuring. Begin with broad model capability inside a restrictive environment and complete tracing. Observe real task variation. Convert recurring successful behaviour into explicit orchestration,

tests, code, database-aware operations, or AKUs. Route routine cases toward the cheaper and more verifiable path while preserving escalation to broader interpretation when the local abstraction is insufficient. Use interaction evidence to propose improvements offline rather than granting uncontrolled online self-modification.

This process increases trustworthiness because assumptions, authority, and checks become explicit. It increases efficiency because the system stops rediscovering the same procedure and stops replaying unnecessary context. It preserves generality through composition, fallback, and a portfolio of agents rather than through one unconstrained planner. The result is neither a static workflow nor a universal agent. It is a harness that can grow its bounded competence while exposing where it remains uncertain.

The deepest insight is that generality should reside more in composition and evolution than in unrestricted local behaviour. Human organisations are broadly capable because specialised roles, tools, procedures, and governance can be recombined. Agent systems can adopt the same strength while making dependencies and evidence more explicit. They must also avoid the same pathologies: duplicated work, diffusion of responsibility, stale procedures, and metric gaming. Observability, named state, and controlled promotion are therefore not administrative overhead; they are part of generality that remains trustworthy.

The next part examines two concrete research directions. AchillesAgentLib uses MainAgent and orchestration skills to coordinate three executable families under scoped context, sessions, and a model-access policy boundary. MRP-VM replaces the generic skill at its core with the Agentic Knowledge Unit, a knowledge-and-execution object managed inside a runtime that can host several agents and improve through a governed sleep phase. Both begin from the same premise: do not promise an agent that is best at everything; build harnesses that make selected useful work cheaper, clearer, and safer.

PART III BEYOND

CHAPTER 8

AchillesAgentLib: Orchestration Skills for Bounded Competence

The objective is not a universal agent. It is a harness that makes recurring work cheaper, safer, and easier to inspect.

8.1 An engineering position, not a universal benchmark claim

AchillesAgentLib is a project-specific agent runtime developed around the ACHILLES and AssistOS research direction. Its purpose is pragmatic: when a class of requests recurs, the system should be able to transform repeated language-model improvisation into a capability with clearer context, execution semantics, review, and operational evidence. The architecture belongs to the broader 2026 movement from prompt engineering toward harness engineering, but it does not assume that one harness pattern should dominate every domain.

The central position is deliberately modest. AchillesAgentLib does not attempt to beat the ReAct pattern in general. ReAct remains an excellent exploratory mechanism when the problem is unfamiliar, observations redefine the next step, or no stable procedure exists. The Achilles question is narrower and more engineering-oriented: where can recurring work be made more trustworthy and more efficient by moving part of the process into explicit orchestration and executable components? When such an opportunity exists, the aim is to implement and validate it quickly rather than to preserve generality as an abstract virtue.

This position changes how success should be measured. A broad leaderboard that mixes browsing, games, coding, and arbitrary reasoning may say little about a specialized customer workflow, database-reporting process, or project-specific research operation. The relevant evidence is local: accepted result quality, latency, total inference and tool cost, run-to-run variance, amount of human review, context size, failure severity, and recoverability. A specialised agent can be a meaningful success if it improves these measures on the problem for which it was designed, even when a general ReAct agent remains better elsewhere.

The essence of the architecture is progressive stabilisation. Broad model interpretation is used to discover intent and handle novelty. Recurring coordination is moved into orchestration skills. Stable execution is assigned to code or database-aware components. Generated code is isolated and evaluated before it receives authority. Context is reduced to the smallest useful frame, with explicit escalation when the local frame is insufficient. The design goal is not maximal autonomy, but a practical increase in bounded competence.

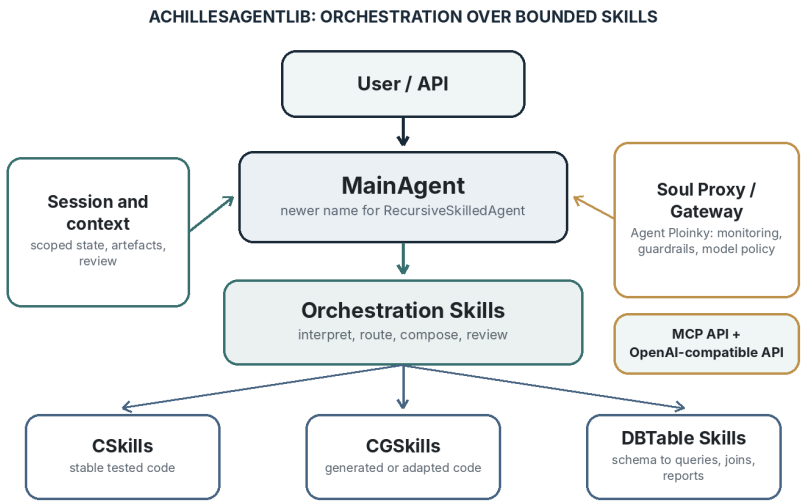


Figure 8.1. MainAgent coordinates orchestration skills, executable skill families, scoped context, and controlled model access.

8.2 MainAgent and the orchestration boundary

The central entry point is currently known as RecursiveSkilledAgent and will be named MainAgent in newer versions. The newer name is clearer because the component is not itself a promise that every task should recurse through a hierarchy. It is the principal coordination boundary. MainAgent receives a request, binds it to a session, determines whether a named capability was requested, discovers an appropriate orchestration path when necessary, prepares the relevant context, and applies the review and model-access policies associated with the operation.

MainAgent provides one public surface while preserving distinct semantics below it. A caller may invoke an explicit orchestration skill, which behaves more like a stable service contract, or submit an open request that requires discovery. The explicit mode reduces routing uncertainty and is preferable for integrated applications. The open mode retains assistant-like flexibility and provides a fallback when no known capability matches. Generality therefore appears as controlled routing between known and exploratory regimes, not as the assumption that every invocation should begin with free planning.

