MRP-VM

Automating the Construction of Trustworthy Custom Agentic Harnesses

MRP-VM

Automating the Construction of Trustworthy Custom Agentic Harnesses

A Governed Architecture for Specialising Large Language Models on Repeatable Work

A conceptual book from the AssistOS research programme

Research conducted by Axiologic Research as part of the European research project Achilles.

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.

Project Note

This book presents MRP-VM as an architecture for automating the construction and controlled improvement of custom agentic harnesses. It is not a frozen description of one software release. Interfaces, file layouts, services, and internal mechanisms may change; the enduring subject is the set of responsibilities that turns powerful model behaviour into repeatable, testable, and governable work.

The central claim is deliberately pro-LLM. MRP-VM does not compete with large language models, coding agents, symbolic systems, or future foundation models. It gives them a disciplined operating environment. Better models can improve interpretation, planning, synthesis, and candidate authoring, while the harness preserves intended purpose, source authority, tool permissions, qualification, trace, version identity, and human control.

A custom agentic harness is the maintained place where an organisation consolidates how a repeatable class of work should be performed. The "how to do this" is expressed through versioned Agentic Knowledge Units, explicit plans, declared interpreters, qualification procedures, tests, settings, and governance rules. Operational experience can motivate candidate revisions, but active behaviour changes only after comparison, approval, publication, and the preservation of a rollback target.

Trustworthy AI is the primary evaluative perspective of this edition. Trustworthiness is treated as a socio-technical and lifecycle property rather than a flattering adjective for a model. The book asks whether a system is valid and reliable for its intended use, safe, secure and resilient, accountable and transparent, explainable and interpretable, privacy-enhancing, fair in the relevant context, and governed through evidence and responsible human roles (National Institute of Standards and Technology, 2023).

The manuscript draws on the current MRP-VM documentation, the MRP-VM position paper, the Meta-Rational Pragmatics article series, the AssistOS research pages, and relevant research on planning, agent orchestration, neuro-symbolic systems, explainability, risk management, and AI governance. The legal discussion is a July 2026 research synthesis rather than legal advice; organisations must verify the applicable law, guidance, standards, and sector rules for their intended use.

MRP-VM is positioned within the wider AssistOS effort, where agent architecture, specification-driven research and development, and trustworthy AI are mutually reinforcing. The institutional framing used throughout is: Research conducted by Axiologic Research as part of the European research project Achilles.

Contents

The entries below are linked to their destination headings.

Preface

Large language models have created an extraordinary engineering opportunity. They can interpret ambiguous requests, reorganise difficult material, propose plans, write and revise code, translate between representations, and communicate results in natural language. Their progress should not be treated as a threat to an architecture such as MRP-VM. It is the reason such an architecture becomes more valuable.

The difficulty is not that LLMs are too capable. The difficulty is that a successful model response does not by itself establish a repeatable organisational capability. The procedure may remain implicit, the context may vary silently, the model or provider may change, and the criteria that made the answer acceptable may never be recorded. The next request can therefore look similar while being handled under a materially different operational configuration.

Organisations need a way to retain the flexibility of general models while controlling what happens around them. They need to state which work is being attempted, which knowledge is admissible, which tools may be used, where human approval is required, what evidence qualifies an intermediate result, and how a change to the procedure is tested before it reaches another user. This is the practical domain of the custom agentic harness.

A harness is more than a prompt wrapper and more than a loop that repeatedly calls tools. It is a bounded, versioned environment in which models, deterministic software, retrieval, solvers, simulators, connectors, and human decisions are assigned explicit responsibilities. The harness makes the division of labour inspectable. It also makes the system's claim smaller and stronger: not "this model can do anything," but "this configuration is designed and evaluated for this class of work under these conditions."

MRP-VM provides a common substrate for building such harnesses. Its runtime keeps authority, state, budgets, effects, trace, and publication under stable control. Its Agentic Knowledge Units provide versioned homes for procedures, instructions, knowledge, tests, and optional interpreters. Its SOP representation exposes meaningful work and dependencies. Its qualification mechanisms determine what may propagate. Its candidate process lets people and coding agents propose improvements without silently rewriting the active system.

This arrangement creates a direct route from repeated work to reusable practice. When the same kind of problem returns, the harness need not improvise everything again. It can consolidate the organisation's "how to do this" in inspectable capabilities, refine

those capabilities when evidence justifies a change, and replace the underlying model without losing the purpose, tests, trace, or governance of the practice.

Trustworthy AI is therefore not presented here as a demand to weaken models or remove judgment. It is the discipline of using model judgment where it adds value while surrounding it with scope, evidence, qualified outputs, human responsibility, security boundaries, change control, and continuous evaluation. This is particularly important in regulated industries, where adoption depends not only on benchmark performance but on documentation, oversight, data governance, resilience, monitoring, incident response, and the ability to show what system was actually in use.

Introduction

The engineering opportunity

How can an organisation turn increasingly powerful LLMs into trustworthy capability for repeatable work without freezing their flexibility or surrendering control over the way they act?

This question is more productive than asking whether a general model or a specialised system will "win." General models provide broad semantic competence. Custom harnesses provide purpose, repeatability, authority, evidence, and institutional memory. The strongest systems will combine them. As models improve, the harness can become more capable; as the harness becomes more disciplined, better models can be adopted with less operational uncertainty.

MRP-VM brings the decisive control points into the architecture. A request executes under a pinned capability and settings generation. Selection chooses a small working set of relevant capabilities. Planning exposes meaningful responsibilities and dependencies. Each local step receives bounded context and authority. Outputs remain provisional until they are qualified. Effects, retries, revisions, and stopping conditions become traceable. Candidate changes are evaluated against a baseline before publication.

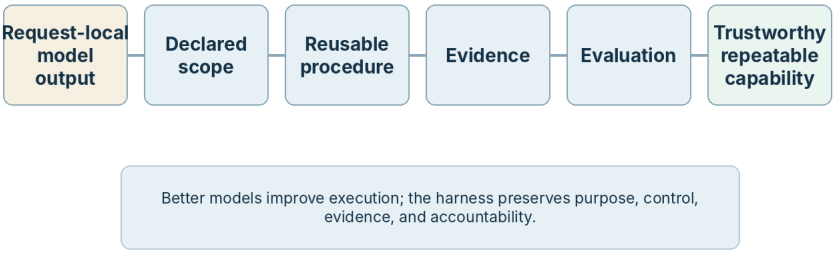


Figure 1. A request-local model output becomes a trustworthy repeatable capability only when declared scope, reusable procedure, evidence, and evaluation are made explicit.

Four concepts that define the system

A custom agentic harness is a versioned operational environment for a defined kind of work. It includes purpose, admissible inputs,

exclusions, capability catalog, knowledge sources, model and tool policy, output contracts, human gates, evaluation, retention, monitoring, and governance. Its boundary is part of its competence.

A capability is a reusable and testable claim about how one responsibility should be performed under specified conditions. It is not merely a prompt. It has an interface, resources, provenance, tests, failure modes, qualification expectations, and an owner. In MRP-VM, an Agentic Knowledge Unit gives that capability a portable, revisable address.

A runtime is the authority-bearing substrate that interprets plans, maintains state, schedules dependencies, invokes capabilities and tools, enforces limits, records effects, controls continuation, and governs publication. The runtime does not need to contain every domain method. It must make domain methods usable without allowing unbounded semantic output to acquire system authority.

Trustworthy AI is an evidence-backed claim about a complete socio-technical system in a particular context of use. It includes validity and reliability, safety, security and resilience, accountability and transparency, explainability and interpretability, privacy, fairness, and effective human responsibility. These properties can conflict and must be balanced under the intended purpose; no model score can substitute for that work (National Institute of Standards and Technology, 2023, 2024).

What MRP-VM is not

MRP-VM is not a foundation model and does not claim to outperform one at raw language capability. It is not a universal agent, a single prompt technique, or a demand that symbolic methods replace neural methods. It is not a compliance certificate. It is an architectural vocabulary and implementation direction for constructing systems in which strong models can be specialised, supervised, evaluated, replaced, and improved without losing the history of what changed.

The position paper is explicit that the contribution is orchestration and control rather than superiority over general LLMs or coding agents. Its strongest near-term domain is the bounded but non-trivial task class: work that repeats often enough to justify a maintained method, varies enough to need semantic judgment, and matters enough to require evidence, review, or controlled action (MRP-VM Project, 2026).

The book's central claim

The central claim is that MRP-VM can automate important parts of building and improving custom agentic harnesses while keeping activation governed. It can help select capabilities, formulate explicit plans, author candidate AKUs and tests, compare a candidate with a baseline, and publish coherent generations. Automation accelerates

construction; versioning, evaluation, authority, and rollback keep construction accountable.

This is where specialisation becomes a trustworthy-AI strategy. A specialised harness can define a narrower intended purpose, restrict sources and actions, route exact subproblems to exact tools, require stronger evidence for higher-risk claims, expose human approval points, and measure regressions on the work that actually matters. A better LLM can be introduced as an interpreter change and evaluated against the same obligations instead of silently redefining the product.

Part I explains why repeatable capability requires a custom harness and grounds the design in Meta-Rational Pragmatics. Part II describes the governed runtime and its control mechanisms. Part III shows how authorised evidence, candidate workspaces, evaluation, and versioning can automate harness construction. Part IV addresses applications, AssistOS and Achilles, trustworthy-AI requirements, regulation, and adoption.

Why Custom Agentic Harnesses

A conceptual division of the MRP-VM argument.

CHAPTER 1

From Model Outputs to Repeatable Capability

A strong answer is not yet a reliable system

Imagine a research team asking an AI system to compare two theories, locate the decisive empirical disagreements, and propose an experiment. The answer is excellent. It isolates the right variables, notices a hidden assumption, and suggests an affordable design. Everyone is impressed. A week later, a similar request receives a shallow comparison. No one can determine whether the first result depended on a better source set, a fortunate model sample, an unrecorded prompt variation, or a temporary tool response. The team possesses a good answer, but it has not acquired a dependable research capability.

This gap is easy to miss because answers and capabilities can have the same surface form. Both may appear as polished text. Yet they belong to different ontological and institutional categories. An answer is situated: one request, one context, one set of available materials, one moment. A capability is counterfactual and reusable: it claims that under stated conditions a system can perform a recognizable responsibility again. That claim must survive variation.

The current MRP-VM philosophy makes this distinction foundational. A raw response has no inherent owner, no stable scope, no reliable boundary between evidence and memory, and no explanation of why the next request should be treated in the same way. A capability, by contrast, needs an interface, resources, tests, provenance, observable failure modes, and a place in a larger composition (MRP-VM Documentation, 2026b). The difference is not bureaucratic. It is the difference between an event and a practice.

Table 1. A request-local model output and a repeatable capability are different kinds of achievement.

Dimension	Answer and capability
Temporal scope	**Answer.** Useful within one interaction. **Capability.** Intended for reuse across a declared class of cases.
Boundary	**Answer.** Often inferred from the immediate prompt. **Capability.** States inputs, exclusions, outputs, resources, and authority.

Dimension	Answer and capability
Procedure	**Answer.** May be improvised and transcript-bound. **Capability.** Versioned, inspectable, replaceable, and owned.
Evidence	**Answer.** Can remain implicit or ephemeral. **Capability.** Carries provenance and admissibility rules.
Evaluation	**Answer.** Judged mainly by the final response. **Capability.** Measured against tests, baselines, and acceptance criteria.
Failure	**Answer.** May simply look wrong after the fact. **Capability.** Has recognizable failure modes, escalation, and unresolved states.
Memory	**Answer.** Can be retained accidentally. **Capability.** Requires explicit retention and promotion.
Change	**Answer.** The next response may differ silently. **Capability.** Revision creates a new version with evaluation and rollback.

Capability as an operational commitment

The word "capability" often sounds purely technical, as though it named a function available in a software library. MRP-VM gives it a richer meaning. A capability is a social contract among those who rely on a harness, those who maintain it, those who provide knowledge, and those who authorise its effects. It states what the system is entitled to attempt, what evidence it may use, which outcomes count as acceptable, and how it should behave when the problem exceeds its scope.

This contract has an inward and an outward face. Inwardly, it disciplines implementation. The capability should have one coherent responsibility. Its instructions, rubrics, knowledge, code, and tests should be inspectable. Its resource needs and permissions should be declared. Its output should have a recognizable form. Outwardly, the contract makes the system legible to domain experts and users. A person should not need to reverse-engineer a prompt in order to understand what role the system claims to occupy.

Consider a contract-review harness. “Review contracts” is not a capability in the useful sense. It is a domain label. A coherent capability might be “identify clauses that change the governing-law assumption in the approved template family.” That capability can name its inputs, jurisdictional sources, exclusion conditions, output artefact, confidence language, escalation rule, and tests. Another capability may assess indemnity structure. A third may compose an executive summary. The harness can then make a bounded claim without pretending that one procedure covers the whole legal world.

Capability as contract also clarifies why negative cases matter. Testing only successful activation teaches us whether a procedure can work. Testing negative transfer teaches us whether the system knows when the procedure should *not* apply. In adaptive systems, the latter is often the more important discipline. A new capability that improves five target cases but activates across fifty unrelated requests has not become useful; it has become contagious.

From improvisation to maintained practice

A successful interaction can reveal several kinds of reusable insight. It may expose a missing knowledge source, a recurrent transformation, a better decomposition, a stronger validation question, a cheaper route to the same result, or a boundary that should have been explicit. The temptation is to capture the interaction itself: copy the response into memory, add the user’s phrase to a prompt, or create a special rule for the incident. This often produces the appearance of learning while narrowing the system around accidental details.

MRP-VM proposes a different conversion. The interaction becomes evidence for a *capability gap*. That gap is restated as a reusable responsibility. A candidate procedure is written or revised. It is tested on varied cases and on cases where it must not activate. Its relationship to existing capabilities is examined. It is evaluated as part of the whole harness. Only then may it enter an active generation.

This sequence is demanding because it refuses to identify repetition with generalisation. Repetition says, “we saw this before.” Generalization says, “we have isolated the invariant that makes these cases relevantly similar.” The first can be achieved by memory. The second requires conceptual work.

The distinction has a close analogue in science. A single observation can motivate a hypothesis, but it does not establish a law. An experimental anomaly can justify attention, but it does not determine the theory that should replace the current one. In software, a bug report can justify investigation, but a patch that hard-codes the exact input is not a robust fix. MRP-VM imports this methodological modesty into agent learning: experience proposes; evaluation disposes.

The capability test
A response becomes evidence for capability only when the system can state the reusable responsibility, the conditions under which it applies, the resources and authority it needs, the observable artefact it produces, the ways it can fail, and the evaluation by which a future revision will be compared.

Why this distinction changes system design

Once answers and capabilities are separated, several architectural consequences follow.

First, request output should not silently become persistent knowledge. Generated text may be useful for the present task while remaining unsuitable as a future source. It may summarise uncertain evidence, combine assumptions, or contain stylistic material that should not be retrieved as fact. Memory governance therefore requires explicit retention decisions and layered scopes.

Second, the procedure that generated an answer should have an address. If a team cannot identify which reusable unit selected the source, performed the transformation, or qualified the result, it cannot improve the relevant practice without risking unrelated behaviour.

Third, improvement must be versioned. A team should be able to compare the active capability with a candidate, identify the evidence for change, observe regressions, and restore the previous configuration. Otherwise adaptation is only drift with a positive story attached.

Fourth, the system must preserve failure as information. A rejected output, an unresolved validation, or a planner frontier can reveal where the current capability contract is incomplete. Forcing every request into a polished completion destroys precisely the evidence needed for responsible learning.

Finally, a harness should be judged by more than immediate fluency. Relevant measures include unseen task success, selection accuracy, use of exact tools, quality of qualification, preserved work after local failure, reviewer effort, latency, cost, negative transfer, and reversibility. These measures describe an institution of capability, not merely a model benchmark.

Bounded scope enables stronger guarantees

The answer-capability distinction may seem conservative because it slows the promotion of apparent success. In fact, it is what makes ambitious adaptation possible. A system can be allowed to explore, generate code, revise procedures, and propose new abstractions precisely because these activities occur in a candidate space where invention is not confused with authority.

The larger ambition is not to eliminate improvisation. Improvisation is one of the strengths of language models. The ambition is to create a disciplined path by which useful improvisation can become maintained practice—and by which failed improvisation can remain visible without poisoning future work. The capability gap is therefore not a complaint about generative AI. It is the opening through which generative intelligence can be connected to engineering responsibility.

The practical opportunity is to move repeatability out of individual prompting skill and into a maintained system. Once scope, procedure, evidence, qualification, and ownership become explicit, a stronger LLM improves execution without forcing the organisation to redefine what it trusts every time the model changes.

CHAPTER 2

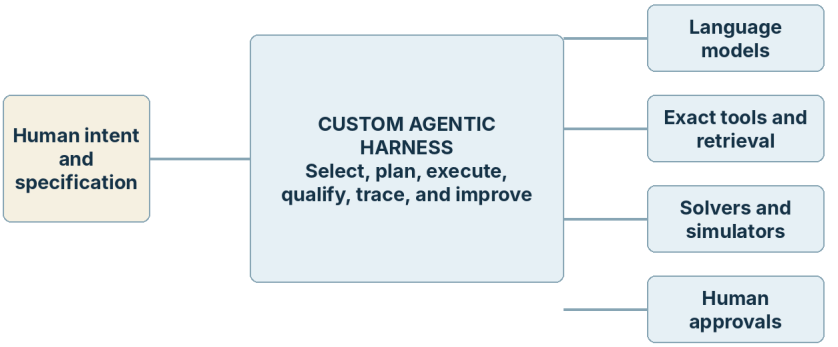
The Agentic Harness as an Engineering Layer

From language to consequence

A typical agent system has an appealing simplicity. Give a model a goal, a set of tools, and a loop. The model reasons, acts, observes, and continues until it declares completion. This pattern can be effective. It is also easy to build. Its weakness appears when the system's intermediate decisions become consequential.

Which part of the request was solved? Which source was considered authoritative? Why was one tool chosen rather than another? What assumptions were made before a solver was called? Which result was checked, and under what criterion? If a local premise changes, which downstream work is invalid? When may a generated statement enter memory? A transcript may contain clues, but it rarely provides a stable operational answer to these questions.