The core reusable object at this layer is an orchestration skill. It describes how a bounded operation should be interpreted and coordinated, which executable families may be used, what context is required, what outputs are expected, and where review or escalation belongs. An orchestration skill may compose other orchestration skills, but recursion is a means rather than an identity. The important property is that the runtime can separate coordination from execution and can test them independently.

This separation is a trust boundary. The language model may propose a route or fill a semantic gap, while the harness decides which execution family is authorised, which credentials are exposed, which artefacts are returned to context, and which evidence closes the operation. A single JSON tool abstraction would conceal important differences between stable code, generated code, and structured-database work. MainAgent preserves a common interface

without pretending that these operations have the same risk or validation regime.

Table 8.1. AchillesAgentLib separates unified orchestration, executable skill families, scoped state, and model-access policy.

Runtime object	Primary role
MainAgent (formerly RecursiveSkilledAgent)	Unified entry, discovery, session binding, context and review
Orchestration Skill	Coordinates bounded work and delegates to executable families
CSkill	Stable, tested and versioned code execution
CGSkill	Generated or adapted code under sandboxing and evaluation
DBTable Skill	Schema abstraction for queries, joins, aggregation and reports
Session and context	Scoped state, artefacts, provenance, retention and escalation
Soul Proxy / Soul Gateway	Model-access monitoring, guardrails, routing, budgets and logging
Agent Ploinky	Agent or policy service exposed through MCP and OpenAI-compatible APIs

8.3 The three executable skill families

CSkills, or Code Skills, package stable executable behaviour. They are appropriate when repeated use has revealed a procedure that can be implemented, tested, versioned, and executed more reliably than a fresh language-model interpretation. A CSkill may still receive parameters extracted by a model, but its core operation is

conventional code. This improves reproducibility and cost predictability, and it permits familiar software assurance: unit tests, static analysis, typed interfaces, dependency review, permissions, and deterministic error handling.

CGSkills, or Code Generation Skills, are used when the operation requires code that cannot be fully fixed in advance. The model generates or adapts a program for the current task, and the harness treats that program as an untrusted artefact. It must run inside a constrained environment with explicit inputs, resource limits, dependency policy, tests, and result capture. CGSkills occupy an important middle ground. They preserve more local generality than CSkills while avoiding the token cost and copying errors that arise when large structured transformations are performed entirely through conversational tool calls.

DBTable Skills specialise the runtime for relational and tabular data. They begin from an abstraction of the database schema: tables, fields, types, keys, relationships, permitted operations, and domain annotations. From this representation, the system can create the capability to formulate queries, connect related tables, aggregate data, produce reports, and explain how a result follows from the schema and selected records. The abstraction is intended to reduce repeated schema discovery and to constrain generated operations. A database engine performs the exact relational work; the model interprets the user's intent and the meaning of the results.

The three families are intentionally distinct because they make different promises. CSkills promise stable implementation. CGSkills promise adaptive implementation under isolation and verification. DBTable Skills promise schema-aware access to structured data under explicit permissions and query semantics. An orchestration skill can combine them, for example using a DBTable Skill to retrieve evidence, a CSkill to compute a regulated metric, and a CGSkill to generate a one-off visualisation. The harness can then apply family-specific logging, context reduction, review, and security rather than one undifferentiated tool policy.

8.4 Context, sessions, and the classical Agent Skills comparison

Context is the recurring research problem in AchillesAgentLib. The architecture attempts to give an orchestration skill the local information it needs instead of replaying a global conversation and every available capability. This can reduce cost, improve tool selection, and protect unrelated data. It can also remove the nuance that explains why a request is exceptional. The important design is therefore not aggressive compression alone, but a layered context policy: a small default frame, named references to durable artefacts, provenance for retrieved knowledge, and an explicit path for requesting broader context when local confidence is insufficient.

Sessions provide the operational container for this policy. A session can hold request history, selected context, intermediate artefacts, skill decisions, model usage, errors, approvals, and review outcomes. It should not be treated as unlimited conversational memory. Different elements require different retention, access, and validity rules. A database result may become stale; a user preference may require consent; generated code may need quarantine; a review decision may need long-term audit. The session is valuable when it makes these distinctions visible and prevents state from leaking between unrelated agents or users.

External Anthropic-style Agent Skills are referred to here as classical Agent Skills. Their current form packages instructions, scripts, and resources that a broad agent can discover and load progressively. Project experiments indicate that this form currently fits most naturally inside a ReAct-style agent: the general loop retains the global task context while the skill supplies local procedural guidance. AchillesAgentLib is intentionally learning a different style in which orchestration skills carry more explicit coordination and delegate to executable families. The two approaches should not be presented as direct substitutes because they allocate context and execution meaning differently.

The Achilles approach can outperform a general ReAct loop on particular repeated tasks because it reduces tool exposure, repeated

deliberation, and context traffic. Its risk is equally clear: by localising the context, it may lose global constraints, subtle user intent, or cross-task dependencies. This is not an implementation detail to be hidden. It is the central customisation and research problem. Every orchestration skill should be evaluated not only for speed and correctness, but for what information it excludes, how that exclusion is detected, and when it escalates back to a broader reasoning context.

8.5 Soul Proxy, Agent Ploinky, and controlled model access

Soul Proxy, also described as Soul Gateway, is conceived as an Agent Ploinky placed between agent runtimes and model providers. Its role is to turn model access into a governed service rather than a collection of direct API calls. It can monitor requests and responses, enforce model and data policies, apply guardrails, route among providers or local models, constrain budgets, support caching, record usage, and mediate what information may leave a trust domain. Centralising this boundary also makes it possible to change model infrastructure without rewriting every orchestration skill.

An Agent Ploinky exposes two complementary integration surfaces. An MCP-facing interface allows compatible agents to discover and invoke approved capabilities through standard tool semantics. An OpenAI-compatible API interface allows existing applications and agent frameworks to treat the governed service as a familiar model or agent endpoint. Compatibility is not the same as unrestricted equivalence: identity, tenant, policy, quotas, trace correlation, and data-handling rules must remain explicit at both surfaces.