The agentic harness layer is the place where such decisions become first-class objects. It sits between open-ended human language and the heterogeneous mechanisms that produce effects. It does not replace either side. It gives them a governable interface.



The harness layer keeps model power while adding repeatability, bounded authority, and evidence.

Figure 2. A custom agentic harness connects human intent to LLMs, exact tools, solvers, simulators, and human approvals while keeping planning, authority, qualification, trace, and improvement explicit.

Why prompts alone are insufficient

A prompt can contain a procedure, but it is a poor long-term home for a practice when several conditions hold: the procedure has distinct responsibilities, different knowledge sources, alternative methods, independent failure modes, or a need for local revision. In a large prompt, these elements coexist as prose. Their boundaries are understood, if at all, by the model at runtime. A change to one instruction may alter the interpretation of another. Intermediate results may not be named. Validation may be performed by the same model under nearly identical context. The system can produce a coherent answer while leaving the operational structure implicit.

This does not mean prompts are unserious. Natural-language instructions are an important procedural medium. The problem is not their linguistic form. The problem is the absence of an execution model around them. Source code becomes software not because text ceases to be ambiguous in principle, but because languages, compilers, runtimes, tests, version control, and operating systems create a disciplined environment for interpretation and effect. MRP-VM asks what analogous environment is needed when part of the procedure remains natural-language and model-mediated.

The same criticism applies to transcripts. A reasoning-action history records a temporal sequence, but sequence alone does not expose the dependency structure of the work. A later step may depend on a particular extraction rather than on the entire conversation. If that extraction is challenged, replaying the whole transcript can reproduce irrelevant work and entangle the repair with previous phrasing. A graph of named responsibilities permits a more local question: what must be recomputed because this value changed?

The harness layer as operational semantics

In formal programming languages, operational semantics describe how expressions change machine state. Natural-language agent systems usually lack a comparable public account. Their practical semantics emerge from the interaction of prompt, retrieved context, model, tools, memory, and control loop. The meaning of "review this evidence," for example, is partly determined by which evidence was selected, which rubric was active, what tool was authorised, and what counted as completion.

Executable pragmatics begins from this runtime fact. Natural language does not need to become perfectly formal before it can participate in controlled execution. It needs an architecture that exposes the policies by which text becomes operational. The key questions are pragmatic: which reading is active, which context is admissible, what uncertainty can be tolerated, what effect is authorised, and when must a soft interpretation be escalated into a stricter regime (Alboaie, 2026h).

This harness layer therefore contains several kinds of object:

At the front of the harness layer is a task representation that preserves human intent without forcing the whole request to be repeated at every step. The representation becomes a visible plan whose nodes carry meaningful responsibilities and dependencies, while a capability catalog gives each reusable unit an explicit scope, resource envelope, test set, and version.

The same layer maintains runtime state that distinguishes provisional from committed artefacts, binds work to language models, code, retrieval systems, solvers, or human review, and applies qualification procedures before results may propagate. It also keeps a trace of selection, effects, revision, and stopping conditions, together with retention and publication controls that separate immediate output, evidence, and reusable knowledge.

None of these objects guarantees that the system is correct. Together they make error localizable and change reviewable.

An architecture of explicit transformations

The harness layer can also be understood as a sequence of translations.

The first translation is from **intention to work**. A request is normalized, bounded by policy and budget, and decomposed into responsibilities. The second is from **work to regime**. Each responsibility is related to an appropriate mode of interpretation or computation: semantic judgement, deterministic transform, retrieval, symbolic reasoning, simulation, or human decision. The third is from **output to artefact**. A provisional result is qualified before it becomes an accepted value. The fourth is from **experience to evidence**. A trace and authorised feedback are packaged without assuming that they already describe a reusable improvement. The fifth is from **evidence to capability**. Candidate work proposes a general procedure and evaluation determines whether it earns adoption.

These translations are where many agent failures hide. A model may answer the wrong interpretation of the request. A planner may omit a necessary qualification. A retriever may select topically similar but legally irrelevant material. A symbolic solver may prove a property of a mistranslated specification. A critic may validate rhetorical completeness instead of evidentiary adequacy. A memory system may retain a tentative claim as fact. When the translations are implicit, the final error appears mysterious. When they are represented, the system can ask where the meaning changed.

A stable kernel and evolving domain practice

A central design choice is to keep the authority-bearing runtime comparatively thin. The kernel should own the mechanics that must not be reinterpreted freely during active execution: parsing the intermediate programme, maintaining state, scheduling

dependencies, enforcing budgets and permissions, isolating untrusted code, recording trace events, pinning versions, and controlling publication. Domain procedures, knowledge, filtering, planning strategies, qualification rubrics, and composition guidance belong in versioned capability units.

The division is direct: the runtime carries stable authority and lifecycle rules, while the capability library carries versioned domain procedures (MRP-VM Documentation, 2026b). The runtime does not prescribe every professional action. It defines roles, limits, permitted effects, records, and legitimate transitions. Domain practice remains adaptable within those controls.

The separation also prevents a subtle circularity. If a system can rewrite the authority that decides whether its rewrite is valid, then evaluation loses a stable reference point. MRP-VM does not claim that the runtime can never change. It claims that runtime change is a different engineering act from ordinary capability learning and should pass through a separate process. Active requests require a finite bootstrap, even if future deployments revise that bootstrap.

> The harness layer in one sentence
> A custom agentic harness is the governable space in which human intent becomes an explicit programme of bounded responsibilities, routed to appropriate interpreters, qualified before consequence, and connected to versioned evaluation and improvement.

Why this is a philosophical problem too

The harness layer is not merely middleware. It encodes a view of meaning. It refuses two extremes. The first is semantic monism: the belief that every task should ultimately be expressed in one privileged representation or solved by one universal reasoning regime. The second is unstructured contextualism: the belief that because meaning depends on context, explicit structure is futile.

MRP-VM takes a middle position. Meaning is stabilized locally through a bounded practice. The practice can specify entities, sources, constraints, admissible moves, success conditions, and a mode of validation. This stabilization is real enough to support action and criticism, but local enough to remain revisable. The architecture does not solve the philosophy of meaning. It operationalizes a disciplined stance toward its incompleteness.

That stance will become clearer in Chapter 4. For now, the engineering consequence is enough: interpretation must be visible wherever it can change what the system is permitted to do.

The harness layer is where control becomes constructive. It does not suppress model interpretation; it turns interpretation into one visible operation among selection,

planning, exact computation, qualification, human review, and retention. This is the difference between using an LLM and engineering a trustworthy capability around it.

CHAPTER 3

Specialisation Complements General Models

The limits of a universal-assistant claim

The universal assistant is an attractive product image. One interface understands every domain, remembers every preference, connects to every service, and improves continuously. The image gains force from the generality of language models: the same model can discuss law, medicine, software, history, and poetry. It is tempting to infer that one harness should likewise absorb every practice.

Yet model breadth and harness legitimacy are different. A model may have broad latent competence while a deployed system still needs a precise account of purpose, authority, sources, risks, and evaluation. A universal harness tends to hide capability boundaries. It becomes difficult to determine whether an output came from retrieval, deterministic computation, a domain procedure, a model judgement, or an accidental interaction among them. Improving one area may place unrelated areas at risk. Memory that helps one practice may violate another. A tool permission appropriate for engineering may be unacceptable in research or finance.

Specialisation offers a smaller and stronger claim: this harness has this purpose, these sources, these permissions, these tests, these limits, and these routes of explanation (MRP-VM Documentation, 2026b). The smaller claim is not less intelligent. It is more accountable.

Table 2. General-model and custom-harness postures make different kinds of operational claims.

Dimension	Universal-assistant and specialised-harness postures
Claim	**Universal-assistant posture.** Broad competence across open-ended tasks. **Specialised-harness posture.** Bounded competence for a defined purpose.
Knowledge	**Universal-assistant posture.** Large, fluid, and often weakly governed context. **Specialised-harness posture.** Named sources with scope and provenance.

Dimension	Universal-assistant and specialised-harness postures
Authority	**Universal-assistant posture.** Permissions may be inferred during interaction. **Specialised-harness posture.** Actions and human gates are declared in advance.
Evaluation	**Universal-assistant posture.** Generic benchmarks or anecdotal success. **Specialised-harness posture.** Task-class tests and operational acceptance criteria.
Failure	**Universal-assistant posture.** Pressure toward plausible completion. **Specialised-harness posture.** Explicit refusal, escalation, or bounded non-resolution.
Improvement	**Universal-assistant posture.** Prompt and memory may drift together. **Specialised-harness posture.** Candidate changes are isolated, compared, published, and reversible.
Collaboration	**Universal-assistant posture.** One agent attempts to absorb every role. **Specialised-harness posture.** Capabilities federate through explicit compatibility and evidence.

A harness is a bounded operational environment

Specialisation is not defined only by topic. A "scientific assistant" is still too broad. A harness is defined by a bounded operational environment that includes at least seven dimensions.

Purpose. What human practice is the system intended to support? Literature screening, hypothesis mapping, protocol drafting, data-quality review, and statistical interpretation are different responsibilities even when they belong to one research programme.

Authority. What may the system observe, recommend, modify, or execute? Reading a repository is different from writing a patch; writing a patch is different from merging it; recommending a treatment is different from authorising one.

Knowledge. Which sources are legitimate, current, and scoped for the task? A general web result, an approved internal policy, a peer-reviewed database, and a user-provided document carry different epistemic status.

Tools. Which external systems may be accessed, with what credentials, data boundaries, and side effects? Connectors should be named capabilities, not ambient network access.

Output contract. What artefact should the harness produce, and what statuses must it preserve? A ranked list, a proof object, a patch candidate, an evidence table, a refusal, and an unresolved report are not interchangeable.

Evaluation. What would count as improvement or unacceptable regression? The criteria may include accuracy, source coverage, plan validity, cost, latency, negative transfer, safety, and reviewer effort.

Learning policy. What feedback may be retained, who may authorise its use, and how may a candidate change enter production?

Together these dimensions define the operational ethics of the harness. Ethics here is not an abstract appendix. It appears in concrete design choices: whether citation is mandatory, whether exact verification is preferred, whether side effects require approval, whether certain conversations may be retained, whether an unresolved outcome is acceptable, and who may inspect cross-session evidence.

Boundaries strengthen competence

Boundaries are often described negatively, as restrictions imposed on an otherwise capable system. In practice, they can produce competence. A bounded source set reduces irrelevant retrieval. A narrow output contract clarifies qualification. A limited tool surface makes authorisation intelligible. A defined task family permits meaningful tests. A known escalation path prevents the system from improvising beyond its expertise.

Herbert Simon's work on bounded rationality challenged the idea that intelligent action requires global optimisation under complete information. Agents satisfice within constraints, using representations and procedures adapted to their environment (Simon, 1957). MRP-VM adds an institutional dimension: the bounds should not merely constrain the system; they should be inspectable, versioned, and related to a capability claim.

This is also why specialisation should not be confused with rigid workflow. A specialised harness can remain flexible in decomposition, context selection, model routing, qualification, and local repair. Its boundary defines the space of legitimate adaptation. It says, in effect, "we may explore within this practice, under these authorities, toward these acceptance conditions." The result is constrained freedom rather than scripted inflexibility.

Refusal, escalation, and federation

A good boundary makes two positive behaviours possible: precise refusal and explicit federation.

Precise refusal is more informative than a generic "I cannot help." The harness can identify the blocked region, the missing authority or evidence, the attempted recovery, and the capability that would be required. A contract-review system may say that a clause falls outside the approved template family. A research harness may report that the decisive source is unavailable. An engineering harness may produce a patch candidate but stop before deployment. These are not failures of intelligence. They are demonstrations that the system understands its role.

Federation allows one specialised harness to ask another for a bounded capability without dissolving their identities. The first harness may delegate a formal check to a solver-oriented harness, a source review to a literature harness, or a deployment step to an operational harness. The transaction should preserve scope, provenance, authority, and result status. Reuse is explicit rather than contagious.

This principle matters for capability sharing. A procedure developed in one harness should not become active everywhere merely because all harnesses share a database. It may travel when its contract and evidence justify broader use. Cross-harness adoption is therefore analogous to technology transfer: a capability must be re-evaluated in the receiving operational context.

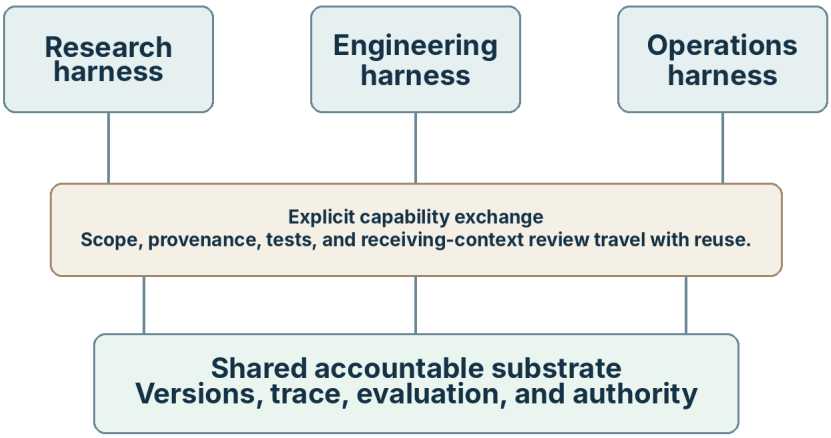


Figure 3. Specialised harnesses can share tested capabilities through explicit federation while retaining distinct purposes, authorities, sources, and evaluation regimes.

Specialisation and human responsibility

Specialised harnesses make responsibility easier to allocate. A domain owner can define purpose and source authority. A capability author can maintain a bounded procedure. An evaluator can design cases that reflect the practice. An operator can manage versions and resources. A governance reviewer can inspect retention and promotion. A user can understand what kind of help is being offered.

No participant must be omniscient. A universal-agent design often transfers unrealistic expectations to users and operators: they must know every hidden capability, anticipate every tool interaction, and audit an undifferentiated system. Specialisation replaces that burden with a visible division of labour.

The same is true for coding agents. A coding agent can be highly productive when it receives a bounded capability gap, relevant evidence, a candidate workspace, and an evaluation contract. It becomes dangerous when asked to "improve the system" without a defined scope or when granted authority to activate its own proposal. Specialisation turns machine creativity into a contribution to a practice rather than an unbounded claim to agency.

A platform for many custom harnesses

MRP-VM is not specialised to one domain. The runtime generalises accountable execution—plans, state, capabilities, trace, versions, and evaluation—while each harness remains specific in purpose, evidence, authority, and practice. Tested capabilities can be shared without erasing those boundaries.

General models contribute breadth and adaptability; custom harnesses add controlled authority, domain evidence, repeatability, and evaluation. New model generations can enter as versioned interpreter improvements rather than undefined system changes.

CHAPTER 4

Meta-Rational Pragmatics for Controlled Specialisation

Reasoning within a frame and evaluating the frame

Every act of reasoning presupposes a frame. The frame determines what counts as an object, which distinctions matter, what evidence is admissible, which operations are legitimate, and what would count as completion. In ordinary work the frame remains mostly tacit. A clinician reads a result under a diagnostic practice; an engineer reads a log under a failure model; a scientist reads a plot under a measurement theory; a lawyer reads a clause under a jurisdiction and doctrine.

Rationality operates within such structures. Meta-rationality begins when the suitability of the structure itself becomes part of the problem. It asks whether the current categories are adequate, whether the evidence regime fits the claim, whether the representation hides a decisive relation, whether the method is being applied beyond its boundary, or whether several partially adequate frames must be preserved.

This does not make object-level reasoning unnecessary. It governs when object-level reasoning should continue, be revised, be supplemented, or stop. The psychological relatives are metacognition, self-monitoring, and reflective regulation: planning, checking, evaluating confidence, and revising strategy (Flavell, 1979; Fleming, 2021). Dialectical and developmental traditions add the capacity to tolerate contradiction and perspective change without collapsing into incoherence (Basseches, 1984; Kegan, 1994). MRP translates these intuitions into computational objects and transitions rather than claiming to reproduce a human psychology (Alboaie, 2026d).

Pragmatics before unjustified closure

The term **pragmatics** emphasizes that meaning is shaped by use, situation, purpose, and the consequences of interpretation. The later Wittgenstein's language-games and forms of life are relevant because they resist the idea that meaning is exhausted by an abstract correspondence detached from practice (Wittgenstein, 1953). Pragmatist traditions similarly treat inquiry as an activity in which concepts are tested through their role in experience and action (Peirce, 1877; Dewey, 1938).

MRP does not infer that meaning is arbitrary. Material reality, evidence, institutional rules, formal constraints, and practical consequences limit admissible interpretation. A

hammer can be described as a physical object, a tool, a design artefact, a workflow component, or an improvised instrument. These descriptions are not equally useful in every task, and the material object constrains all of them. Yet no single description exhausts the hammer's significance. Meaning appears as a constrained space of admissible moves within a bounded practice (Alboaie, 2026g).

This yields a useful formula: **pluralism without relativism**. Several frames may be partially adequate, but they can still be compared. They differ in explanatory power, evidentiary support, computational tractability, risk, scope, and fit to purpose. MRP rejects premature closure, not evaluation.

Meta-rationality
Meta-rationality is the disciplined capacity to examine and revise the frame, method, representation, evidence regime, or closure condition under which local reasoning is being conducted. It is not a license to evade commitment; it is a way to make commitment corrigible.

Commitment under revisable assumptions

Meta-rationality occupies an uncomfortable position between two easier attitudes. One is rigid certainty: the current framework is treated as the final structure of the problem. The other is dissolving scepticism: because every framework is partial, none deserves strong commitment. MRP rejects both. It asks for action under acknowledged assumptions and revision when reality, evidence, or failure justifies it.

The position is vulnerable to several misunderstandings. It can be reduced to relativism, indecision, or mere nuance. It can become a style of sounding complex without adding constraint. It can be appropriated by actors who use criticism of established frameworks to avoid ordinary standards of evidence. It can remain "too meta" for too long, spending more effort examining assumptions than the scale of the problem warrants (Alboaie, 2026e).