The gateway is also a natural point for context protection and observability. It can redact or tokenise sensitive fields, apply compression policies, reject requests that exceed an authority boundary, attach provenance, and connect model calls to session-level traces. It cannot determine the full meaning of every request and should not become a single opaque policy oracle. Local

skills still need domain checks, and high-consequence actions still require appropriate validation or human authority. The value of Soul Proxy is consistent mediation, not magical safety.

8.6 How AchillesAgentLib should be evaluated and evolved

AchillesAgentLib should be evaluated through task-specific benchmark packages derived from the applications it intends to support. Each package should include representative requests, difficult edge cases, hidden perturbations, unacceptable outcomes, expected evidence, cost and latency budgets, and a ReAct-style baseline using comparable models and tools. The objective is not to prove that the architecture produces a generally superior agent. It is to determine whether a particular orchestration and skill configuration earns its additional structure.

The most informative experiments are ablations. Compare global context with local context plus escalation. Compare a broad tool catalogue with skill-scoped exposure. Replace a CSkill with model-mediated execution to measure the value of stabilisation. Compare a CGSkill with repeated tool calls to examine token traffic and copying error. Compare a DBTable Skill with open SQL generation to measure schema errors, policy violations, reporting quality, and explanation. These experiments can reveal whether improvements come from the model, the context policy, the executable family, or the reviewer.

The expected development path is progressive. A new task may begin as an instrumented ReAct prototype. Repeated traces reveal the stable coordination pattern and common failures. That pattern becomes an orchestration skill. Stable operations become CSkills; variable transformations may temporarily become CGSkills; structured-data work becomes a DBTable Skill. The change is promoted only when local evaluation shows a useful improvement and when the context boundary does not erase critical nuance. In this sense the harness grows by converting experience into explicit structure.

AchillesAgentLib therefore represents a disciplined engineering thesis: use language models where interpretation is genuinely valuable, but do not require them to rediscover every recurring procedure. Preserve ReAct as an open path rather than as the universal control law. Build narrow competence where evidence justifies it. Make context selection, execution semantics, model access, and review visible enough to improve. MRP-VM extends this thesis by asking whether the executable knowledge itself can become the main substrate of a multi-agent environment and can be refactored through controlled periods of self-research.

CHAPTER 9

MRP-VM: AKUs, Self-Research, and Evolvable Harnesses

A harness should not merely remember conversations. It should be able to convert experience into inspectable knowledge and better execution under controlled change.

9.1 Why another runtime model

MRP-VM begins from a direct confrontation with the limits of standard ReAct patterns and contemporary tool use. Long transcripts accumulate schemas, observations, corrections, and duplicated prose; API costs rise as the same history is repeatedly processed. Agents repeat actions, hallucinate state, lose logical consistency in long-form generation, simplify complex specifications, and miss constraints buried in the middle. Tools extend capability, but the loop still asks natural language to copy, remember, and coordinate too much of the program.

The transcript is therefore an inadequate universal representation of memory, program state, and explanation. It orders events without exposing dependencies, mixes instructions with hypotheses and outputs, and does not identify which claims remain valid after change. Summaries reduce size by creating another lossy bottleneck. MRP-VM instead represents long-running work through named executable knowledge objects and explicit intermediate values, while conversation remains one interface rather than the whole machine.

The philosophical premise is that natural language is an interpretation medium, not an automatically authorised instruction set. A statement may be ambiguous, provisional, contested, or perspective-dependent. Execution therefore requires an interpreter that declares how language becomes an operation, what evidence it

consumes, what result it produces, and how that result is qualified. Explanation concerns this visible mapping from request to interpretation, evidence, operation, and result—not an unverifiable claim to reveal private model reasoning.

MRP-VM rejects the assumption that one agent can be good at everything. Useful systems instead combine bounded agents with a harness that can grow under evidence. Research, database, and software agents may share a runtime and default knowledge while retaining separate working knowledge, authority, models, budgets, and public identities. Generality comes from a governed portfolio and forkable knowledge, not unrestricted competence granted to one model.

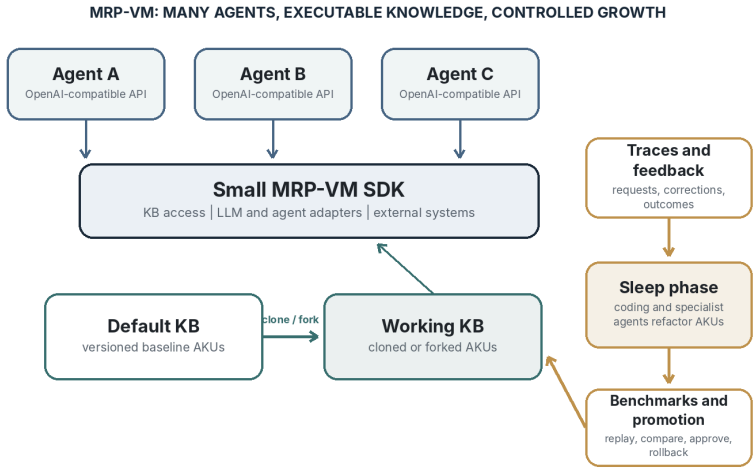


Figure 9.1. MRP-VM hosts several API-compatible agents over a small SDK, layered AKU knowledge bases, and a governed sleep loop.

9.2 AKUs and executable natural-language programs

The core abstraction is the Agentic Knowledge Unit, or AKU. An AKU contains more than a prompt. It can contain the natural-language specification, the interpretation or execution semantics, interfaces to language models, other agents, code, tools, and external systems, and metadata such as version, provenance, authority, context requirements, tests, evaluator expectations, and prior performance. The exact representation can evolve, but the conceptual boundary is essential: knowledge and the method by which that knowledge becomes operational are kept together.