These risks are not external objections; they belong inside the framework. Meta-rationality must itself be treated as fallible. Not every task requires reframing. Many problems are well served by stable semantics and deterministic procedures. A mature system should invoke meta-level reconsideration proportionally, usually when a concrete signal justifies it: failed validation, contradiction, missing context, repeated state, unsuitable method, or unresolved ambiguity. The normal path should remain efficient.

This proportionality is essential to MRP-VM. Plurality is available where doubt arises, but the runtime need not execute every alternative. Priority defines a common path. Reconsideration opens the nearest relevant choice point. A different context filter changes the knowledge regime; a different capability changes the method; a different

interpreter changes the computational realization; a different validator changes the standard of demonstration; replanning changes the local representation of the problem. Meta-rationality is therefore distributed across mediation points rather than embodied in one mystical "reasoning module" (MRP-VM Project, 2026).

Active theory frames

A useful conceptual object is the **active theory frame**: a bounded, goal-conditioned assembly of entities, assumptions, constraints, hypotheses, sources, tools, inference policies, and closure conditions. It is "theory" in a practical sense, not necessarily a grand scientific theory. The frame says what the local problem has become for the purposes of action.

For example, an incident-triage frame may include a service boundary, a time window, a set of logs, recent deployments, an approved runbook, candidate failure modes, and a criterion for escalation. A policy-review frame may include a jurisdiction, policy version, request interpretation, formal constraints, and a required proof or counterexample. The frame reduces an open field into a tractable object without pretending that the reduction is final.

This concept is closely related to bounded rationality. Computation becomes tractable when the system isolates the relevant entities, dependencies, invariants, and success criteria, then routes the subproblem to an interpreter appropriate to that structure. Some subproblems are constraint-dominated and suit symbolic verification. Others are similarity-driven and suit retrieval or embedding methods. Others involve synthesis, probabilistic uncertainty, graph structure, or human judgement. Regime selection is therefore more fundamental than simple tool choice: the system must first form a hypothesis about the computational shape of the problem (Alboaie, 2026i).

No Free Lunch results provide a formal reminder that no optimisation procedure is best across all possible problems without assumptions about the problem distribution (Wolpert and Macready, 1997). In practical terms, intelligence requires bias. MRP's contribution is to make the choice and revision of local biases more explicit.

Grounding through evidence and effects

Meta-rational pragmatics also offers a middle position on grounding. It does not promise a final, once-and-for-all foundation that eliminates interpretation. Nor does it accept free-floating language detached from reality. Grounding is treated as disciplined and revisable contact: source provenance, measurement, tool effects, formal constraints, human observation, and practical consequences constrain the active frame.

This view is compatible with scientific realism while remaining fallibilist. Scientific models can improve contact with reality without becoming identical to reality. Formal systems can produce exact conclusions within a representation while leaving the adequacy of the representation open to criticism. A symbolic solver may be impeccable and still solve the wrong formalisation. A language model may identify the correct practical distinction and still lack the evidence needed to justify it. Grounding is therefore distributed across translation boundaries and qualification practices.

The philosophy of science has long distinguished healthy corrigibility from dogmatic closure. Popper emphasized exposure to refutation; Kuhn showed that scientific development can transform conceptual frameworks; Lakatos described research programmes whose hard cores and auxiliary hypotheses evolve under empirical pressure (Popper, 1959; Kuhn, 1962; Lakatos, 1978). MRP does not reproduce any one of these theories. It borrows their shared warning: successful methods become dangerous when their local achievements are reified into final metaphysics.

From philosophical principle to operational control

Philosophical subtlety is not enough. A system that praises plurality while leaving interpretation unrecorded has not become meta-rational; it has become vague. The executable turn requires concrete objects: frames, plans, capability units, interpreter choices, qualification regimes, traces, budgets, and stopping rules. It requires the ability to preserve alternatives when warranted and to terminate unresolved when no available procedure justifies a winner.

MRP is executable pragmatics because it governs how a fragment is read, formalised, checked, deferred, revised, or kept open. MRP-VM operationalises this through five architectural translations: situated meaning becomes bounded context selection; fallibilism becomes provisional values and qualification; pluralism becomes alternative capabilities, interpreters, and validators; bounded rationality becomes local decomposition and tractable regime selection; and accountability becomes versioned trace, evaluation, publication, and rollback (Alboaie, 2026d).

PART II

Architecture for Trustworthy LLM Work

A conceptual division of the MRP-VM argument.

CHAPTER 5

MRP-VM as a Governed Runtime

What the virtual-machine abstraction means

The phrase “virtual machine” can sound more technical than the core idea requires. In ordinary computing, a virtual machine defines an execution environment. It provides a programme representation, state, operations, scheduling rules, interfaces to external resources, and conditions under which execution continues or stops. The VM is not the application. It is the stable machinery that allows many applications to run under a common discipline.

MRP-VM adopts this image for natural-language work. A task is compiled or revised into an intermediate programme. The programme names meaningful values and dependencies. Commands are realized by reusable capability units and heterogeneous interpreters. The runtime selects context, enforces local contracts, records effects, qualifies provisional results, and decides whether to commit, reconsider, replan, or stop. The machine does not contain all intelligence. It organises where intelligence is allowed to operate and how its results enter shared state.

The abstraction has limits. Natural-language procedures do not have complete formal semantics. A language model is not a deterministic instruction decoder. A domain judgement may not have one exact truth condition. MRP-VM is therefore a pragmatic virtual machine rather than a conventional bytecode VM: its runtime constrains interpretation without pretending to eliminate it.

Stable authority, evolving capability

The architectural center is a separation between two kinds of change.

The first concerns **authority-bearing mechanics**: how requests are identified, how plans become graphs, how dependencies are scheduled, how state changes, how budgets are enforced, how untrusted code is isolated, how effects are authorised, how traces are recorded, how versions are pinned, and how a new catalog becomes active. These mechanics should be stable during a request and difficult to alter casually.

The second concerns **semantic practice**: domain procedures, instructions, rubrics, source-governed knowledge, selection policy, planning strategy, qualification criteria, composition guidance, and the choice of interpreter. These elements must be able to evolve because they encode what the harness is learning about its work.

The runtime-and-capability division assigns two different responsibilities. The runtime determines who or what may act, under which procedure, with which limits, with which record, and through which change process. The capability library contains the versioned knowledge and procedures that answer domain questions within those controls.

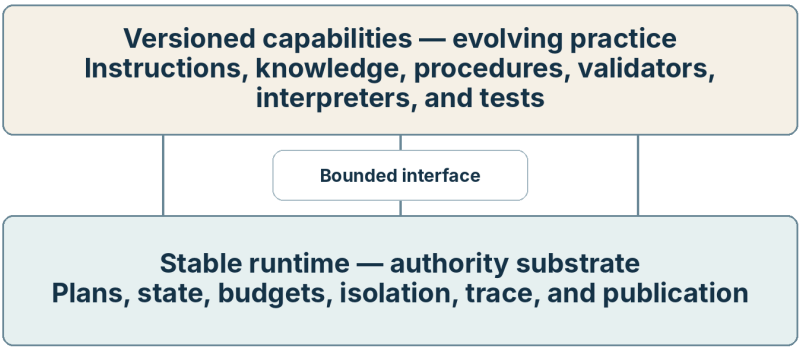


Figure 4. The stable runtime holds authority and lifecycle rules; versioned capabilities carry the semantic practices that may evolve.

This separation makes accountable improvement possible. If a coding agent can modify the publication mechanism while proposing a capability change, it can alter its own acceptance process. If a running request can silently adopt a newly generated procedure, the trace no longer refers to one coherent configuration. If every domain rule is embedded in the runtime, domain improvement becomes risky infrastructure change.

The separation is not absolute. Runtimes do evolve, and some semantic decisions will inevitably become native when their meaning is stable enough. The point is procedural: changes to the authority substrate belong to a different release and review path than candidate changes to ordinary capabilities.

The runtime kernel

A thin kernel performs only the mechanics that need uniform authority. The position paper describes a core that parses the intermediate language, maintains the graph and variable store, schedules ready nodes, binds commands to applicable capabilities, invokes configured filtering and validation procedures, records trace events, and applies explicit promotion and override rules (MRP-VM Project, 2026).

“Thin” does not mean small in lines of code. It means narrow in semantic ambition. The kernel should not contain hidden domain rules or hard-coded interpretations that properly belong to a versioned capability. Its job is to validate structure and enforce boundaries, not to become an invisible second intelligence layer.

This principle can be tested with a simple question: *Could a domain expert or capability author reasonably want to revise this behaviour without changing the authority model?* If yes, the behaviour probably belongs in a capability or declared resource. If the behaviour concerns integrity—graph consistency, permission enforcement, generation pinning, isolation, or publication atomicity—it probably belongs in the runtime.

The thin kernel also prevents infinite regress. A system cannot ask an unrestricted selector to choose the selector that will choose the selector forever. Active execution needs configured defaults, metadata-compatible candidates, terminal checks, reconsideration limits, and designated acceptance procedures. These policies may be revisited between deployments, but each run begins from a finite bootstrap.

Explicit state

Many agent systems carry state in a transcript or hidden framework object. MRP-VM treats state as part of the execution model. Conceptually, a running session includes the original task, the active capability and knowledge view, the current graph, a store of named values, budgets and policy, and an append-only trace.

This state distinguishes several statuses that ordinary conversation often blurs:

Before work is accepted, the public state distinguishes what has been requested but not yet planned, what has been selected as relevant but not yet used, and what has been planned but cannot run because dependencies are missing. Once execution begins, it distinguishes work that has run but remains provisional from work that has been qualified and committed.

The state also records rejected artefacts, artefacts invalidated by an upstream change, and questions that remain unresolved. Beyond the immediate run, it distinguishes material retained for the session, material proposed as learning evidence, and capability versions that have actually been published for reuse.

The distinctions are operational. A provisional value should not release dependent work. A rejected candidate can remain in the trace without becoming a shared state. A committed value can later be invalidated if a strong dependency changes. A useful answer can remain request-local without entering the knowledge base. A feedback item can enter an evidence package without becoming active instruction.

The state therefore implements memory governance. The machine does not ask only "what text do we have?" It asks "what status does this artefact have, who authorised it, and what may it influence?"

Controlled effects

A virtual machine becomes especially important when tasks have side effects. Reading a document, querying a database, executing code, writing a file, sending a message, or updating an external record are not equivalent operations. Each should be represented as a declared capability or connector with explicit authority, resource limits, and trace events.

MRP-VM's design direction is to prevent ambient authority. A capability should not inherit unrestricted network access or credentials merely because the host process has them. It should request named resources through a narrow interface. The runtime can then approve, deny, simulate, substitute, or record the effect. Tests can replace real services with deterministic stand-ins. Evaluation can distinguish a correct recommendation from an unauthorised action.

This is where the virtual-machine abstraction becomes an execution boundary rather than an organisational convention. Flexible semantic interpretation can propose an effect, but the runtime decides whether the effect is admissible. The same principle applies to generated code: a coding or synthesis capability may create a task-local programme, yet execution occurs in an isolated environment with bounded resources and no implicit path to publication.

Builder and user surfaces

A sophisticated runtime does not require a complicated user experience. The current MRP-VM philosophy explicitly protects a two-surface design. The final user may interact through chat, an API, a workflow button, or another application. The developer and operator surface exposes capability authoring, plans, trace, evaluation, settings, and publication. A thin compatibility interface can preserve the governed lifecycle without forcing users to learn the internal representation (MRP-VM Documentation, 2026b).

This separation is important because simplicity at the edge can conceal either discipline or disorder. A minimal interface is trustworthy only when it sits on top of a rigorous execution surface. Conversely, internal richness is useful only when it can be projected into explanations appropriate to different readers rather than imposed on everyone.

Current implementation snapshot — July 2026

The current documentation describes a Node.js runtime with SOP parsing and graph execution, request and session state, budgets, trace and structural replay, generation pinning, semantic model routing, isolated code execution, candidate workspaces, evaluation, publication, rollback, SDK and HTTP interfaces, and an OpenAI-compatible adapter. These mechanisms demonstrate one implementation of the conceptual contract. Future releases may change their exact form without changing the book's argument.

Why the runtime matters for trustworthiness

The virtual machine makes a philosophical commitment concrete: interpretation is never free-floating. It occurs inside a bounded operational configuration. A result has meaning not only because of its text, but because of the task, sources, procedure, dependencies, interpreter, qualification, and authority under which it was produced.

At the same time, the VM resists the opposite temptation to formalise away all ambiguity. It admits that some commands require semantic judgement. Its discipline lies in localising that judgement, surrounding it with explicit context and effects, and making reconsideration possible. The VM is therefore a machine for disciplined incompleteness: enough structure to govern action, enough openness to handle language, and enough history to revise the structure when it fails.

A governed runtime provides the stable control plane that LLM systems usually lack. It keeps state, authority, effects, budgets, version identity, and publication outside the discretion of one model call, while still allowing semantic capabilities to evolve rapidly inside explicit boundaries.

CHAPTER 6

Executable Natural Language and Explicit Dependencies

From specification to controlled execution

Natural language has always been used to specify work. Laws, contracts, operating procedures, research protocols, recipes, design briefs, and user stories all guide action without possessing the determinism of machine code. The arrival of large language models changes the execution model. Text can now be interpreted dynamically by systems that select context, invoke tools, transform representations, and generate local procedures.

This creates a new possibility and a new risk. The possibility is that natural language can serve as a high-level programming medium for complex knowledge work. The risk is that operational meaning becomes distributed across the text, the model, the retrieved context, the memory, the tools, and the control loop. A sentence can be executable while its execution policy remains opaque.

The Meta-Rational Pragmatics programme distinguishes formal-semantics-first and runtime-pragmatics-first approaches. Controlled languages such as Attempto Controlled English obtain executability by restricting language until it maps reliably into formal structures. Systems such as Gherkin bind readable steps to external code. Model-centred runtimes leave more underdetermined until execution, gaining flexibility at the cost of weaker guarantees. The likely future is hybrid: readable specification, explicit intermediate representation, bounded model interpretation, and hardening through schemas, code, tests, and formal tools (Alboaie, 2026h).

SOP as an intermediate programme

MRP-VM uses the idea of a Standard Operating Procedure language—SOP Lang—as a lightweight intermediate representation. The enduring point is not the exact syntax. It is that meaningful responsibilities and dependencies become visible before or during execution.

A conceptual example might read:

```
@evidence collectEvidence
Assemble the admissible observations and sources for the request.

@interpretation interpretCase
Develop the leading interpretation from $evidence.

@qualification qualifyInterpretation
Test $interpretation against $evidence and the applicable criteria.

@response composeResponse
Produce the user-facing result from $interpretation and $qualification.
```

Each declaration names a value and a command. The body gives the local frame. The programme does not attempt to expose every internal token or micro-inference. It exposes units of work that matter because they may need independent inspection, reuse, validation, comparison, routing, recomputation, or retention.

This granularity is critical. A plan with one node—"solve the request"—has not made work visible. A plan that transcribes every linguistic operation becomes unreadable and burdens the runtime with artificial detail. The right unit is a coherent responsibility whose output has operational significance.

Dependencies make assumptions visible

Dependencies are not only scheduling hints. They are claims about what a result means and what would invalidate it.

A **strong dependency** says that the downstream value materially depends on an upstream value. The node should not execute until the dependency is resolved; if the upstream value changes, the downstream result becomes suspect and should be recomputed. In current SOP Lang, a strong reference is written with a dollar sign.

A **weak dependency** says that an upstream value may guide context selection, interpretation, or qualification without establishing the same automatic recomputation obligation. It can provide orientation without being injected in full. In current syntax, a weak reference uses a tilde. Any material use should still appear in the trace.

The distinction prevents two common failures. Without strong dependencies, a system cannot localise invalidation. Without weak dependencies, every potentially relevant piece of context becomes a hard edge, producing sprawling graphs and unnecessary recomputation. The architecture acknowledges that influence has degrees and different operational consequences.

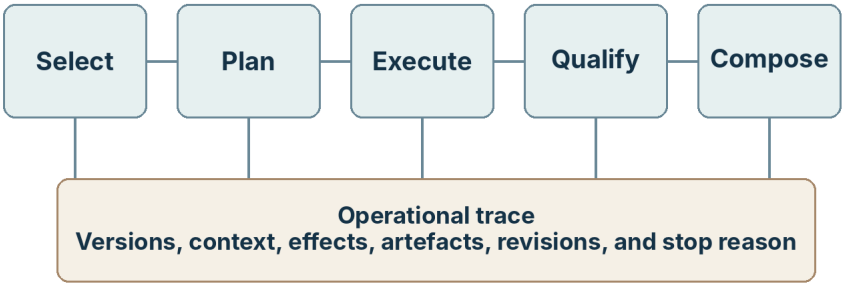


Figure 5. Visible work moves from bounded selection to an explicit plan, local execution, qualification, composition, and a traceable terminal state.

Local execution contracts

When a node becomes ready, it should receive a local contract rather than the entire compound request. The contract includes the node's responsibility, direct resolved dependencies, selected knowledge, bounded state, authorised capabilities, and applicable budgets. This prevents an upstream model call from performing work that belongs to later nodes and makes local output easier to qualify.

Locality changes the economics of reasoning. Smaller contexts can be cheaper and less distracting. Exact tools can receive structured inputs instead of conversational history. Different nodes can use different models or no model. Independent work can proceed in parallel. A reviewer can inspect the evidence and instruction relevant to one artefact rather than reconstruct the whole session.

Locality also improves contestability. If a diagnosis is challenged, the system can ask whether the evidence collection, interpretation method, or qualification regime was at fault. The objection becomes a proposal about a bounded part of the programme rather than a demand to regenerate everything.

Alternatives and plural execution

Some local problems admit more than one plausible method. Current SOP Lang makes two forms of plurality explicit. An ordered alternative means "try the preferred applicable route, and open another only if needed." Plural execution means "retain more than one result for comparison or composition." Similar plurality can exist implicitly through capability variants and interpreter choices.

The distinction reflects the meta-rational principle of proportionality. The runtime does not explore every combination of methods. It follows a normal route until the programme or the evidence provides a reason to branch. A high-risk task may request plural demonstration from the start. A routine task may escalate only after negative or inconclusive qualification.

Plurality is valuable when methods encode different assumptions. A symbolic checker and a language-model reviewer may not be redundant. One tests formal consistency inside a representation; the other may detect that the representation missed the user's intention. A source-constrained retriever and a broad semantic retriever may reveal different evidence boundaries. The goal is not a majority vote. It is a comparison under known regimes.

Ambiguity becomes a runtime responsibility

Executable natural language cannot promise to remove ambiguity. It can decide how ambiguity is handled.

Some ambiguity is harmless because several readings lead to the same action. Some can be constrained by schema, terminology, or examples. Some require clarification from the user. Some should trigger parallel interpretations. Some should block consequential action. The runtime and capability contract should make these choices explicit enough to evaluate.

This is the core of executable pragmatics. The text does not carry its entire operational meaning by itself. The runtime determines what reading is active, what evidence is admissible, what uncertainty is acceptable, and when to escalate to a stricter regime. The architecture treats this determination as part of the work rather than an invisible precondition.

Current implementation snapshot — SOP Lang

The current position paper describes declarations with named variables and command labels; strong references that create data dependencies; weak references that guide interpretation; ordered alternatives; plural execution; and planner-driven graph replacement. These constructs should be read as one compact implementation of enduring needs: named state, dependency, locality, plurality, and revisable decomposition.

Executable language as institutional procedure

There is a deeper reason natural language remains important. Professional practices are already expressed in language. Domain experts know how to explain what counts as relevant evidence, a good review, an unacceptable omission, or a proper escalation.

Forcing every such distinction into low-level code can hide the semantic policy from the people responsible for it.

MRP-VM therefore treats code and natural-language instruction as complementary. Code is appropriate where semantics are complete, stable, and testable. Model-mediated instruction is appropriate where interpretation, transformation, synthesis, or judgement remains necessary. The challenge is not to choose one medium universally. It is to give each procedure a visible place, version, test path, and boundary.

Executable natural language becomes institutionally meaningful when a reviewer can ask, “what instruction changed?”, “which sources were visible?”, “what exact tool executed?”, “what result was provisional?”, and “what rule allowed it to propagate?” The programme is then not merely text that a model happens to follow. It is a public artefact of work.

Executable natural language creates a useful middle ground between opaque prompting and fully formal software. It keeps the flexibility needed for interpretation while exposing the responsibilities and dependencies that reviewers, evaluators, and regulated organisations need to inspect.

CHAPTER 7

Agentic Knowledge Units: Consolidating How Work Is Done

Where reusable behaviour should live

Long-lived AI systems need a place for reusable, evolving behaviour. Several tempting locations are inadequate. A server route can hide domain policy inside infrastructure. A shared prompt can mix unrelated responsibilities. A model-provider configuration can change semantics without a meaningful diff. A retrieval corpus can contain knowledge without stating how it should be used. A tool plugin can expose an action without defining selection, qualification, tests, or lifecycle.

The Agentic Knowledge Unit—or AKU—is MRP-VM's answer. An AKU is a portable, versioned home for a coherent capability. It can contain natural-language instructions, source-governed knowledge, executable entry points, helper code, configuration, metadata, tests, and an optional interpreter binding. It declares how it matches a local responsibility, what resources it needs, what artefact it produces, and how its behaviour can be evaluated.

The position paper offers a compact abstraction: an AKU consists of a body, execution metadata, and an optional interpreter. The body may carry knowledge or procedure. Metadata makes the unit addressable by command, scope, status, priority, source, version, and override relationships. The interpreter may be a language model, deterministic programme, parser, retrieval procedure, symbolic reasoner, simulator, coding agent, or composite service (MRP-VM Project, 2026).

The exact folder structure is implementation-specific. The durable idea is that a practice has an address.

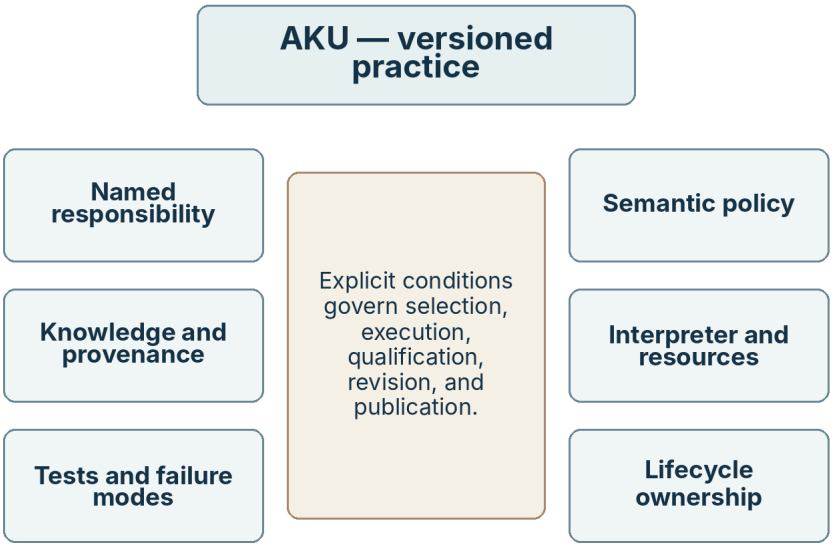


Figure 6. An AKU packages a named responsibility, semantic policy, knowledge, declared resources, an optional interpreter, tests, provenance, and lifecycle ownership.

More than a plugin

A plugin convention usually answers how code is loaded and called. An AKU must answer a broader set of questions:

Responsibility and selection. An AKU must state the responsibility it owns and the evidence under which it should be selected.

Policy, authority, and interfaces. It must identify the knowledge and instructions that define its semantic policy; the models, tools, connectors, time, memory, and side effects it may request; and the input and output artefacts it accepts and produces.

Failure and qualification. It must explain how it can fail or decline, what qualification applies to its result, and which positive and negative cases protect its scope.

Lifecycle. It must name the owner of its lifecycle and the active generation in which it is deployed.

These questions make the unit suitable for learning. A coding agent can revise one AKU without guessing where behaviour is hidden. An evaluator can compare two versions. An operator can identify which harnesses depend on it. A domain expert can inspect the instructions and sources. A governance reviewer can see why it was admitted.

An AKU is also more than a "skill" in the casual sense. Skills are often presented as convenience wrappers around tools. An AKU may invoke no external tool. Its primary value may be an interpretive rubric, a planning strategy, a context filter, a composition

policy, or a validation discipline. Conversely, an AKU that binds to exact code is still semantic because the binding expresses a claim: this procedure, under these assumptions, is an adequate realization of the responsibility.

Code and instruction as complementary procedure

A reliable harness should not force every task through a language model. It should also not imitate understanding with brittle phrase lists. The choice of medium depends on the uncertainty present.

Deterministic code is appropriate when semantics are stable enough to specify completely: parsing a formal language, enforcing a graph invariant, calculating a checksum, applying a schema, executing a solver, or normalizing a known data format. Natural-language instruction and managed model invocation are appropriate when the task requires interpretation, synthesis, comparison, explanation, or qualification under incomplete semantics.

Many capabilities combine the two. A literature capability may use code to query an approved database and deduplicate records, then use a model to classify relevance under a visible rubric. A policy capability may use a model to translate a request into a formal representation, a solver to check constraints, and another capability to verify that the translation preserves the user's meaning. The AKU makes the composition explicit.

The instruction resources deserve first-class version identity. If the meaningful policy lives inside incidental strings scattered through code, a reviewer cannot determine what semantic change a patch introduced. Named resources allow a concrete diff: the evidence rubric changed, the escalation rule changed, the definition of admissible source changed, or the composition guidance changed.

Operational roles in one representation

AKUs can serve several primary operational roles:

Within this common representation, an AKU may contribute knowledge, filter context, create or revise a plan, produce a task-specific value, validate or demonstrate that value, compose accepted artefacts while preserving disagreement and status, or govern retention, maintenance, and knowledge promotion.

These roles are not a hierarchy. A domain rule and a system planner share the same lifecycle discipline. A plan can be validated. A poor composition can reopen local processing. A validation result can request a different context filter. A proposed maintenance change can be evaluated before promotion. Uniform representation makes control procedures themselves open to inspection and revision.

Table 3. Primary operational roles of Agentic Knowledge Units.

Role	Effect and typical use
Knowledge	**Effect at the runtime boundary.** Adds reusable knowledge or instruction to a local invocation. **Typical use.** Definitions, rules, examples, conventions.
Context filtering	**Effect at the runtime boundary.** Selects relevant capabilities, sources, variants, and material. **Typical use.** Semantic, rule-based, source-constrained filtering.
Planning	**Effect at the runtime boundary.** Creates or revises the visible dependency structure. **Typical use.** Initial decomposition, checks, local refinement.
Processing	**Effect at the runtime boundary.** Produces a task-specific provisional value. **Typical use.** Extraction, diagnosis, comparison, transformation.
Qualification	**Effect at the runtime boundary.** Determines whether a value is fit to propagate. **Typical use.** Schemas, tests, evidence, proof, review.
Composition	**Effect at the runtime boundary.** Combines accepted values while preserving status and disagreement. **Typical use.** Reports, recommendations, synthesis.
Retention / maintenance	**Effect at the runtime boundary.** Changes reusable capability content, metadata, priorities, or interpreters. **Typical use.** Promotion, deprecation, benchmarked revision.

Generated output is not reusable knowledge

The AKU model supports an important separation between reusable knowledge and task-local output. The position paper describes layered knowledge: baseline units, deployment-specific persistent units, and session-scoped units. The exact layers may evolve, but the principle is durable. A provisional model output should not be retrieved later as though it were an established fact. Even a result accepted for the current answer may be unsuitable for future use.

Promotion therefore requires a retention policy. A source-linked extraction may be retained while the narrative built from it remains transient. A domain expert's correction may become evidence for a revised rubric without copying the person's wording into the capability. A generated hypothesis may remain in a research session but never enter an approved knowledge collection.

This is memory governance in a strong sense. It treats memory not as maximal accumulation, but as governed status transition. The system should be able to answer: where did this knowledge come from, who considered the source authoritative, what transformation was applied, for which harness is it valid, and why may it influence future work?

The lifecycle of a capability

A mature AKU has a comprehensible lifecycle. It begins with a capability gap, not with a difficult conversation alone. A candidate is implemented in isolation. Its structure and declared resources are validated. It is tested locally, including negative-transfer cases. It is evaluated as part of a complete candidate generation against a baseline. It may be approved, rejected, revised, merged with another unit, retired, or archived. The history should preserve why the capability exists, not only its latest files.

This lifecycle turns an AKU library into a maintained set of reusable procedures rather than an archive of incident-specific fragments. Without discipline, agent systems accumulate prompts, memories, tools, and special cases until selection becomes unreliable and no one knows why a unit exists. Proliferation is not an improvement. Improvement requires consolidation: a reusable responsibility that explains variation with fewer, stronger procedures.

A useful rule is that a new AKU should be created only when it expresses a distinct responsibility that cannot be more coherently represented by improving an existing unit. The tests should include cases where the new behaviour must not activate. This makes scope a property that can fail, not merely a descriptive label.

Current implementation snapshot — the AKU contract

The current documentation describes AKUs as independent versioned folders with content-addressed identity, bounded matching and execution operations, declared instruction resources, optional code, local knowledge, tests, and a narrow host SDK. Predefined capabilities and learned overlays remain distinct; learned revisions do not silently overwrite baseline units. The reserved filter and planner identities keep selection and decomposition learnable while the runtime retains graph and authority integrity.

A capability library as an engineering asset

The deeper opportunity is that a capability library can become a language for maintaining intelligent practice. A procedure can be named, selected, composed into a plan, observed in a trace, discussed with a human expert, revised by a coding agent, compared against a baseline, and promoted as a versioned unit.

This changes software development. Teams do not only implement features. They cultivate executable practices. They ask whether a responsibility has the right boundary, whether its knowledge is authoritative, whether its rubric reflects expert judgement, whether its interpreter is appropriate, whether its tests capture negative transfer, and whether its use has earned broader reuse.

In this sense, AKUs are neither static knowledge objects nor autonomous agents. They are maintained units of institutional competence.

AKUs are where repeated success becomes maintained "how to do this." They let an organisation consolidate domain procedure without hard-coding every semantic decision, and they make model replacement, source revision, testing, ownership, and rollback part of the same capability lifecycle.

CHAPTER 8

Planning, Selection, and Local Control

A plan is visible work, not hidden thought

Language-model discussions often treat planning as private reasoning: a chain of thoughts the model produces before answering. MRP-VM uses the word differently. A plan is a public operational artefact. It names responsibilities, dependencies, expected outputs, and the structure by which work can be scheduled, inspected, and revised.

The distinction matters for both privacy and epistemology. Hidden neural computation cannot be made transparent by asking a model to narrate its thoughts. Such narration may be useful as a heuristic, but it is not guaranteed to be causally faithful. An executable plan makes a smaller, more defensible claim: these are the units of work the runtime actually scheduled; these inputs were provided; these artefacts were produced; these checks controlled propagation.

The plan is therefore an **executable explanation**. It explains how the harness intends to work, not everything that occurs inside each interpreter.

Meaningful granularity

Planning begins with a choice of granularity. Too coarse, and the plan becomes decorative. Too fine, and it becomes an unreadable simulation of cognition.

A step deserves its own named value when one or more of the following are true:

A step deserves its own named value when it must be inspected independently, reused by more than one downstream step, or qualified under a distinct regime. The same is true when a different interpreter may compute it or when it must be compared with an alternative.

A named boundary is also justified when the step may fail or be challenged without invalidating unrelated work, when it may be retained under a different policy, or when it exposes a meaningful boundary of authority.

Entity extraction inside a stable parser may remain internal. Source selection for a high-stakes conclusion may deserve an explicit node. Formatting a final paragraph may remain inside composition. A formal translation that determines what a solver proves should be visible and independently checked.

This granularity aligns the graph with responsibility. It is not a claim about the smallest mental steps. It is a claim about what the institution needs to see.

Plans expose omissions

A visible graph reveals missing work. If an interpretation depends on evidence but no evidence-collection responsibility exists, the omission becomes structural. If a user-facing recommendation has no qualification dependency, the risk is visible. If two independent sources can be gathered in parallel, the graph shows the opportunity. If a final composition has no path from an unresolved branch, the plan cannot truthfully claim completion.

This is one reason plans matter even when a powerful model could produce the answer in one call. The plan makes the system's claim inspectable before the final prose conceals the route. It creates a place to attach tests, sources, tools, and human gates.

Planning also improves communication among people. A domain expert can challenge the decomposition without reading implementation code. An operator can see which region is stalled. An evaluator can inject a fault into one node and measure whether the system localises recovery. A capability author can observe that a supposedly reusable step is never selected or always bypassed.

Local adaptation through planner frontiers

Some tasks cannot be fully decomposed at the start. The structure becomes clear only after a document is parsed, an external fact is retrieved, or a local method fails. A rigid workflow handles this poorly; a free agent loop handles it invisibly. MRP-VM proposes a middle path: explicit planner frontiers.

A planner frontier is a visible point in the graph where further decomposition is required. The planner may propose a replacement fragment. The runtime validates the fragment as a connected graph, checks dependencies and budgets, preserves successful sibling work, and atomically substitutes the accepted region. A rejected proposal leaves the prior checkpoint intact and records diagnostics.

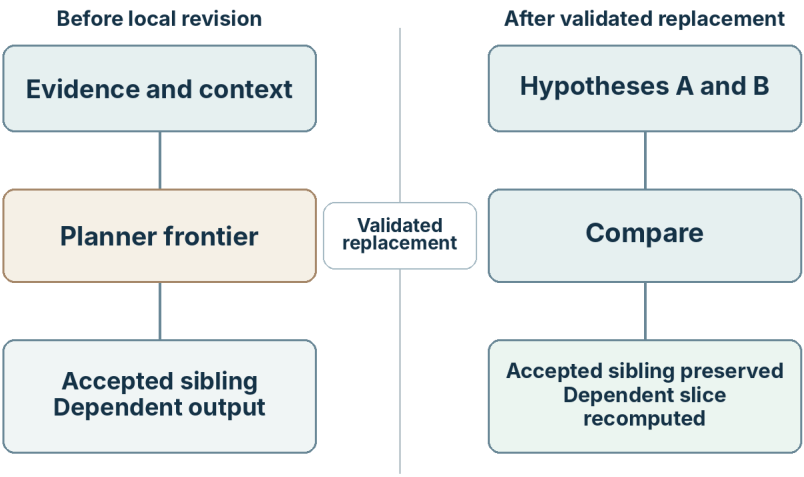


Figure 7. Local replanning replaces only the visible frontier, preserves accepted sibling work, and recomputes the dependent slice.

This structure turns iteration into an inspectable event. A retry loop often repeats the entire context with a slightly different prompt. A planner frontier says what changed in the representation of the problem. It distinguishes "the same method failed again" from "the system reframed the local task."

Selective recomputation

Strong dependencies support selective recomputation. Suppose a research synthesis contains accepted source extractions, a comparison, and a conclusion. A reviewer discovers that one source was misclassified. The system need not discard the entire session. It invalidates the affected extraction and the values that strongly depend on it. Independent work remains committed.

Selective recomputation has practical and epistemic value. Practically, it saves cost and time. Epistemically, it preserves a history of what remains justified. A global restart treats all prior work as equally suspect and can produce a different answer for unrelated stochastic reasons. Local invalidation relates revision to a specific changed premise.

The idea resembles incremental build systems and spreadsheets: change one input, rebuild the dependent artefacts. But natural-language work adds complexity because dependencies can be semantic and sometimes weak. The trace must record when a supposedly weak influence became materially important, and the qualification regime may decide that a broader slice requires reconsideration.

Replanning as controlled revision

At a deeper level, replanning is the architectural form of theory revision. A plan embodies a local theory of how the task should be solved. When reality resists—through failed validation, contradiction, missing evidence, or repeated state—the system can revise not only the answer but the decomposition itself.

This is the "meta" in operational form. A non-meta system adjusts a candidate within the current representation. A meta-rational system can also ask whether the representation is wrong. It may split a broad diagnosis into competing hypotheses, replace semantic review with formal checking, add a translation-verification step, or recognise that the request requires human clarification.

Replanning should remain bounded. Depth, time, resource, and repetition limits prevent indefinite reflection. The runtime should detect recurring states and stop with an explicit unresolved outcome when no meaningful progress is available. The ability to stop is as important as the ability to revise.