Replacing the generic skill with an AKU changes what the runtime can inspect. An AKU can declare its interpreter, the named values it reads and writes, the evidence that qualifies them, and the meaning of failure. It can be cloned, forked, combined, compared, or retired as a versioned object. Natural language remains central for intent and domain meaning, but it is bound to an operational contract.

Interpreter bindings are heterogeneous by design. An AKU may use a language model, stable code, a database, a symbolic solver, a remote OpenAI-compatible agent, or several interpreters. The runtime preserves these distinctions because reliability, cost, and explanation differ. It records which interpreter produced each value and under which configuration, separating model errors from code, tool, data, and orchestration errors.

Executable natural-language programs can use SOP-like structures with named commands, values, dependencies, and acceptance conditions. A value may represent documents, a schema mapping, an interpreted requirement, an artefact, or a verified result. Strong context references keep a current value available; weak references may use a summary or identifier. Explicit dependencies permit selective recomputation and different context policies across the program.

Qualification belongs to the value itself. Results may be observed, inferred, generated, tested, reviewed, disputed, stale, or rejected, with links to evidence and evaluators. This supports process explainability: users and auditors can see what ran, what knowledge

it used, where model interpretation entered, and where uncertainty remains, without exposing hidden chain-of-thought.

9.3 A small SDK over an executable knowledge base

MRP-VM aims to keep the fixed runtime small. The SDK provides the primitives needed to interact with knowledge bases, invoke language models or other agents, communicate with external systems, execute approved code, manage sessions and values, and record traces. It should not contain every useful business procedure. The more domain behaviour is hardwired into the runtime, the harder it becomes to inspect, fork, improve, and specialise. The VM supplies execution discipline; the knowledge base supplies most useful behaviour.

Useful code therefore lives in the knowledge base as part of AKUs. A code-bearing AKU can package source, dependencies, tests, constraints, and a natural-language account of its capability. Model-facing AKUs can package prompts, examples, context selectors, schemas, and evaluators. Composite AKUs connect them, allowing code, instructions, documentation, and benchmarks to evolve as one operational unit.

A Default Knowledge Base provides versioned baseline AKUs distributed with the environment or approved by an organisation. A Working Knowledge Base contains the units active for one agent or application. Default AKUs can be cloned or forked locally without altering the shared baseline. Their known origin and local divergence remain visible, so later versions can be compared, selectively merged, or rejected.

This layering supports controlled promotion and rollback. Candidate AKUs are tested in an experimental branch and promoted only when local benchmarks improve without violating safety or context constraints. A successful local unit may later be proposed upstream but must not propagate silently. Provenance, signatures, dependency pinning, and migration rules matter because executable knowledge is also a software supply chain.

The knowledge base must not collapse into a vector store of decontextualised prose. Retrieval may discover candidates; the runtime decides whether and how they execute. Each AKU retains identity, contract, evidence, authority, and compatibility constraints. Relevance is not permission, and similarity is not validity.

Table 9.1. MRP-VM treats executable knowledge, working state, agent identity, and improvement evidence as versioned runtime objects.

MRP-VM object	Execution meaning
Agentic Knowledge Unit (AKU)	Versioned instructions, interpreter bindings, interfaces, tests, provenance, and policy
SOP command or named value	Operation, dependency, intermediate result, or qualified state
Default Knowledge Base	Shared versioned baseline AKUs
Working Knowledge Base	Cloned or forked AKUs local to one agent or application
Small runtime SDK	KB, model and agent adapters, external access, state, and tracing
Agent instance	Isolated identity, policy, budget, working KB, and API endpoint
Interaction corpus	Curated cases, corrections, feedback, and execution evidence
Sleep phase	Offline refactoring, replay, benchmarking, and governed promotion

9.4 Many agents in one environment

One MRP-VM installation is intended to manage several agents in parallel. Each has an identity, policy, Working Knowledge Base, model portfolio, budget, and external capabilities. Agents are

scheduled independently while sharing infrastructure and selected Default AKUs, enabling consistent tracing, governance, comparison, and composition across domains.

Agents are exposed through OpenAI-compatible APIs so existing applications can call them through familiar semantics. The endpoint represents an agent service, not a raw model: a request may execute AKUs, query knowledge, call peers, and return artefacts or evidence. Responses should therefore carry agent identity, session correlation, policy metadata, and trace links where appropriate.

Isolation remains necessary. Working Knowledge Bases must not leak private state; credentials and resources must be scoped; shared AKUs require version pinning; and cross-agent calls must be explicit, authenticated, and traceable through OpenAI-compatible APIs, MCP, or another protocol.

The architecture supports deliberate specialisation. Low-cost agents can execute high-volume narrow work with compact contexts and stable AKUs. More capable models can handle ambiguity or propose sleep-phase changes. Policy agents monitor access; coding agents refactor executable units. These are services with distinct authority, economics, and evaluators, not theatrical personas.

Generality therefore emerges from the environment. New agents are instantiated by selecting and forking AKUs, defining policy, and exposing endpoints. Trusted foundations can be shared while local behaviour evolves. When no configured unit is adequate, the system escalates to broader interpretation or human work instead of pretending universal competence.

9.5 Context minimisation, qualification, and explainability

Context is MRP-VM's central optimisation variable. Every AKU should declare what it needs instead of inheriting the whole conversation. The runtime assembles a local frame from the active instruction, named values, selected artefacts, policy, and minimal interfaces, while recording deliberate omissions. Token use becomes attributable to a capability rather than to session length.

Minimisation remains conditional. Compression can erase tone, exceptions, cross-cutting constraints, or the reason for a decision. AKUs therefore need context escalation: retrieve the source behind a summary, inspect a parent objective or named value, or ask another agent or human to resolve ambiguity. The core research problem is minimising context without losing nuance that changes the interpretation.

Named state counters lost-in-the-middle effects because critical values no longer depend on transcript position. Programs refer directly to current schemas, requirements, evidence, or contradictions. Summaries become views over preserved artefacts rather than replacements. Explicit dependencies trigger selective recomputation, while references avoid copying large artefacts through every model turn.