Current implementation snapshot — epochs and graph replacement
Current documentation describes iterative planning through validated replacement fragments and execution epochs. The planner owns semantic decomposition; the kernel owns structural validation, atomic replacement, budgets, and trace. A failed fragment does not corrupt the accepted graph, and partial failure preserves completed sibling work. The terms may change, but the invariant is local, accountable revision.

Planning and bounded autonomy

Planning is sometimes equated with autonomy. MRP-VM offers a more precise view. A system can plan flexibly while remaining bounded by purpose, authority, resources, and qualification. Conversely, a system can follow a fixed workflow while exercising dangerous authority through one opaque step.

Planning is trustworthy when each choice is represented, constrained, testable, and reviewable. Failures can then be localised to one responsibility, preserving accepted work while context or method changes without restarting an opaque transcript.

CHAPTER 9

Qualification, Evidence, and Human Oversight

Output is provisional until qualified

Language models are optimized to continue. Institutions need systems that know when a continuation is fit to influence later work. MRP-VM therefore treats the immediate result of a capability as provisional. The runtime records it, but dependent nodes should not consume it until an applicable qualification process accepts it for the present purpose.

This distinction separates generation from commitment. A rejected candidate can remain inspectable without contaminating the state. Several candidates can be compared. A technically successful tool call can still fail semantic qualification. A fluent explanation can be rejected for missing evidence. A correct calculation can be withheld because the input was not authorised.

Qualification is not a universal truth oracle. It is a judgement about fitness to propagate under a task contract.

Validation and demonstration

The word **validation** often suggests a binary test. Many tasks permit exact checks: schema conformance, type checking, unit tests, policy constraints, proof verification, numerical invariants, or deterministic replay. Where such checks exist, they should be used.

Other results require broader **demonstration**. A claim may be supported by source-linked evidence, a derivation, a proof object, an unsatisfied constraint, a counterexample search, agreement with a specialist, a replay against observed data, or a domain expert's review. The outcome may be qualified rather than timelessly true.

The position paper deliberately uses "demonstration" to widen the field. In natural-language tasks, the right question is often not "is this output true in the abstract?" but "has the system shown enough, under the relevant standard, for this result to be used in this context?" (MRP-VM Project, 2026).

Table 4. Qualification is plural because different claims require different demonstrations.

Form	What it establishes and its typical limitation
Schema or deterministic test	**What it can establish.** Structural conformity and executable invariants. **Typical limitation.** May miss semantic or contextual errors.
Source coverage	**What it can establish.** Whether material claims are supported by admissible evidence. **Typical limitation.** Coverage does not guarantee interpretation quality.
Derivation or proof object	**What it can establish.** That a conclusion follows within a formalised regime. **Typical limitation.** The formalisation may not match the original request.
Counterexample search	**What it can establish.** Whether an apparent rule fails on known or generated cases. **Typical limitation.** Failure to find a counterexample is not proof.
Replay against observations	**What it can establish.** Whether the result reproduces or explains recorded behaviour. **Typical limitation.** Past observations may be incomplete or shifted.
Specialist comparison	**What it can establish.** Agreement or disagreement with a stronger method or expert. **Typical limitation.** Shared assumptions can create correlated blind spots.
Human gate	**What it can establish.** Authorisation under institutional responsibility. **Typical limitation.** Human review can be inconsistent or overloaded.

Acceptance, rejection, and unresolved status

At minimum, qualification should distinguish three states.

Acceptance permits commitment for the present purpose. It does not necessarily authorise retention, publication, or action outside the node's contract.

Rejection prevents propagation and should record the likely reason: missing context, unsuitable method, interpreter failure, contradiction, incomplete plan, or a criterion the result does not satisfy.

Unresolved says that the available procedure cannot justify acceptance or definitive rejection. This is not a weak form of acceptance. It is a meaningful terminal or intermediate state.

The capacity to remain explicitly unresolved is philosophically and operationally important. Many systems are pressured to produce closure because interfaces expect a final answer. This pressure encourages fabricated confidence. MRP-VM instead treats uncertainty as structured information. An unresolved result can identify the missing source, contested assumption, unavailable tool, incompatible frames, exhausted alternatives, or required human decision.

In scientific work, unresolved may preserve competing hypotheses. In policy review, it may identify an ambiguity that only an authority can settle. In operations, it may trigger escalation. In education, it may expose a conceptual disagreement rather than flatten it into one answer. Honest non-resolution is a capability.

Closure is a policy
A system should not stop merely because a model emitted final-sounding text. It should stop when the required artefacts have reached an allowed terminal status under the task's closure condition: accepted, explicitly refused, or bounded and unresolved.

Plural validators and correlated failure

One model criticizing another model is not an oracle. The two may share training biases, source gaps, or stylistic preferences. A validation prompt can reward plausible form rather than substantive adequacy. More validators do not automatically solve the problem if they are correlated.

Qualification should therefore seek diversity of *regime*, not merely multiplicity of samples. Structural checks, source-coverage checks, formal solvers, test runners, alternative retrieval policies, domain rubrics, counterexample searches, and human review test different properties. The appropriate mix depends on risk and cost.

A low-risk transformation may need only schema validation. A consequential formal decision may require translation checking plus a proof object. A scientific synthesis may require source provenance, evidence coverage, and explicit preservation of disagreement. A recommendation that triggers side effects may require human authorisation even when the reasoning is otherwise accepted.

The runtime normally uses a preferred qualification regime. It opens additional checks when the plan requires them, the risk profile demands them, or the first qualification is doubtful. This preserves efficiency while making escalation principled.

Qualification can challenge the frame

A validator should do more than mark an output wrong. It should help locate the failure. If the decisive source was omitted, rerunning the processor with the same context is unlikely to help. If the method was inappropriate, a stronger model may repeat the conceptual mistake. If the plan lacks a required translation check, another candidate answer is beside the point.

Qualification can therefore reopen several choice points:

Qualification may therefore reopen context selection, the capability or method, the interpreter binding, the validation regime, the plan structure, the framing of the task, or the authority boundary that requires user clarification.

This diagnostic role links qualification to meta-rational control. Failure is information about the regime, not merely the candidate.

Human oversight as an executable control

Human oversight is often added as a generic instruction: "ask a human when needed." MRP-VM can represent human review as an explicit capability or gate with an input artefact, a decision contract, a trace event, and authority. The plan can state where human judgement is required and what information should be presented.

This improves both safety and usability. The person does not receive the entire internal transcript. They receive the bounded question: approve this side effect, resolve this source conflict, select between interpretations, or determine whether the residual uncertainty is acceptable. Their decision becomes part of the operational record.

Oversight also has limits. A person can be overburdened, misled by polished explanations, or asked to approve matters outside their expertise. Good design treats human review as a capability with scope and evaluation, not as a magical source of correctness.

Qualification and trustworthy AI

Trustworthiness is a system property, not a synonym for model accuracy. The NIST AI Risk Management Framework treats validity and reliability, explainability, transparency, accountability, safety, privacy, security, and fairness as contextual characteristics that must be governed across the lifecycle (NIST, 2023). The European AI Act likewise places emphasis on risk management, data governance, technical documentation, record-keeping, transparency, human oversight, accuracy, robustness, and cybersecurity for high-risk systems (European Union, 2024).

MRP-VM does not satisfy these requirements automatically. It supplies potential control points: explicit responsibilities, traceable source and tool use, qualified artefacts, human gates, versioned capabilities, replay, and promotion evidence. A complete trace can still document a wrong execution. A validator can still fail. The contribution is architectural evidence, not regulatory absolution.

Why bounded non-resolution is necessary

The most mature system is not the one that always answers. It is the one that distinguishes when it knows enough to act, when another regime may resolve the doubt, and when the remaining ambiguity should be reported.

Stopping unresolved protects against two symmetrical errors: endless reflection and fabricated closure. A bounded harness can terminate because alternatives are exhausted, the same state recurs, an external fact is unavailable, the required authority is absent, or the disagreement itself is the result. The stop reason is then part of the answer.

Qualification gives LLM output a governed path to consequence. It permits semantic judgment where necessary, exact checking where possible, human approval where responsibility requires it, and explicit non-resolution where evidence does not justify commitment.

CHAPTER 10

Operational Trace, Replay, and Auditability

Trace is not transcript

A transcript records what was said. A trace records what the system did. The difference is decisive.

A useful trace includes the request identity, session, pinned capability generation and settings, selected capabilities, plan and graph history, dependencies, model and tool invocations, authorised effects, provisional and committed artefacts, qualification events, retries, invalidations, planner replacements, budgets, stop reasons, retention decisions, and publication provenance. It may include references to prompts or model responses under protected access, but it does not need to expose private chain-of-thought.

The trace is generated during execution rather than reconstructed afterward. This makes it accurate with respect to the orchestrated process, though not necessarily with respect to truth. It can tell us that a particular source was selected and a validator accepted the result. It cannot by itself prove that the source was sufficient or the validator correct.

This modest distinction protects against "explanation theatre": a fluent post-hoc story that resembles the process but cannot be tested against it.

The operational record

The MRP-VM philosophy describes traceability as operational record. The phrase is apt because a long-lived harness needs to remember not only content but transitions: what was attempted, under which version, what changed, what failed, what was preserved, and why a candidate was promoted or rejected.

Operational record serves several temporal functions.

During a request, it supports status, cancellation, local recovery, and explanation. After a request, it supports review, feedback, and replay. Across requests, it reveals recurrent capability gaps, expensive paths, selection errors, and validator disputes. During learning, it provides bounded evidence. During governance, it supports accountability for publication and rollback.

A trace also records absence. A selected capability that was never used, a plan fragment that was rejected, a check that could not run, or a candidate that failed a promotion gate

can explain behaviour as much as an executed step. Absence is often where overconfidence hides.

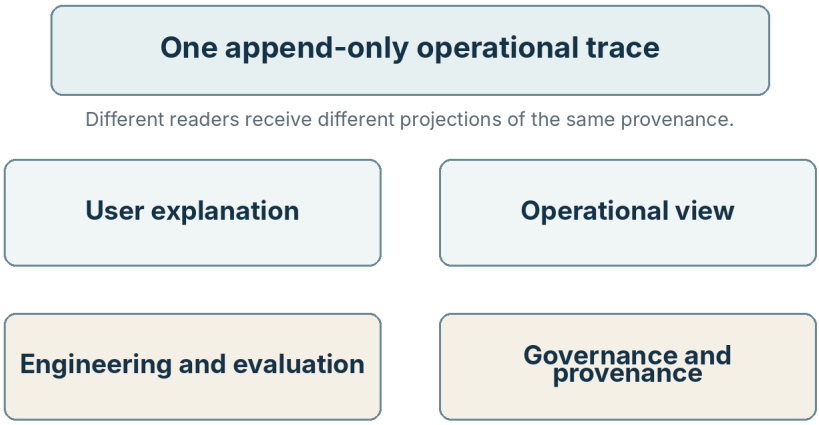


Figure 8. One operational trace supports different projections for users, operators, engineers, evaluators, and governance reviewers.

Explanations for different roles

Different readers need different views of the same record.

The **user layer** should provide a concise, truthful account of what was attempted, which sources or tools materially supported the result, and why the outcome is qualified or limited.

The **operational layer** should show selected capabilities, plan shape, current state, budgets, health, effects, typed errors, and recovery actions.

The **engineering layer** should expose capability and instruction versions, interpreter bindings, dependency values, graph history, replay inputs, and performance information.

The **evaluation layer** should compare baseline and candidate behaviour across cases and metrics without unnecessarily revealing private request content.

The **governance layer** should record authority, consent, retention, review, evaluation evidence, publication, and rollback.

These are not separate telemetry systems. They are access-controlled projections of one provenance model. This matters because reconciling independent logs after an incident produces uncertainty about which record is authoritative.

Table 5. One provenance model can support different trace projections.

Reader	Primary questions and appropriate projection
User	**Primary questions.** What happened, what supports the result, and what remains uncertain? **Appropriate projection.** Compact explanation, sources, status, next action.
Operator	**Primary questions.** What is running, waiting, failing, or consuming resources? **Appropriate projection.** Node state, effects, retries, budgets, stop reason.
Engineer	**Primary questions.** Which versions, contexts, interpreters, and dependencies produced the outcome? **Appropriate projection.** Detailed events, artifacts, timings, graph changes.
Evaluator	**Primary questions.** How did candidate and baseline differ, and which cases regressed? **Appropriate projection.** Replay, metrics, holdouts, ablations, failure slices.
Governance reviewer	**Primary questions.** Who authorised sources, actions, retention, and publication? **Appropriate projection.** Provenance, approvals, policy versions, audit trail.

Replay makes traceability testable

A trace becomes a testable claim when the system can replay a request under the archived capability generation and settings, substitute recorded model responses where deterministic reproduction is impossible, execute in a disposable environment, and compare structural results with the original.

Replay does not mean that stochastic models will reproduce every token. The target is the operational structure: selection, plan, effects, artefact identities, qualification, stop reason, and the relationship among them. Recorded responses can be used as fixtures when the purpose is to test orchestration rather than model drift.

Replay serves three important purposes. First, it verifies that the trace contains enough information to reconstruct the process. Second, it provides a baseline for candidate evaluation. Third, it lets investigators distinguish a capability regression from a changed external service, model provider, or source.

The current documentation treats fail-closed replay and pinned versions as central integrity properties. A replay that silently substitutes the newest capability or current settings is not reproduction; it is a new experiment.

Local failure as a repair record

A graph-aware trace changes the meaning of recovery. A failure is not merely an error string at the end of a transcript. It is an event at a particular node, under a particular version, after particular dependencies, with a particular attempted method. The system can preserve accepted sibling work, expose a planner frontier, record the replacement fragment, and show whether the next execution epoch made progress.

This produces a history of constructive revision. A reviewer can see that the source filter was changed, not the conclusion prompt; that the formal translation was rechecked, not the solver; that one branch remained unresolved while another was accepted. The trace becomes a map of contestability.

Such maps matter for learning. A future candidate should learn from a class of failure rather than copy the wording of the incident. Trace structure helps abstract the pattern: missing source authority, over-broad selection, absent qualification, repeated expensive route, or poor boundary between capabilities.

Privacy and proportional observability

More trace is not always better. Traces can contain sensitive inputs, source references, tool outputs, user feedback, and operational metadata. They can become expensive to store and burdensome to review. A mature system needs minimisation, access control, retention limits, redaction, and separate views.

The distinction between trace and chain-of-thought is important here. Trustworthy execution does not require publishing hidden model deliberation. It requires recording the public operational decisions and artefacts that affected the process. This approach is both more defensible and more useful. A private token-level rationale may be misleading; a recorded tool call, input artefact, version, and qualification event is concrete.

Traceability must also be usable. A dense event stream can remain opaque. The architecture must support summarisation without inventing facts, graph visualisation without hiding status, and explanations tailored to the reader's role.

Lifecycle accountability

Versioning tells us *what* was active. Trace tells us *what happened under it*. Together they create historical responsibility.

A team can say: this result was produced by this harness generation; this plan selected these capabilities; this source and tool were used; this artefact was qualified under this criterion; this feedback was authorised for learning; this candidate changed this practice; this evaluation supported adoption; and this prior generation remains available for rollback.

Such statements are modest compared with claims that an AI "understood" or "reasoned like an expert." They are also far more actionable. They let institutions investigate, contest, improve, and accept responsibility for systems in the open.

PART III

Automating Harness Construction and Improvement

A conceptual division of the MRP-VM argument.

CHAPTER 11

Evidence, Candidate Capabilities, and Evaluation

Improvement is a controlled engineering process

MRP-VM treats harness construction and improvement as an engineering lifecycle with distinct statuses and authorities. The online system serves the current request under a pinned generation. The improvement system studies authorised evidence, proposes candidate capabilities, tests them against the current baseline, and prepares a publication decision. This separation protects both the user and the integrity of the experiment.

The opportunity for automation is substantial. A coding agent can inspect recurring traces, compare failed and successful plans, draft AKU instructions, write contained helper code, construct tests, propose a better selection rule, or reorganise source material. A strong LLM can help formulate the reusable invariant behind several examples. What it cannot do is silently declare its own proposal active.

This distinction makes the process more ambitious, not less. Candidate work can be creative because it is isolated. The system can explore alternative procedures, model bindings, validators, and decompositions without changing the configuration used by active requests. Evaluation and publication determine which proposal, if any, becomes part of the maintained harness.

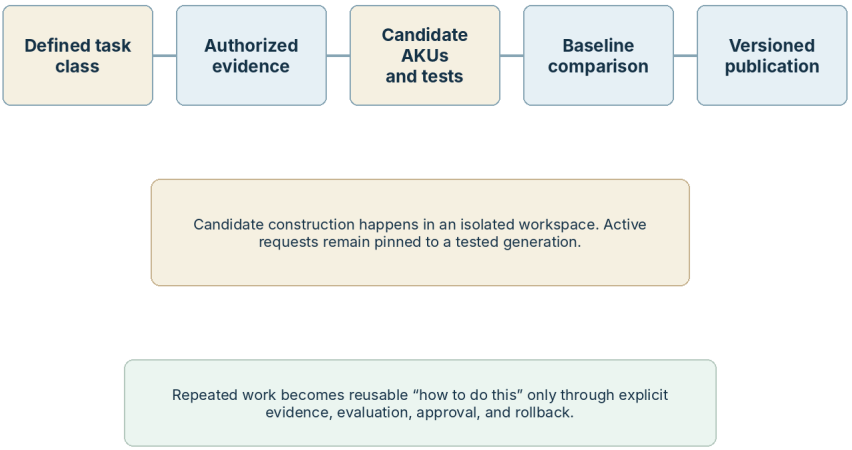


Figure 9. Active requests remain pinned to the published generation while authorised evidence enters an isolated candidate-construction and evaluation path.

Operational experience becomes authorised evidence

A harness learns from operational work only when the relevant information is explicitly authorised for that purpose. Requests, outputs, traces, ratings, corrections, human reviews, later outcomes, and marked regressions have different evidentiary meanings. A correction can repair one answer without establishing a general rule. A low rating can reveal dissatisfaction without identifying its cause. An expert annotation may be authoritative inside one organisation and inapplicable elsewhere.

MRP-VM therefore preserves distinctions among conversation content, request-local artefacts, execution trace, retained session summary, feedback, evaluation cases, and source-governed knowledge. Retention policy states which transitions are permitted. This is simultaneously a privacy control, a research-quality control, and a defence against the accidental conversion of fluent generated text into permanent instruction.

The result is a better substrate for trustworthy AI. The organisation can explain which evidence motivated a change, whether the data was permitted for that use, which groups or task slices it represented, and what information was deliberately excluded. The same discipline supports data minimisation, provenance, impact assessment, and later challenge.

Evidence analysis and capability-gap definition

Evidence analysis asks a precise question: what reusable responsibility is missing, weak, over-broad, or poorly qualified? The answer may be a new capability, but it may also be a narrower match rule, a better source boundary, a deterministic checker, a revised plan, a different model route, an additional human gate, or a decision to keep the current behaviour and improve documentation.

This step prevents incident memorization. A difficult request should not automatically produce another special case. The maintenance system compares cases, identifies common structure, searches for negative-transfer risk, and formulates a capability gap in terms that can be evaluated beyond the motivating examples.

The growing harness becomes a maintained account of how work is done. Repeated transformations, decision criteria, source rules, verification methods, and escalation conditions acquire stable names and tests. The library is not a transcript archive. It is an engineering representation of reusable practice.

Candidate construction in isolated workspaces

A candidate workspace gives human developers and coding agents a bounded place to implement a proposed change. The candidate may revise natural-language instructions, knowledge layouts, metadata, planner guidance, qualification logic, model routing, connector use, or interpreter code. Its dependencies and tests travel with it. The active generation remains unchanged.

Candidate authoring is where MRP-VM can increasingly automate the construction of custom harnesses. An initial harness can be bootstrapped from existing standard operating procedures, approved sources, examples, acceptance criteria, and expert interviews. Later candidates can use trace-derived evidence to refine the same capability catalog. In both cases, the output is not merely a better prompt; it is a versioned operational object with scope, resources, tests, provenance, and expected failure modes.

The coding agent contributes speed and breadth. It can generate variants, refactor overlapping units, add deterministic tests, write adapters, and explain the semantic policy it changed. Human and organisational authority remains visible through source approval, risk classification, evaluation design, security review, and publication.

A candidate should state the responsibility it improves, the invariant it attempts to encode, the evidence used, the expected benefits, the task slices that could regress, and the

conditions under which it must not activate. This makes review possible before production experience is asked to bear the cost of discovery.

Evaluation against a coherent baseline

A candidate is compared with the complete baseline generation rather than judged only on the example that inspired it. Structural conformance checks the capability contract. Targeted cases test the intended improvement. Negative-transfer cases test scope. Historical replay tests affected work. Full regression protects established behaviour. Security and policy tests exercise authority boundaries. A sealed holdout probes generalisation.

System-level metrics matter because a locally attractive change can degrade the harness elsewhere. Evaluation should examine task acceptance, context and capability selection, plan validity, qualification accuracy, calibration of unresolved outcomes, latency, model use, connector effects, privacy exposure, trace quality, and reviewer effort. Regulated deployments may add domain-specific safety, fairness, clinical, financial, or fundamental-rights measures.

> Automated construction is not autonomous activation
> MRP-VM can give coding agents broad freedom to propose instructions, code, tests, plans, and evaluation cases because active requests remain pinned and no candidate can publish itself. The more ambitious the construction automation becomes, the more explicit evaluation, approval, version identity, and rollback must become.

The governing question is not whether a candidate can now solve one example. It is whether the candidate produces a better, safer, and more intelligible harness for the declared task class without unacceptable regressions.

Holdout, regression, and negative transfer

Adaptive systems can overfit evaluation. If the authoring agent sees every test, it can optimize surface resemblance rather than the intended responsibility. If evaluation measures only final text, a candidate can improve style while weakening provenance, cost, scope, or oversight. Baseline, development cases, regression suites, and sealed holdout therefore need distinct roles and version identities.

Negative transfer is especially important for custom harnesses. A new procedure may improve the target slice but activate on unrelated cases. Selection tests must therefore include examples where the capability should remain inactive. This is one reason specialisation supports trustworthy AI: the system can test not only whether it knows how to act, but whether it knows when it is not entitled to act.

No suite eliminates gaming or blind spots. Human review, independent evidence, changing curricula, red-team work, and post-deployment monitoring remain necessary.

MRP-VM contributes a structure in which those activities can refer to a coherent generation and produce candidate changes that are themselves traceable.

Publication and operational monitoring

A candidate that satisfies its evaluation still requires explicit publication. Publication activates a complete generation: capability identities, knowledge resources, settings, model routes, tests, and rollback target. Requests already in progress remain pinned to the generation under which they began. Later requests may use the new generation according to deployment policy.

Operational monitoring then tests the assumptions of pre-deployment evaluation. Traces can reveal new failure patterns, cost changes, unexpected model behaviour, connector errors, reviewer burden, or distribution shift. These observations become new evidence; they do not automatically rewrite the system. Corrective action returns through the same candidate lifecycle.

Rejected candidates remain useful research artefacts. Their hypotheses, diffs, test results, and failure analysis can improve future design. The history shows not only what the harness adopted, but what it considered and declined.

How construction becomes increasingly automated

MRP-VM can automate more of harness construction as the organisation's specifications, sources, tests, traces, and governance become machine-addressable. Model-assisted analysis can propose capability boundaries; coding agents can implement candidate AKUs; planners can generate visible workflows; evaluators can compare generations; and publication services can activate only approved, coherent catalogs.

Automation does not remove human purpose. It concentrates human work where responsibility is highest: defining intended use, approving sources, setting risk tolerance, designing evaluation, authorising effects, reviewing consequential failures, and deciding what may be published. The machine handles more of the repetitive engineering while the organisation retains control of the claim it is making.

Better LLMs make candidate analysis and local execution more capable. MRP-VM makes those gains adoptable by testing the new interpreter against the previous generation, preserving the surrounding practice, and supporting rollback. Separated authority lets candidate authors experiment ambitiously while evaluation, approval, publication, and rollback remain distinct.

CHAPTER 12

Versioning, Publication, and Rollback

Unversioned adaptation is uncontrolled change

A system that changes but cannot identify its active state has not become adaptive in a trustworthy sense. It has become historically opaque. Versioning is the mechanism by which change acquires identity.

The relevant unit is not necessarily one file or prompt. A harness operates through a resolved configuration: capability versions, instruction resources, knowledge, model bindings, tool policies, budgets, isolation settings, retention rules, and interface policy. MRP-VM calls the capability component a catalog generation and associates it with a settings snapshot. A request pins the resulting operational configuration before selection begins (MRP-VM Documentation, 2026b).

This pinning answers a basic question: what system handled this request? Without it, the answer can change mid-execution as shared prompts, tools, or model settings drift. With it, the trace refers to one coherent configuration.

Coherent generations and rollback

Rollback is often described as restoring a previous file. In adaptive systems, that may be insufficient. A capability revision can depend on changed settings, neighbouring units, knowledge versions, or interpreter bindings. Reverting only one fragment may create a configuration that never existed or passed evaluation.

A complete generation makes rollback meaningful. The system can restore a tested operational configuration: the resolved capability set, relevant settings, and publication provenance. It can then compare outcomes without hidden mixtures.

This property supports research as well as operations. Experiments can be reproduced against archived generations. Baseline and candidate comparisons have stable identities. A regression can be localised to a transition. Historical analysis can ask whether improved quality came from a procedure, a model route, a knowledge source, or a validator change.

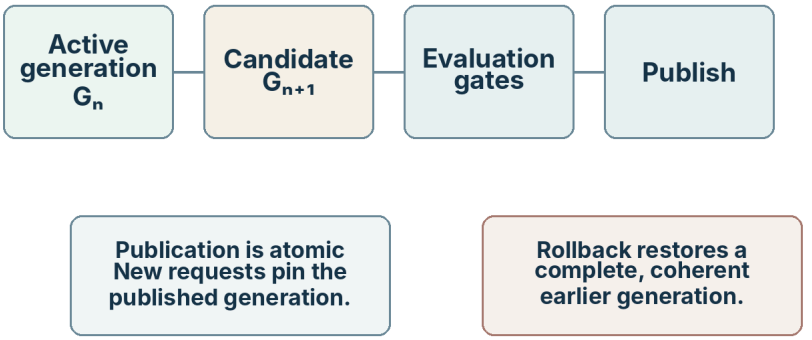


Figure 10. Candidate work becomes active only through evaluation and atomic publication of a coherent generation; rollback restores an earlier tested configuration.

Settings as deployment policy

Settings are often treated as a miscellaneous control panel. MRP-VM treats them as explicit deployment policy. They determine which semantic model roles map to which providers or local models, how much time and compute a request may use, how many isolated operations may run, which connectors are available, what may be retained, which model providers are permitted, and what authentication or streaming policy applies.

The distinction between semantic role and provider identity is particularly useful. A capability can request "fast interpretation," "coding," "deliberate reasoning," or "planning" without hard-coding a vendor. Deployment settings resolve those roles according to cost, privacy, availability, locality, or policy. The request snapshots the binding, so model choice remains traceable.

Settings should distinguish ordinary defaults from hard guardrails. A capability may request fewer resources than the limit, but it should not enlarge the safety boundary unilaterally. This preserves a hierarchy of authority: local procedures operate within deployment policy.

Technical access is not organisational permission. The ability to read a trace, start a candidate job, inspect cross-session material, publish a generation, or roll back production should be assigned explicitly by role and recorded as an authority decision.

Separated authority for proposal, evaluation, and activation

Accountable learning requires a division of authority. The exact organisational roles will differ, but the functions should remain distinguishable.

Users and domain participants provide goals, examples, feedback, and contextual knowledge. They do not automatically authorise permanent retention or generalisation.

Domain owners define purpose, source authority, acceptable outcomes, escalation, and risk.

Capability authors and coding agents implement candidate procedures, resources, code, and tests. They do not activate their own work.

Evaluators design and run comparisons, interpret metrics, and identify side effects. They should not quietly change the candidate to make it pass.

Operators manage infrastructure, model bindings, capacity, health, and deployments.

Governance reviewers or publication authorities decide whether evidence is sufficient for adoption under the relevant risk class.

The runtime enforces the transition rules and records them.

Table 6. Harness improvement remains governable when evidence, implementation, evaluation, approval, and activation are distinct acts.

Role	Legitimate action and silent failure to prevent
Evidence contributor	**Legitimate action.** Provide attributable feedback, examples, or failure reports. **Must not happen silently.** Convert feedback directly into active capability.
Capability author	**Legitimate action.** Propose instructions, knowledge, code, tests, and metadata. **Must not happen silently.** Declare the candidate successful by authorship alone.
Evaluator	**Legitimate action.** Compare candidate and baseline under declared suites. **Must not happen silently.** Change the suite opportunistically to favour a candidate.
Approver	**Legitimate action.** Accept or reject evidence for publication under policy. **Must not happen silently.** Edit the candidate while approving the same act.
Publisher / operator	**Legitimate action.** Activate a complete validated generation and preserve rollback. **Must not happen silently.** Mix active components from incompatible generations.

Role	Legitimate action and silent failure to prevent
Runtime	**Legitimate action.** Enforce state, budgets, isolation, transitions, and trace. **Must not happen silently.** Delegate its own authority to unbounded semantic output.

The same person may occupy several roles in a small team, but the system should still distinguish the actions. A developer editing a candidate is performing a different act from the same developer approving publication. The trace and interface should make the transition visible.

This separation reduces conflicts of interest and makes responsibility legible. It also scales. As a harness becomes more consequential, thresholds can become risk-sensitive: additional reviewers, stronger holdouts, formal checks, or mandatory human gates.

Authority at every transition

Governance is often added after a model has already produced a result. MRP-VM places authority at transitions throughout the lifecycle:

Authority is exercised when a request is admitted and bound to a session, when sources or connectors are accessed, and when tools are invoked or side effects are attempted. It is exercised again when provisional artefacts are committed and when outputs or feedback are retained.

The same discipline governs the creation of learning evidence, the execution of candidate code, access to sealed evaluation, and the publication or rollback of a capability generation.

Each transition asks two questions: *is this operation technically possible?* and *is it authorised for this harness, purpose, and actor?* The second question cannot be inferred from the first.

This principle is especially important for privacy. A session may contain sensitive context that is necessary for current work but prohibited from learning. A capability may be allowed to query an approved internal database without being allowed to export retrieved content into a candidate workspace. An evaluator may need aggregate metrics without access to raw cases.

Content-addressed identity and provenance

A capability's identity should reflect its contents or immutable version, not merely a mutable name. Content-addressed identity supports reproducibility: the trace can point

to the exact instruction, code, and knowledge used. A catalog generation resolves human-readable responsibilities to immutable capability versions.

Provenance extends beyond code. Knowledge should record source, authority, scope, transformation, review, and intended harness use. Instruction changes should identify the evidence and rationale. Evaluation cases should distinguish origin and learning-use permissions. Publication should record who approved the transition and which evidence was considered.

Provenance is not decorative metadata. It answers why an artefact was permitted to influence action. In an adaptive harness, "where did this come from?" is part of meaning.

Governance enables rapid experimentation

Governance is sometimes portrayed as friction that prevents systems from learning quickly. The opposite can be true. People are more willing to teach a system when they know how their feedback will be used, who can inspect it, whether it can become permanent, and how a harmful change can be reversed.

A trustworthy learning relationship requires visible status. The system should distinguish suggestion, evidence, candidate, evaluation result, active capability, and archived history. When these states blur, users cannot give informed consent and maintainers cannot accept responsibility.

The deeper human-centred definition of adaptation is not increased autonomy for its own sake. It is reduced distance between human intention, operational knowledge, and reliable action. People teach through examples, constraints, sources, and critique. Coding agents translate that teaching into candidate practice. The runtime keeps the boundary between contribution and activation unmistakable.

Current implementation snapshot — publication discipline

Current documentation describes candidate-only workspaces, complete generation staging, affected replay, full regression, sealed holdout, explicit publication, request-level generation pinning, and generation rollback. It also states that the current A/B/C learning experiment is planned rather than yet reported as completed evidence. This distinction between implemented mechanism and unproven outcome is part of the research discipline.

Lifecycle accountability

Versioning and governance give improvement a grammar. A team can state: the prior generation behaved this way; these traces revealed a capability gap; this candidate encoded a proposed invariant; this evaluation compared it with the baseline; these side effects were

observed; this authority approved publication; this generation activated the change; and this earlier generation remains a rollback target.

That statement is the practical meaning of answerability for change. It does not guarantee that the decision was wise. It makes the decision contestable, reproducible, and owned.

Coordinating specialist interpreters is how MRP-VM avoids a false competition between LLMs and exact methods. The LLM handles incomplete semantics and translation; deterministic and formal components handle stable relations; human judgment carries institutional responsibility. The harness makes those boundaries explicit and testable.

CHAPTER 13

Coordinating LLMs, Exact Tools, and Specialist Interpreters

Not every problem should be solved by the LLM alone

The broad competence of language models creates a powerful temptation: route every local task through the model and ask it to approximate whatever method would have been appropriate. This can work surprisingly often. It also discards the strongest property of mature computational methods—their specialised semantics.

A parser can deterministically recognise a formal grammar. A database can retrieve records under explicit query semantics. A symbolic solver can establish satisfiability or return a counterexample. A simulator can generate consequences under a model. A statistical procedure can quantify uncertainty under stated assumptions. A human expert can exercise institutional judgement and accept responsibility. A language model can interpret messy material, suggest structure, translate interfaces, synthesize prose, and operate where semantics remain incomplete.

MRP-VM coordinates these as a set of specialist interpreters. The runtime's intelligence lies partly in deciding which component is adequate for which bounded subproblem, what evidence that component must return, and how its result should be reintegrated into the plan.

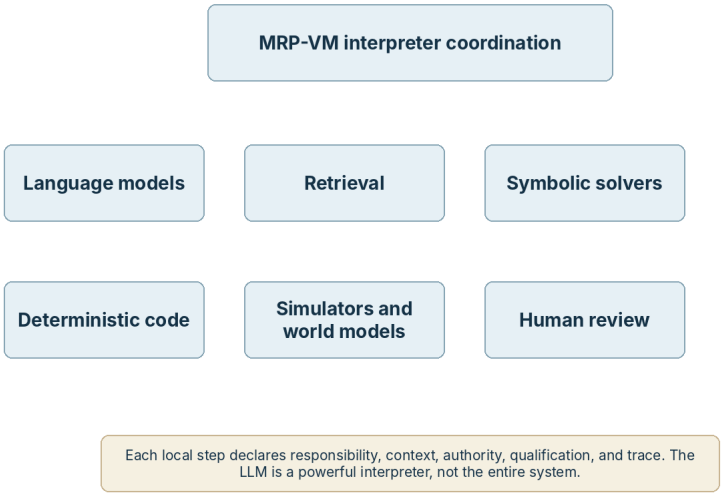


Figure 11. MRP-VM coordinates language models, deterministic code, retrieval, symbolic methods, learned routers, simulators, and human gates without asking any one interpreter to perform the entire task.

Structural regime selection

Tool choice is downstream of a more fundamental judgement: what is the computational character of the local problem?

A **structural regime** is a hypothesis about that character. Constraint-dominated problems may justify symbolic verification. Graph-structured problems may require relational decomposition. Similarity-driven problems may suit geometric retrieval. Uncertain causal processes may call for probabilistic models or simulation. Synthesis tasks may require programme generation followed by tests. Interpretive disputes may require multiple frames and human review.

Before calling an interpreter, the system must isolate entities, scope, dependencies, invariants, admissible evidence, and success criteria. It must stabilize the subproblem into an active frame for which solving is meaningful. Regime selection is therefore not merely a router choosing a tool name. It is the formation of a tractable problem representation (Alboaie, 2026i).

Tractable does not only mean fast. It means that a reliable local interpreter exists at acceptable cost and under an adequate validation regime. A hard problem may become tractable through narrowing, factorization, change of representation, approximation, or escalation to a specialist.

A coordinated interpreter stack

Different computational traditions contribute at different stages.

Language models are strong at moving from informal material to explicit structure: interpreting requests, proposing decompositions, mapping terminology, synthesizing candidate schemas, and composing explanations. They remain weak as universal engines for exact reasoning, stable long-horizon state, and proof of their own adequacy.