Explainability follows execution. The system can report selected AKUs, assembled context, interpreters, produced values, checks, omissions, and human or agent decisions. This process account supports debugging, contestability, and compliance without claiming that generated reasoning text faithfully exposes neural computation.

9.6 Self-discovery and harness self-research

MRP-VM treats interaction as research evidence. Requests, corrections, rejected answers, feedback, latency outliers, tool failures, and review decisions form a curated corpus of bounded cases. Privacy, consent, minimisation, and retention apply before use. The objective is not indefinite transcript storage but evidence of missing or inadequate AKUs, context policies, interpreters, and agent boundaries.

The harness analyses this corpus into a failure taxonomy. Specification loss may require a named objective; database mistakes, a DB-aware AKU; costly tool copying, code execution; repeated escalation, a wider context frame; and contradictory long-form answers, decomposition or independent checking. Self-research investigates the harness as a system rather than letting the serving agent rewrite itself online.

Evidence can motivate a new AKU, a split or merge, a revised prompt, interpreter binding, context selector, test, router rule, model policy, or specialised agent. Each proposal states its evidence and expected trade-off. Fewer tokens are not progress if nuance disappears, and better benchmark scores may not improve the operational frontier when cost rises excessively.

Candidates never mutate the serving knowledge base directly. They enter an experimental branch for replay against historical cases, hidden tests, adversarial variants, and production samples. Risk-based human or policy review prevents feedback from becoming an unbounded self-modification channel and limits poisoning, overfitting, and metric gaming.

The harness thus becomes a laboratory for its own operation. It can ask why a task was expensive, why a specification or context selection failed, and whether another AKU would have changed the result. This is more ambitious than prompt tuning but more constrained than autonomous recursive self-improvement: the unit of change is an inspectable runtime artefact under evaluation.

9.7 The sleep phase

The sleep phase is an offline period in which capable but expensive coding or specialist agents analyse recent interactions and propose consolidation. Online serving should remain stable and conservative; sleep can spend more computation comparing traces, versions, duplicated logic, and candidate improvements without exposing users to experimental behaviour.

A sleep cycle may refactor code, prompts, instructions, context selectors, tests, knowledge, and interfaces inside AKUs. It may merge units that travel together, split an overloaded unit, stabilise a repeatedly generated program as code, update a schema abstraction, or add an evaluator. Coding agents address executable structure; domain experts and specialised agents remain necessary for semantic and organisational validity.

Every consolidation is replayed and benchmarked against quality, consistency, specification adherence, hallucination, context size,

latency, cost, and review burden. Regression cases and unseen variants matter more than the motivating conversations alone. Promotion follows an explicit reversible rule; sleep cycles continue only while the declared frontier improves and the research budget permits.

9.8 Evaluation, governance, and limits

MRP-VM should be evaluated on concrete task families, including online performance and offline improvement cost. Comparisons should include a fixed baseline, an equivalent-tool ReAct agent, an AKU agent before sleep, and the candidate after sleep. Trace-level analysis should attribute gains to context selection, interpreter choice, code, reuse, verification, or stronger models rather than making universal claims.

Ablation is essential. Remove named values, restore full history, prevent forking, replace sleep with manual edits, or remove expensive coding agents. Such experiments reveal whether explicit state, local context, adaptation, and offline consolidation provide causal value rather than architectural decoration.

The design has serious limits. AKUs can encode wrong interpretations; sleep can overfit, absorb malicious feedback, or optimise weak evaluators; coding agents can introduce vulnerabilities; compact contexts can omit decisive nuance; and shared installations create isolation and scheduling risks. Governance therefore requires provenance, access control, evaluator diversity, red-team cases, staged promotion, audit, and rollback.

MRP-VM is a proposal for cultivable agent infrastructure. Several bounded agents share, fork, execute, and improve explicit knowledge; context becomes a managed resource, explanation a property of visible execution, and learning governed refactoring rather than online mutation. Its testable hypothesis is that a harness which consolidates recurring failures can become more trustworthy and efficient precisely because it does not claim competence at everything.

CHAPTER 10

Beyond 2026: Growing Evidence-Bearing Agentic Systems

The frontier is not autonomy without structure. It is the controlled growth of systems that can act, explain, and improve within declared boundaries.

10.1 The convergence ahead

The most important convergence of 2026 is among models, harnesses, protocols, execution environments, and evidence. Foundation models continue to improve, but industrial systems increasingly separate the model from a durable control layer and from sandboxed compute. Protocols such as MCP and A2A standardise connections. Tracing conventions make episodes portable. Evaluation moves closer to production. Research now names context policy, intervention recording, verification, and failure attribution as first-class harness responsibilities. The agent is no longer a prompt loop; it is a distributed software system whose semantic core remains probabilistic.

This convergence will not produce one canonical architecture. Tasks differ in observability, reversibility, regulation, data structure, and economic value. A coding agent benefits from executable tests and branch isolation. A research agent needs source provenance and contradiction management. A database agent needs schema and permission semantics. A public-facing assistant needs identity, disclosure, and redress. The durable architecture will therefore be a family of runtimes able to compose broad interpretation with narrow, specialised interpreters.

The likely replacement for the universal-agent narrative is a portfolio of bounded competence. Organisations will operate several

agents, each with a known purpose, authority, evidence regime, and cost envelope. Generality will emerge through routing, shared protocols, reusable knowledge, and controlled escalation. The open-ended agent remains useful as an exploratory layer and as a way to discover new procedures. Stable value migrates into explicit workflows, skills, AKUs, code, databases, and validators.

Context will be the main operating substrate of this portfolio. The critical question will not be how many tokens a model can technically accept, but which information deserves attention at a particular decision. Context policies will select, compress, reference, and protect evidence according to task state and budget. Research such as ReSum, ContextBudget, Less Context Better Agents, and evolving-context systems shows that this policy can change both reliability and cost. The remaining challenge is semantic: compression must preserve the nuance that alters action and expose what was omitted (Lodha et al., 2026; Wu et al., 2025, 2026).