Classical machine learning can specialise recurring decisions: routing, boundary detection, confidence estimation, anomaly detection, ranking, and prediction under a stable task distribution.

Deterministic software is strongest where the procedure has been sufficiently stabilized: parsing, transformation, scheduling, authorisation, exact calculation, graph algorithms, and integration with mature tools.

Symbolic and neuro-symbolic methods provide explicit constraints, proof traces, compositional structure, and tractable reasoning where the representation supports them (Garcez and Lamb, 2020).

Human judgement remains necessary where authority, value, legitimacy, or unresolved contextual knowledge cannot be delegated appropriately.

The architecture is therefore not "LLMs versus classical methods." Flexible models expose structure and handle incomplete semantics; specialist learners support recurring routing decisions; deterministic software executes stable operations; solvers establish exact relations inside formal representations; and qualification governs the boundaries among them. Each method is used for the part of the problem where its assumptions are justified.

Orchestration requires explicit responsibility

Research on model cascades, routing, ensembling, debate, and multi-agent systems shows that inference-time organisation can improve quality and cost on bounded benchmarks. Cascades can invoke stronger models only when needed. Routers can exploit differences among models. Ensembles can combine complementary failures. Debate can prompt revision. Decomposition can make smaller models competitive on structured tasks (Chen, Zaharia, and Zou, 2023; Ding et al., 2024; Jiang, Ren, and Lin, 2023; Wang et al., 2024).

MRP-VM shares this interest in orchestration but makes a different claim. It is not primarily an ensemble wrapper or model router. It governs problem decomposition, interpretive regimes, capability identity, dependencies, qualification, trace, and learning. The question is not only "which model should answer?" but "what has this local

problem become, which interpreter is entitled to handle it, what demonstration is required, and what should happen if the frame fails?" (Alboaie, 2026n).

Diversity alone is not intelligence. Several weak or correlated agents can amplify error. A single strong model may outperform a poorly designed debate. The value lies in structure: decomposition, selective escalation, explicit validation, and the ability to diagnose where the route was wrong.

Progressive specialisation

Repeated execution can reveal stable patterns. A prompt-heavy sequence may become a deterministic transform. A model-based selection may become a small trained router. A recurring schema may become a typed intermediate. A costly chain may be cached. A broad capability may split into specialised units. A mature symbolic component may replace approximate reasoning.

This is a form of compilation. The system progressively converts expensive, weakly structured behaviour into compact, explicit, reusable competence while preserving a common execution grammar. Compilation here is not restricted to lowering code. It is the movement from improvisation toward maintained practice.

The process should remain reversible and evidence-driven. A deterministic replacement may be faster but too narrow. A small router may misclassify rare cases. A specialised solver may require a translation whose errors exceed the gain. Candidate evaluation must compare both quality and side effects.

This progressive specialisation is a sober account of self-improvement. The system does not rewrite a monolithic mind from first principles. It learns better decompositions, concept boundaries, routes, validators, and local interpreters.

Compatibility within a regime, choice among regimes

Energy-based models and Joint Embedding Predictive Architectures offer a useful analogy. They evaluate compatibility within a chosen representation: which configuration best fits constraints or predicted structure. MRP asks a prior question: why is this representation and compatibility function appropriate to the current subproblem? Which evidence and closure condition justify it? Should the regime be revised or combined with another? (LeCun, 2006; LeCun, 2022; Assran et al., 2023; Alboaie, 2026o).

A regulatory review may value provenance and admissibility more than latent similarity. A solver may value exact satisfiability. A user-facing explanation may value fidelity to a proof object and comprehensibility. A research hypothesis may be judged by evidence

coverage and discriminating predictions. There is no single universal scalar that captures all these regimes without losing their meaning.

MRP therefore generalises compatibility upward. It manages plural local standards and the transitions among them. Low "energy" in one regime may remain unacceptable because another required regime has not been satisfied.

Concept formation and reusable abstractions

The research agenda extends beyond routing existing interpreters. Intelligent systems may learn to propose better intermediate concepts, decompositions, and capability boundaries. The MRP article series calls this macro-AI: improving not only local representations or models, but also the controlled inquiry that decides which distinctions matter, which variables deserve to become concepts, and when a recurring solution pattern should become a reusable procedure (Alboaie, 2026p).

A useful concept has operational value when it reduces descriptive burden, organises variation, and makes inference cheaper. Scientific laws, types, schemas, and capability contracts can all serve this function. Automated conceptualisation would propose such structures and test whether they improve tractability and generalisation.

This research direction is ambitious and uncertain. A system may invent misleading abstractions, overfit its evaluation, or proliferate capabilities faster than people can review them. MRP-VM does not guarantee conceptual discovery; it provides a governed process in which candidate abstractions can be proposed, compared, rejected, and adopted without erasing their history.

Methodological competence

The deepest promise of interpreter coordination is methodological competence: the ability to recognise what kind of reasoning a local fragment requires, make the fragment tractable, choose an adequate interpreter and validator, and revise that choice when evidence exposes a mismatch.

Methodological competence means choosing the right interpreter and demonstration for a local claim, then revising that route when evidence exposes a mismatch. This differs from maximising one model call: it organises complementary methods so the result can be cheaper, more inspectable, and easier to explain.

PART IV

Adoption, Regulation, and Research

A conceptual division of the MRP-VM argument.

CHAPTER 14

Custom Agentic Harnesses for Repeatable Work

A method for selecting suitable applications

MRP-VM is easiest to misunderstand when applications are described only by domain: “an agent for science,” “an agent for law,” or “an engineering agent.” The architecture becomes concrete when each case is described as a bounded practice.

A useful design canvas asks nine questions:

A useful design canvas therefore fixes the harness's human purpose, admissible input family, output contract, authoritative knowledge sources, required capabilities and interpreters, permitted effects and approval gates, qualification of important artefacts, honest terminal states, and the evidence that may support future learning. Table 7 adds the associated ownership commitment and expresses each question as an operational design decision.

Table 7. A concise design canvas for a specialised harness.

Design question	Required commitment
Purpose	Name the class of work and the human value it serves.
Inputs and exclusions	Specify admissible inputs, missing-data behaviour, and out-of-scope cases.
Knowledge and provenance	Name legitimate sources, versions, and retention rules.
Output contract	Define artifacts, statuses, evidence, and audience.
Actions and authority	State permitted effects, transactions, and human gates.
Capability set	Identify reusable responsibilities and their composition.
Qualification	Match tests and demonstrations to the claims being made.
Terminal states	Define acceptance, refusal, escalation, and unresolved closure.

Design question	Required commitment
Learning evidence	State what may motivate revision and what may be retained.
Ownership	Assign maintenance, evaluation, approval, and rollback responsibility.

The cases below are not product promises. They show how the same conceptual architecture can support different operational ethics.

Incident triage: bounded flexibility

Consider incident triage for one production service. The input family is restricted to support tickets, a known log schema, recent deployment changes, service telemetry, and an approved runbook. The output contract requires a normalized incident summary, one or more plausible failure modes, supporting observations, a confidence-qualified next action, and explicit escalation when evidence is insufficient.

A monolithic prompt may work on familiar incidents but make it hard to determine why a diagnosis changed. A fixed workflow may be too brittle when different evidence is available. An MRP-VM harness can remain bounded while adapting its route.

The plan may select a short path when a known error signature is present and a longer path when logs need normalization or deployment history must be compared. A parser can handle structured logs. A retrieval capability can select the applicable runbook version. A semantic interpreter can formulate candidate diagnoses. A qualification capability can test whether each diagnosis explains the observed symptoms and whether the recommended action is permitted.

Suppose the preferred diagnosis ignores a recent deployment. Qualification can identify missing context, reopen selection, and rerun only the diagnosis and dependent recommendation. Accepted facts remain intact. If evidence still cannot discriminate among hypotheses, the final result can recommend a bounded diagnostic action or escalation rather than fabricate certainty.

The learning loop can then examine whether a filter routinely omits deployment changes, whether a smaller model suffices for normalization, whether one validator is redundant, or whether a recurrent failure pattern deserves a new exact detector. Improvement remains inside the task class.

Natural-language access to a symbolic core

A second pattern begins with a strong specialised interpreter. An organisation may already have a rule engine, theorem prover, constraint solver, or policy checker that is reliable within its formal representation. The difficulty lies at the boundaries: users speak natural language, the relevant rules must be selected, requests must be translated, formal results must be checked against intent, and explanations must be composed.

An MRP-VM harness can provide the agentic layer without imitating the solver's reasoning. The plan separates request interpretation, policy selection, formal translation, solver invocation, translation checking, and explanation. The solver capability returns a decision with a proof object, unsatisfied constraint, or counterexample. A qualification capability checks that the formalisation corresponds to the original request and that the artefact supports the reported conclusion.

This pattern is important because exact computation does not eliminate interpretation. A solver can be perfectly correct about the wrong formal statement. MRP-VM places the translation boundary under explicit review. It lets the user benefit from flexible language while retaining a reliable computational core.

Potential domains include access control, configuration validation, scheduling, compliance constraints, formal software checks, and research models whose local claims can be expressed symbolically. The architecture does not demand that the whole domain become formal. It escalates the fragment for which formalisation earns its cost.

Scientific evidence synthesis

Scientific synthesis illustrates the need for plural but disciplined frames. A bounded harness might support one research programme or a stable family of review questions. It would not claim to "do science" in general. Its inputs could include a protocol, approved databases, a source corpus, inclusion criteria, and a target evidence table. Its output might contain study selection, extracted claims, methodological limitations, disagreement, and a synthesis whose confidence is linked to source coverage.

The plan could separate search strategy, source eligibility, extraction, risk-of-bias assessment, claim normalization, comparison, and narrative composition. Exact tools can manage identifiers, deduplication, and statistics. Language models can interpret prose and map terminology. Qualification can check source linkage, coverage, extraction consistency, and whether the composition preserves unresolved disagreements.

The key memory rule is that generated synthesis does not become source knowledge. Retained knowledge should remain linked to authoritative publications and explicit transformations. A reviewer's correction may improve an extraction rubric without copying private review language into future output.

Such a harness could support scientific health by making assumptions and evidence regimes visible. It could also amplify error through incomplete databases, publication bias, shared model blind spots, or a misleading quality rubric. The trace and qualification structure make these limits inspectable; they do not remove them.

Software engineering and patch candidates

A software-engineering harness can combine semantic understanding with exact execution. Its purpose might be to diagnose and propose patches for one repository family under a defined test and review policy. Inputs include a repository snapshot, issue description, build configuration, and permitted tools. Outputs include a patch candidate, affected files, rationale, test results, remaining risks, and a stop reason if the issue cannot be reproduced.

Planning can separate reproduction, codebase mapping, hypothesis formation, patch synthesis, static checks, tests, and explanation. A coding agent may operate inside a task-local sandbox. The runtime controls file access, command execution, time, network, and artifact export. Tests and linters provide exact qualification where available. Human review remains an explicit gate before merge or deployment.

Learning should not memorize the issue report. It may identify a reusable repository-navigation capability, a better failure-classification rubric, a deterministic parser, or a recurrent test gap. Candidate changes are evaluated across historical issues and negative cases where the new behaviour must not activate.

This application shows why coding agents and MRP-VM are complementary. The coding agent supplies powerful synthesis. The virtual machine supplies scope, state, authority, trace, evaluation, and publication discipline.

Policy and contract review

Policy and contract work is often described as text analysis, but its operational structure is richer. A bounded harness may review one contract family against approved templates and jurisdiction-specific sources. It can extract clauses, identify deviations, map them to policy concerns, and compose questions for human counsel. It should not claim universal legal judgement.

Different artefacts deserve different qualification. Clause extraction can be checked against document spans. Template deviation can be deterministic. Legal interpretation may require source-linked reasoning and human review. A final recommendation may remain unresolved when the document is ambiguous or the relevant authority is missing.

The harness's operational ethics are central: citation requirements, confidentiality, retention, jurisdiction, and the boundary between information and legal decision.

Specialisation allows these to be encoded as inspectable policy rather than hidden in a generic assistant.

Education as guided practice

An educational harness offers a different output contract. Its goal is not simply to produce the correct answer, but to support learning under a curriculum and pedagogical policy. It may diagnose a learner's current misconception, select an explanation or exercise, observe the response, and adapt the next step.

The capability library can make pedagogical practices explicit: Socratic questioning, worked examples, retrieval practice, misconception classification, feedback rubrics, and assessment. Qualification asks whether the intervention is age-appropriate, aligned with the curriculum, and supported by evidence—not merely whether the prose is fluent.

Session memory is especially sensitive. Continuity can improve teaching, yet personal data and inferred ability should have clear retention, consent, and access rules. Feedback from one learner should not silently reshape instruction for all. A evidence-analysis phase can identify under-covered concepts; candidate development can propose a candidate lesson capability; evaluation can test learning outcomes and negative transfer.

Creative and specification-driven work

AssistOS's specification-driven R&D vision extends beyond analytical tasks. A harness can turn specifications into documents, diagrams, software, media, or research artefacts while keeping the specification, plan, intermediate products, qualification, and revisions linked.

Creative work does not eliminate the need for governance. It changes the criteria. Qualification may concern fidelity to brief, structural constraints, source permissions, audience, style consistency, or human approval. Several alternatives may be preserved rather than forced into one "correct" output. The trace can record which specification and capability generation produced an artefact without exposing private model deliberation.

The capability gap remains relevant. A successful one-off design is not yet a reusable design practice. The learning system should identify the invariant—perhaps a composition method, a quality rubric, or a transformation pipeline—and evaluate it across varied briefs.

Choosing where MRP-VM fits

MRP-VM is most promising when the task class is bounded but non-trivial; intermediate artefacts matter; several interpreters or sources may be involved; local failure should be

repairable; explanation and review have value; and the system is expected to learn over time.

It may be unnecessary for a simple deterministic workflow, an ephemeral low-risk chat, or a task with no need for reuse or trace. It may be insufficient for highly reactive environments with continuous state, complex transactions, or open-ended physical action unless extended with stronger semantics.

The architecture should be adopted because its control points solve a real problem, not because every AI application needs a virtual machine. Meta-rationality includes knowing when not to become meta.

Custom harnesses are most valuable where work repeats, variation remains meaningful, and the consequence of error justifies maintained evidence. These are precisely the settings in which stronger LLMs can create major value, provided their flexibility is integrated with scope, tools, verification, human responsibility, and change control.

CHAPTER 15

AssistOS, Achilles, and the Research Agenda

A research programme, not an isolated component

MRP-VM is best understood within the wider AssistOS programme. AssistOS connects modular agent architecture, specification-driven research and development, and trustworthy AI. MRP-VM contributes the governed capability lifecycle through which those directions can be assembled into custom agentic harnesses that remain versioned, testable, traceable, and controllable as they improve (AssistOS, 2026a; AssistOS, 2026c).

The programme was launched by Axiologic Research and is advanced in the context of the European Achilles project. This institutional attribution matters because the research question is not only how to demonstrate an intelligent prototype, but how to move from experimental capability toward practical technologies that organisations can evaluate, operate, and trust.

Research conducted by Axiologic Research as part of the European research project Achilles is therefore the operating context of this book, not a decorative credit line. It places MRP-VM inside a programme concerned with the path from advanced models and formal reasoning to dependable human assistance.

Agent architecture and governed skills

AssistOS describes agents as orchestrators of skills. MRP-VM makes that orchestration more explicit by treating capabilities as versioned operational objects rather than informal prompt fragments. A capability is selected under evidence, placed in a plan, executed with bounded context and authority, qualified, traced, evaluated, and published as part of a coherent generation.

This relationship clarifies the place of powerful models. The LLM is a highly capable interpreter and collaborator inside the harness. It can help understand requests, formulate plans, generate artefacts, and author candidates. The identity of the harness remains its purpose, approved capabilities, sources, tools, qualification, settings, tests, and governance. A model provider can therefore change without silently redefining the entire system.

The result is a stronger agent architecture: one that benefits from model progress while retaining the ability to explain what the agent was allowed to do, which method it used, what evidence supported the result, and which version handled the request.

Specification-driven research and development

The AssistOS vision emphasizes specification-driven production: people describe desired artefacts and AI systems help compile those specifications into code, diagrams, theories, articles, designs, and operational outputs. MRP-VM supplies the control structure around that compilation. A specification becomes a bounded request; planning exposes the route; capabilities produce and qualify intermediate artefacts; trace connects the result to versions and sources; and repeated transformations can become reusable AKUs.

This is particularly important when the specification is incomplete or contested. The harness can preserve alternative readings, request clarification, route exact fragments to code or solvers, and record why one interpretation was accepted. It can also stop with a qualified unresolved state rather than turn ambiguity into confident invention.

Specification-driven work becomes trustworthy when interpretation itself is treated as work that can be inspected, evaluated, and revised. MRP-VM is designed to make that transition routine.

Trustworthy AI and AGISystem2

AssistOS's trustworthy-AI direction includes explainability, formal reasoning, logical consistency, and neuro-symbolic integration through AGISystem2. The Meta-Rational Pragmatics articles provide a theoretical basis for MRP-VM's pluralism of methods: no single representation, interpreter, or standard of proof is assumed to be universally adequate.

The relationship is complementary. AGISystem2 represents a route toward formal and neuro-symbolic reasoning. MRP-VM provides the harness layer in which a formal interpreter can be selected, supplied with an explicit representation, checked at the translation boundary, and combined with model-based interpretation, retrieval, deterministic code, and human review.

A solver can be exact and still receive the wrong formalisation. A language model can understand the practical distinction and still lack admissible evidence. Trustworthy AI requires both capable local methods and accountable transitions among them.

The wider AssistOS toolchain

AssistOS identifies a wider toolchain that includes development environments, reusable agent libraries, formal reasoning, deployment systems, command-line tooling, and skills. MRP-VM can operate as the governed intelligence lifecycle behind a simple agent or application: selection, planning, execution, qualification, trace, candidate construction, evaluation, publication, and rollback.

The boundary must remain explicit. A convenience API should not bypass generation pinning, capability checks, budgets, or trace. A host should not broaden the systems a connector can reach without review. A model abstraction should not hide which provider or version served a request. Integration should increase competence while keeping authority inspectable.