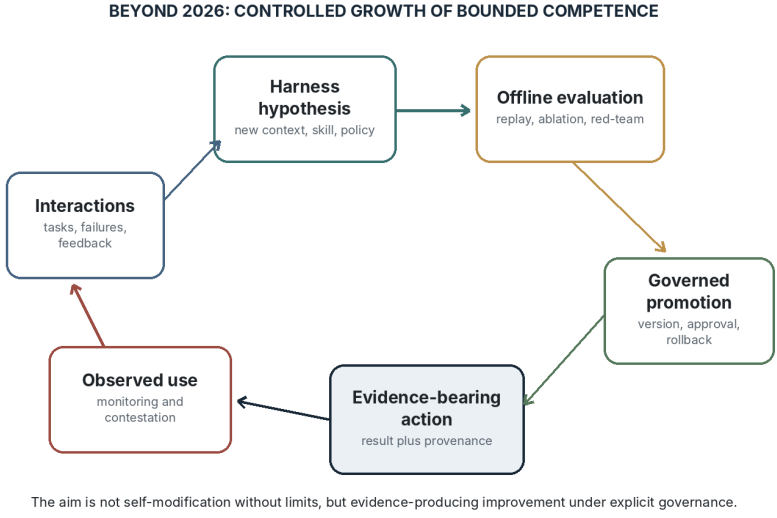


Figure 10.1. Evidence-bearing agent systems improve through offline hypotheses, evaluation, governed promotion, and monitored use.

10.2 From static harnesses to controlled learning

Most deployed harnesses are configured manually and remain largely static between releases. A growing research line treats traces

as material for optimisation. Agent-training systems optimise behaviour from trajectories; MemoHarness retrieves prior diagnoses and adapts several harness dimensions to new cases; industrial platforms optimise prompts and evaluators; coding-agent teams use structured artefacts and independent QA to sustain longer work. These developments suggest that the harness itself will become a learned and versioned artefact, not merely the infrastructure around learning (Huang et al., 2026).

Controlled learning must remain distinct from uncontrolled self-modification. Production interactions contain sensitive data, adversarial content, organisational exceptions, and evaluator errors. A successful outcome can include hidden failures, and a dissatisfied user can be wrong. Improvement therefore requires curation, provenance, privacy, counterexamples, held-out evaluation, and accountable promotion. MRP-VM's sleep phase is one proposed mechanism: use expensive specialist agents offline to refactor executable knowledge, then promote only changes that improve a declared evidence package.

The strongest systems will learn at several timescales. A single execution can adapt through local replanning. A session can preserve user-approved state. A team can update a Working Knowledge Base after review. An organisation can publish a new Default Knowledge Base after broader evaluation. A model provider can train new weights from aggregated experience. These levels should not collapse into one automatic feedback loop because their authority, privacy, and blast radius differ.

10.3 Verification, explainability, and evidence-bearing action

The field will move from answer confidence toward evidence-bearing execution. A result may carry source provenance, test outcomes, type checks, database queries, policy decisions, model and prompt versions, human approval, or a formal certificate. These evidence types have different scope. A test covers declared cases; a proof covers a formal model; a human approval assigns

responsibility; a source supports a claim but may itself be wrong. A trustworthy runtime preserves these distinctions instead of compressing them into one confidence score.

Process explainability will matter more than raw reasoning disclosure. Users and auditors need to know what task the system understood, which data and policies were used, what operations occurred, which parts were generated or inferred, what checks ran, what was omitted from context, and where uncertainty remained. They do not need an unfiltered chain-of-thought, which may be unfaithful and may expose sensitive information. Named state, AKU or skill identities, interpreter bindings, and trace-linked artefacts provide a stronger account of operational causality.

Selective recomputation strengthens contestability. When dependencies are explicit, a reviewer can replace a disputed source, policy assumption, interpreter, or model and observe which results change. This is valuable for science, regulation, and ordinary debugging because disagreement becomes localisable. It also prevents one correction from forcing the system to regenerate an entire long narrative, where unrelated claims may change accidentally. MRP-VM's named values and qualification states make this principle concrete; graph runtimes and workflow engines express related ideas through durable state.

Verification will remain incomplete. No evaluator can anticipate every future environment or eliminate correlated error among models. Safety claims must therefore be configuration-specific, time-bounded, and linked to monitoring. A system that passed an evaluation suite can degrade when models, tools, data, prices, policies, or adversaries change. Trustworthiness is a maintained relation between evidence and reliance, not a permanent property awarded at release.

10.4 A research and industrial agenda

The first priority is valid long-horizon evaluation. Benchmarks should measure repeated reliability, recovery, context behaviour, cost, latency, and policy compliance under changing state. They

should publish complete system configurations and episode packages, audit their scorers, and preserve hidden cases. Local application benchmarks deserve the same methodological discipline as broad leaderboards. AI Agents That Matter and recent harness-engineering work make clear that a score without cost, reproducibility, and system description is weak evidence (Kapoor et al., 2024; Zhong and Zhu, 2026).

The second priority is secure, explainable context. Research must combine provenance, trust domains, typed instructions, relevance, privacy, memory promotion, revocation, and resistance to indirect prompt injection. Context reduction should be evaluated for both omission and distraction. Tool and protocol ecosystems should support progressive discovery so thousands of capabilities do not become thousands of unreviewed instructions. The long-term goal is a context architecture in which untrusted content cannot silently acquire authority and omitted nuance can be recovered.

The third priority is cost-aware orchestration. Schedulers should optimise accepted quality, elapsed time, monetary cost, energy, context, and risk together. They should learn when to plan, react, parallelise, use code, call another agent, compress history, escalate to a stronger model, or stop. This is a compiler problem over heterogeneous interpreters and a governance problem over delegated authority. The success criterion is not maximum tool use, but useful work at the best justified point on the local Pareto frontier.

The fourth priority is agent identity and transaction semantics. Agents need authenticated identities, scoped capabilities, expiry, quotas, and trace correlation. Side-effecting actions need idempotency, atomicity where possible, compensating actions, and approvals at meaningful boundaries. Multi-agent calls need cancellation, confidentiality, responsibility, and clear ownership of the final artefact. MCP and A2A can transport capabilities and messages; they do not by themselves define who is accountable for a composed action.

The fifth priority is progressive formalisation. Recurring successful behaviour should become reusable orchestration, tests,

code, schema abstractions, validators, or AKUs. The process should preserve a route back to broader interpretation when exceptions appear. AchillesAgentLib expresses this through orchestration skills and three executable families. MRP-VM expresses it through executable knowledge, forkable working bases, named state, and sleep-phase consolidation. Their value will be determined by empirical local improvements, not by conceptual richness alone.

The sixth priority is institutional design. Human roles will shift from manually producing every artefact toward defining objectives, constructing evaluators, reviewing exceptions, authorising consequences, and maintaining knowledge. Human oversight must be substantive rather than a ceremonial click after an opaque process. European trustworthy-AI policy reinforces this direction by treating human agency, transparency, robustness, fairness, accountability, and redress as properties of the complete sociotechnical system.

Table 10.1. The frontier combines bounded competence, explicit evidence, controlled learning, and human-centred governance.

Frontier direction	Research and engineering question
Bounded competence	Which task families justify stronger assumptions and specialised execution?
Context operating system	How can context be minimised without losing decisive nuance?
Evidence-bearing action	Which evidence supports each claim, operation and authority transition?
Controlled harness learning	How can traces improve prompts, code, AKUs and policy without unsafe mutation?

Frontier direction	Research and engineering question
Interoperable agent services	How should MCP, A2A and OpenAI-compatible APIs carry identity and responsibility?
Human-centred governance	Where must people specify, approve, contest, and remain accountable?

10.5 The essence of Agentic AI 2026

The central fact of 2026 is that agents are becoming software systems rather than clever prompts. ReAct remains the alphabet of model-environment interaction, but production grammar now includes state graphs, context policies, sandboxes, typed tools, skills, memory, schedulers, gateways, traces, evaluation, and human authority. The model remains powerful and uncertain. The harness determines how that uncertainty enters the world.

The central limitation is that placing a language model in a loop does not make it a reliable planner, faithful witness, or consistent long-form program executor. Long trajectories amplify error and cost. More context can reduce relevance. More agents can increase coordination failure. Tool use can ground computation while expanding attack surfaces and context. The response is not to eliminate model interpretation, but to stop treating natural-language continuation as the only representation of state and control.

The central design problem is the relationship among trustworthiness, efficiency, and generality. Fixed workflows are stored knowledge about stable tasks, not the enemy of intelligence. Open planning is valuable at genuine novelty, not the enemy of trust. Good systems combine both and admit where each fails. They use broad models at semantic boundaries, narrow interpreters for exact operations, explicit evidence for claims, and human authority for decisions whose significance cannot be legitimately delegated.

The central ACHILLES contribution is an engineering stance aligned with the European ambition for AI that is lighter, clearer, and safer. Do not promise one agent that is best at everything. Identify concrete problems where stronger structure can improve trust and efficiency. Keep the path to broader reasoning when the structure is inadequate. Measure context loss, not only context savings. Expose model access, evidence, and review. Treat improvement as a governed lifecycle rather than a permanent prompt.

AchillesAgentLib and MRP-VM express two stages of this stance. AchillesAgentLib coordinates orchestration skills over CSkills, CGSkills, and DBTable Skills, with MainAgent, sessions, and Soul Proxy providing bounded entry, state, and model policy. MRP-VM moves useful behaviour into AKUs that combine knowledge, interpretation, interfaces, and evaluation; it supports several agents, forkable knowledge, named state, and a sleep phase for controlled self-research. Neither architecture should be evaluated as a claim to universal superiority. Their hypothesis is that a growing harness can improve selected tasks more defensibly than repeated unconstrained improvisation.

Agentic AI is therefore best defined as the architecture of constrained delegation and cultivable competence. A model may interpret, propose, explore, and compose. The runtime decides what becomes action, evidence, memory, executable knowledge, or institutional fact. The future belongs neither to the most rigid workflow nor to the agent that appears most independent. It belongs to systems that can produce valuable work, preserve uncertainty honestly, learn from correction without surrendering control, and show why their actions deserved trust.

References

Sources include peer-reviewed papers, current preprints, official standards and policy documents, and platform documentation. Industrial and regulatory status was verified to 23 July 2026 where relevant. Project-specific chapters identify proposals under development and do not imply external validation.

AI HLEG. 2019. Ethics Guidelines for Trustworthy AI. High-Level Expert Group on Artificial Intelligence, European Commission.

AI HLEG. 2020. Assessment List for Trustworthy Artificial Intelligence (ALTAI) for Self-Assessment. European Commission.

Anthropic. 2024. Building Effective Agents. Anthropic Engineering.

Anthropic. 2025a. Code Execution with MCP: Building More Efficient AI Agents. Anthropic Engineering, 4 November 2025.

Anthropic. 2025b. Agent Skills Specification and Official Skills Repository. Anthropic.

Anthropic. 2026a. Harness Design for Long-Running Application Development. Anthropic Engineering, 24 March 2026.

Anthropic. 2026b. Trustworthy Agents in Practice. Anthropic, 9 April 2026.

Cemri, M., Pan, M. Z., Yang, S., et al. 2025. Why Do Multi-Agent LLM Systems Fail? arXiv:2503.13657.

Chen, L., Zaharia, M., and Zou, J. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. arXiv:2305.05176.

Debenedetti, E., Zhang, J., Balunovic, M., et al. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. arXiv:2406.13352.

European Commission. 2025. General-Purpose AI Code of Practice. European AI Office, 10 July 2025.

European Commission. 2026a. AI Act: Regulatory Framework and Application Timeline. Shaping Europe's Digital Future, current to 23 July 2026.

European Commission. 2026b. Guidelines on Transparency Obligations under Article 50 of the AI Act. 20 July 2026.

European Commission. 2026c. Human-Centred Machine Learning: Lighter, Clearer, Safer - ACHILLES Project Fact Sheet. CORDIS, Grant Agreement 101189689.

European Union. 2024. Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence. Official Journal of the European Union.

Google. 2026. ADK Go 1.0 Arrives. Google Developers Blog, 31 March 2026.

Guo, T., Chen, X., Wang, Y., et al. 2024. Large Language Model Based Multi-Agents: A Survey of Progress and Challenges. arXiv preprint.

Huang, J., Chen, X., Mishra, S., et al. 2023. Large Language Models Cannot Self-Correct Reasoning Yet. arXiv:2310.01798.

Huang, Y., Wang, W., Bao, H., et al. 2026. MemoHarness: Agent Harnesses That Learn from Experience. arXiv:2607.14159.

Kapoor, S., Stroebl, B., Siegel, Z., Nadgir, N., and Narayanan, A. 2024. AI Agents That Matter. arXiv:2407.01502.

Kim, S., Moon, S., Tabrizi, R., et al. 2024. An LLM Compiler for Parallel Function Calling. Proceedings of ICML 2024.

Liu, N. F., Lin, K., Hewitt, J., et al. 2024. Lost in the Middle: How Language Models Use Long Contexts. Transactions of the Association for Computational Linguistics 12:157-173.

Linux Foundation. 2025. Agent2Agent Protocol Project Announcement and Governance. Linux Foundation, 23 June 2025.

Lodha, A., Varnosfaderani, M. P., Chakraborty, A., and Mithal, A. 2026. Less Context, Better Agents: Efficient Context Engineering for Long-Horizon Tool-Using LLM Agents. arXiv:2606.10209.

Microsoft. 2026. Microsoft Agent Framework Documentation. Microsoft Learn, current to July 2026.

Model Context Protocol. 2025. Specification 2025-06-18. MCP Project.

OpenAI. 2025a. New Tools for Building Agents. OpenAI, 11 March 2025.

OpenAI. 2025b. Introducing AgentKit. OpenAI, 6 October 2025.

OpenAI. 2026a. The Next Evolution of the Agents SDK. OpenAI, 15 April 2026.

OpenAI. 2026b. Speeding Up Agentic Workflows with WebSockets in the Responses API. OpenAI Engineering, 22 April 2026.

OpenTelemetry. 2026. Semantic Conventions for Generative AI Systems. OpenTelemetry Specification, current to July 2026.

Packer, C., Fang, V., Patil, S. G., Lin, K., Wooders, S., and Gonzalez, J. E. 2023. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.

Schick, T., Dwivedi-Yu, J., Dessi, R., et al. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. NeurIPS 2023.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. NeurIPS 2023.

Turpin, M., Michael, J., Perez, E., and Bowman, S. R. 2023. Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting. NeurIPS 2023.

Valmeekam, K., Olmo, A., Sreedharan, S., and Kambhampati, S. 2023. Large Language Models Still Can't Plan. arXiv:2206.10498, revised 2023.

Valmeekam, K., Marquez, M., Olmo, A., Sreedharan, S., and Kambhampati, S. 2024. On the Planning Abilities of Large Language Models: A Critical Investigation. NeurIPS 2024.

Wang, X., Ji, Z., Wang, Y., et al. 2025. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. Technical report and ICLR workshop paper.

Wei, H. 2026. Architectural Design Decisions in AI Agent Harnesses: An Empirical Study of Open-Source Systems. arXiv:2604.18071.

Wei, J., Wang, X., Schuurmans, D., et al. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. NeurIPS 2022.

Wu, X., Li, K., Zhao, Y., et al. 2025. ReSum: Unlocking Long-Horizon Search Intelligence via Context Summarization. arXiv:2509.13313.

Wu, Y., Zheng, Y., Xu, T., et al. 2026. ContextBudget: Budget-Aware Context Management for Long-Horizon Search Agents. arXiv:2604.01664.

Xie, T., Zhang, D., Chen, J., et al. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. NeurIPS 2024.

Xu, F., Hao, Q., Zong, Z., et al. 2023. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. arXiv:2305.18323.

Xu, H., Wang, S., Li, X., et al. 2024. TheAgentCompany: Benchmarking LLM Agents on Consequential Real-World Tasks. arXiv:2412.14161.

Yang, J., Jimenez, C. E., Wettig, A., et al. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. NeurIPS 2024.

Yao, S., Zhao, J., Yu, D., et al. 2023a. ReAct: Synergizing Reasoning and Acting in Language Models. ICLR 2023.

Yao, S., Yu, D., Zhao, J., et al. 2023b. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. NeurIPS 2023.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. 2024. Tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv:2406.12045.

Zhong, H., and Zhu, S. 2026. AI Harness Engineering: A Runtime Substrate for Foundation-Model Software Agents. arXiv:2605.13357.

Zhou, S., Xu, F. F., Zhu, H., et al. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. ICLR 2024.

Colophon

Agentic AI 2026: Essential Patterns and Beyond was prepared as a revised draft for scholarly, technical, and project review. The manuscript was authored by Sînică Alboaie and published by Axiologic Research as part of the research activity surrounding ACHILLES. Generative AI was used for literature discovery, source comparison, synthesis, critical restructuring, diagram ideation, and language formulation under the author's direction and responsibility.

The diagrams are original explanatory schematics created for this edition. AchillesAgentLib and MRP-VM are described as active research and engineering directions; the text distinguishes those proposals from published external evidence. Product names and project names remain the property of their respective owners. Regulatory discussion is informational and does not constitute legal advice.

REVISED DRAFT EDITION, VERSION 0.7 - JULY 2026