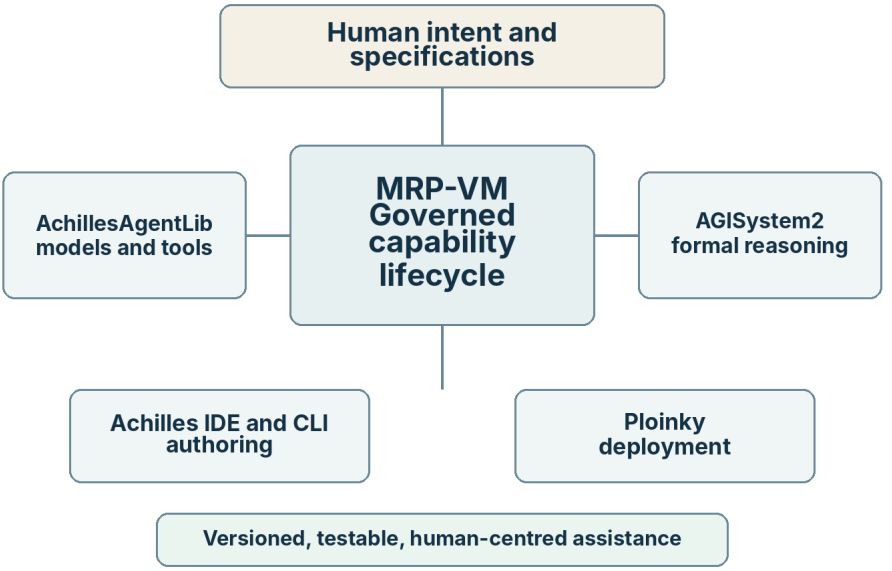


Figure 12. Within AssistOS, MRP-VM contributes the governed custom-harness lifecycle connecting specifications and user intent to models, tools, formal reasoning, deployment, and evaluated improvement.

This is why the user surface can remain simple while the builder and operator surface remains rigorous. A person may interact through chat, an API, a workflow button, or another application. Behind that interface, the harness retains the evidence and controls required for dependable operation.

Research conducted within Achilles

The Achilles context adds a practical imperative. Research claims become valuable when they survive contact with real workflows, constrained resources, privacy obligations,

security boundaries, domain evaluation, and the organisational need to explain and reverse change. Moving toward production readiness requires sharper intended-purpose claims and stronger lifecycle evidence, not only more features.

MRP-VM offers a coherent vocabulary for that transition. A prototype can begin with a small capability catalog and a simple interface. As the use case matures, the same concepts can support source governance, bounded connectors, specialised qualification, richer sessions, candidate workspaces, regression evaluation, approval, publication, monitoring, and cross-harness reuse.

The conceptual architecture is meant to survive this maturation. The implementation can become more sophisticated without abandoning the distinction among active execution, authorised evidence, candidate change, evaluation, and publication.

From prototype to evidence

A working prototype establishes feasibility, not superiority. The research task is to discover where MRP-VM offers a better trade-off than monolithic prompting, fixed workflows, sequential agent loops, or other orchestration systems. The strongest comparison keeps the underlying models and tools as similar as possible so that the architectural contribution can be isolated.

Evaluation should test the properties MRP-VM explicitly claims to expose: task acceptance, capability and context selection, plan completeness, qualification accuracy, calibration of unresolved outcomes, preservation of accepted work after local failure, reviewer localisation time, trace usability, latency, cost, unauthorised effects, negative transfer, and rollback integrity.

A symbolic-wrapper study should separately measure translation fidelity, solver correctness, proof or counterexample use, explanation fidelity, and the ability of qualification to detect a mismatch between natural language and formal input. A regulated-domain study should add the required sector evidence, human-factors evaluation, data governance, security, and post-deployment monitoring.

Research claims and current limits

MRP-VM improves visibility and revisability; it does not guarantee correctness. A complete trace can document a semantically wrong execution. A coherent plan can omit the decisive responsibility. A processor and validator can share the same blind spot. A human reviewer can approve a polished but misleading result. A candidate can pass a weak suite and fail under distribution shift.

Natural-language capabilities can remain ambiguous, and large catalogs can become difficult to select and maintain. Trace can burden reviewers or expose sensitive information if retention and projections are poorly designed. Side-effecting actions require authorisation, transaction, recovery, and security semantics beyond text generation. Third-party models, tools, and knowledge introduce supply-chain risk.

Automated candidate construction creates additional risks: benchmark overfitting, proxy optimisation, self-confirming evaluation, complexity growth, and code that is difficult to review. Versioning, isolation, independent evidence, sealed holdout, human publication gates, monitoring, and rollback are necessary controls, but their effectiveness must be measured rather than assumed.

Open research questions

The research agenda asks how traces and feedback can be converted into genuine reusable capability gaps without memorizing incidents; how semantic qualification can become more independent and rigorous; how the runtime can recognise the computational regime of a subproblem; and which trace projections help different roles without overwhelming them or leaking private information.

It also asks when capabilities can safely transfer among harnesses, how evaluation should adapt to regulatory risk, how concept boundaries can be proposed and tested, and where the current dependency-graph model needs stronger semantics for persistent state, transactions, event streams, cyclic interaction, or open-world action.

These are empirical and engineering questions. MRP-VM should be evaluated under the same discipline it asks of its harnesses: explicit claims, realistic comparators, ablation, independent evidence, and willingness to preserve a negative result.

A measured but ambitious agenda

The ambition is not to build one agent that absorbs every practice. It is to make it easier to construct many accountable custom harnesses, each with a clear intended purpose and a tested way of working. Such harnesses can share capabilities deliberately, adopt stronger models under evaluation, and federate through explicit contracts without becoming one opaque system.

This agenda is ambitious because it treats the organisation of intelligence as an engineering object. It is measured because it does not claim that the architecture is optimal everywhere. The research contribution will be established by mapping where explicit capability, plan, qualification, trace, and governed improvement produce materially better outcomes.

CHAPTER 16

Trustworthy AI, Regulation, and Regulated Adoption

Legal and standards snapshot — July 2026. This chapter is a research synthesis, not legal advice. Classification, application dates, guidance, standards, sector rules, and national enforcement must be verified for each deployment.

Trustworthy AI is a system property

Trustworthy AI is not a synonym for a high benchmark score, a well-known model provider, or an attractive explanation. It is a property of a complete socio-technical system in a defined context of use. NIST describes relevant characteristics as validity and reliability, safety, security and resilience, accountability and transparency, explainability and interpretability, privacy enhancement, and fairness with harmful bias managed. These characteristics must be balanced according to context and risk (National Institute of Standards and Technology, 2023, 2024).

The word system matters. A capable model can still be used with the wrong data, excessive authority, weak monitoring, ambiguous responsibility, or an interface that prevents effective human intervention. Conversely, a specialised harness can make a model more trustworthy for a bounded task by narrowing the intended purpose, controlling sources and actions, using exact tools where possible, qualifying consequential outputs, and preserving evidence across the lifecycle.

Trustworthiness is therefore a maintained claim. It begins with intended purpose and risk analysis, continues through design and validation, and extends into deployment, monitoring, incident response, change control, and retirement. A model upgrade, prompt revision, new connector, changed knowledge source, or altered threshold can all change that claim.

The European regulatory baseline

The European Union's AI Act establishes a risk-based legal framework for AI systems and general-purpose AI models. Prohibited practices and AI-literacy duties have applied since 2 February 2025, and obligations for general-purpose AI models have applied since

2 August 2025. Article 50 transparency obligations for certain AI systems and generated content become applicable on 2 August 2026 (European Union, 2024; European Commission, 2026a, 2026b, 2026c).

On 29 June 2026, the Council gave final approval to a regulation simplifying parts of the AI Act and fixing later application dates for high-risk rules: 2 December 2027 for stand-alone high-risk systems and 2 August 2028 for high-risk systems embedded in regulated products. Organisations should verify the published and consolidated legal text before relying on any transition date (Council of the European Union, 2026).

For high-risk AI systems, the core direction is already clear. Providers need a continuous risk-management process, appropriate data and data-governance practices, technical documentation, automatic logging, transparency and instructions for deployers, effective human oversight, and levels of accuracy, robustness, and cybersecurity appropriate to the intended purpose. They also need quality-management, conformity-assessment, registration, post-market monitoring, incident reporting, and corrective-action processes where the Act applies (European Union, 2024).

Deployers also have responsibilities. They must use the system according to instructions, assign competent people to human oversight, monitor operation, retain relevant logs under their control, and ensure that input data is relevant and sufficiently representative where they control it. Certain public-sector or essential-service uses may require a fundamental-rights impact assessment, and affected people may have transparency or explanation rights depending on the use and legal basis (European Commission, 2026b).

AI literacy is not optional preparation for a distant future. Article 4 requires providers and deployers to take measures that ensure a sufficient level of AI literacy among staff and other people operating systems on their behalf, taking account of their knowledge, experience, training, the context of use, and the people affected (European Commission, 2026c). A trustworthy harness therefore needs not only technical controls but role-appropriate instructions and training.

The AI Act does not turn every LLM application into a high-risk system. Classification depends on the intended purpose, role in the value chain, and use case. The engineering lesson is nonetheless broad: systems that may become consequential should be designed from the beginning to identify their versions, document their purpose, constrain their authority, retain appropriate logs, support oversight, and control change.

Data protection and automated decisions

Where personal data is processed, the General Data Protection Regulation remains central. Lawfulness, fairness, transparency, purpose limitation, data minimisation, accuracy, storage limitation, security, and accountability apply independently of whether the processing is called AI. Data protection by design and by default requires safeguards to be built into the system rather than added after deployment (European Union, 2016).

A data-protection impact assessment may be required where processing is likely to create high risk to people's rights and freedoms. Solely automated decisions that produce legal or similarly significant effects are restricted by Article 22 and require an applicable legal basis and safeguards. Human involvement must be meaningful rather than ceremonial when the law or risk profile requires it.

For an agentic harness, these duties affect context construction, retrieval, session memory, trace, feedback, evaluation data, and model-provider routing. The system should send only the information needed for the local responsibility, preserve the purpose and authority under which data was obtained, distinguish service data from learning evidence, and apply retention, access, redaction, and deletion policies to operational records.

MRP-VM's local contracts and source-governed AKUs create useful control points for data minimisation and provenance, but they do not automatically establish a lawful basis, complete a DPIA, or determine whether an automated decision falls under Article 22. Those judgments remain organisational and legal responsibilities.

What regulated industries require in practice

Regulated adoption usually demands more than formal compliance with one horizontal AI law. Financial institutions operate under DORA requirements for ICT risk management, incident reporting, operational-resilience testing, and third-party risk. Medical AI may be subject to the Medical Devices Regulation or In Vitro Diagnostic Medical Devices Regulation alongside the AI Act, with quality management, clinical or performance evidence, risk management, cybersecurity, and post-market surveillance. Public-sector deployments may add fundamental-rights, administrative-law, records, procurement, and explanation duties (European Union, 2022; Medical Device Coordination Group, 2025).

Voluntary frameworks and standards often become practical adoption requirements through procurement, audits, certification, or organisational policy. ISO/IEC 42001 establishes an AI management system based on governance and continual improvement; ISO/IEC

23894 provides AI risk-management guidance; ISO/IEC 42005 addresses AI system impact assessment. The NIST AI RMF organises work through Govern, Map, Measure, and Manage. None substitutes for applicable law, but each reinforces a lifecycle view of trustworthiness (ISO/IEC, 2023a, 2023b, 2025; National Institute of Standards and Technology, 2023).

A regulated organisation will ask concrete questions. What is the intended purpose and prohibited use? Which model, knowledge, tools, and settings formed the deployed system? What evidence supports performance on representative cases? How are human reviewers trained and empowered? How are errors detected, reported, investigated, and corrected? What happens when a model provider changes? Can the organisation reproduce the configuration that handled a consequential case?

These questions are where MRP-VM's architecture becomes commercially and institutionally relevant. It provides a common way to bind model capability to operational controls and to turn scattered evidence into a coherent lifecycle record.

How MRP-VM creates control points and evidence

MRP-VM does not make a deployment compliant by itself. It provides architectural control points and evidence structures that can support compliance, assurance, certification, and internal governance. The mapping is strongest when each harness is designed for a declared intended purpose and its evaluation is tailored to the risks of that purpose.

Generation pinning identifies the capability and settings configuration that served a request. AKU identity and provenance show which procedure, knowledge, model role, tool, and tests belonged to that configuration. SOP plans and local contracts expose how work was divided and what context and authority each step received. Qualification records why a result was permitted to propagate. Trace and replay preserve the operational facts needed for investigation.

Candidate workspaces, baseline comparison, sealed evaluation, approval, atomic publication, and rollback create change control. Connector contracts, isolation, budgets, and effect validation constrain authority. Trace projections can give users, operators, engineers, evaluators, auditors, and regulators different views of the same underlying provenance without exposing every internal detail to everyone.

The table below summarises the practical correspondence. The right-hand column should be read as a contribution to an evidence package, not as an automatic legal conclusion.

Table 8. Trustworthy-AI obligations and regulated-adoption expectations mapped to MRP-VM control points.

Requirement or adoption expectation	MRP-VM control point and evidence
Intended purpose and bounded scope	Harness purpose, admissible inputs, exclusions, output contract, capability catalog, settings snapshot, refusal and escalation states.
Risk management	Risk-sensitive capability selection, effect classes, budgets, closure policy, explicit unresolved status, targeted qualification, and documented residual risk.
Data governance and provenance	Source-governed AKUs, local context selection, provenance, retention policy, authorised feedback, and separation between service data, evidence, and reusable knowledge.
Technical documentation	AKU manifests, instructions, knowledge resources, interfaces, tests, SOP plans, model and tool bindings, generation records, and known limitations.
Logging, traceability, and replay	Append-only operational trace, generation pinning, settings snapshots, selected capabilities, dependencies, effects, artefacts, qualification, stop reason, and structural replay.
Transparency and explanation	Named intermediate artefacts, evidence-linked qualification, audience-specific trace projections, disclosed limitations, and contest or correction routes.
Human oversight	Explicit approval gates, role-based authority, readable state, pause/refuse/override/escalate actions, decision records, and operator-specific instructions.
Accuracy, robustness, and cybersecurity	Exact tools where semantics are stable, plural qualification, isolation, connector contracts, budgets, adversarial tests, regression, sealed holdout, and recovery policy.
Privacy and data protection	Minimum necessary local context, purpose-bound retention, access control, redaction, consented learning evidence, provider routing policy, and trace minimisation.

Requirement or adoption expectation	MRP-VM control point and evidence
Quality management and change control	Isolated candidate workspaces, baseline comparison, review, approval, atomic publication, coherent generations, version monitoring, and rollback.
Post-deployment monitoring and incidents	Trace telemetry, marked regressions, affected-slice analysis, incident records, corrective candidates, post-publication comparison, and reproducible configuration identity.
Third-party model and tool risk	Versioned model roles, provider and connector identity, declared permissions, recorded dependencies, replacement evaluation, and the ability to change interpreters without losing the harness contract.
AI literacy and role competence	Role-specific documentation, builder and operator views, training based on intended use and risk, explicit limits, and evidence that oversight roles can exercise their authority.

The value of the mapping is cumulative. No single trace event proves trustworthiness. A credible assurance case emerges from consistent intended-purpose definition, capability scope, data governance, evaluation, operational control, monitoring, and change history across the complete harness lifecycle.

Human oversight that can actually intervene

Human oversight is effective only when the person can understand the relevant system state, has time and competence to act, and possesses real authority to pause, refuse, override, request more evidence, or escalate. A generic instruction to "have a human in the loop" is not enough.

MRP-VM can express oversight as an explicit command or gate in the SOP graph. The gate can receive a focused explanation, supporting sources, qualification status, unresolved issues, and the precise effect awaiting authorisation. The decision and its authority become part of the trace. Higher-risk work can require multiple gates or independent review; lower-risk work can use sampling and retrospective monitoring.

The same design supports AI literacy. Builders need visibility into capabilities, sources, settings, tests, and failure modes. Operators need status, budgets, effects, and stop reasons. Domain professionals need evidence and the boundary of the recommendation. End users need understandable notice, limitations, and routes for contest or correction. One trace can support these different explanations when projections are designed deliberately.

A path to regulated adoption

A regulated adoption programme begins by defining the intended purpose, users, affected people, decisions, actions, data, and prohibited uses of one bounded harness.

This determines whether the task is suitable for automation, what legal and sector classifications may apply, and which properties require independent evidence.

The organisation then constructs a baseline generation from approved sources, existing procedures, model and tool policy, connector contracts, human gates, and an evaluation suite. MRP-VM makes these materials addressable as capabilities, settings, plans, and tests. The initial goal is not maximal autonomy; it is a system whose claim can be understood and challenged.

Validation compares the harness with realistic alternatives and tests representative, difficult, out-of-scope, adversarial, and negative-transfer cases. Domain quality, safety, fairness, privacy, cybersecurity, robustness, oversight, and operational resilience are assessed according to the intended use. Documentation records both performance and known limitations.

Deployment begins with constrained authority, trained operators, monitoring, incident procedures, and explicit escalation. Operational evidence is reviewed for distribution shift, new failure modes, model-provider change, reviewer burden, and unexpected effects. Improvements return through candidate construction and evaluation rather than being patched directly into production.

This staged path makes adoption faster in the long run because it avoids rebuilding governance after the system has become operationally important. MRP-VM turns trustworthy-AI work into part of the architecture and development workflow instead of a separate compliance document assembled at the end.

Replacing models without replacing responsibility

The relationship between MRP-VM and LLM progress is intentionally positive. A stronger model may improve interpretation, planning, extraction, synthesis, qualification, or candidate authoring. The harness can introduce that model through a versioned binding, replay representative work, compare quality and cost, test safety and negative transfer, and publish the change only when the complete system remains acceptable.

This protects organisations from both stagnation and uncontrolled novelty. They do not need to freeze an older model to preserve trust, and they do not need to accept every provider update as an undefined product change. The harness becomes the stable place where purpose and evidence persist while interpreters evolve.

Specialisation is therefore not a retreat from general intelligence. It is the means by which general intelligence becomes operationally credible for a defined practice. Better models

make the custom harness more capable; the custom harness makes better models more adoptable.

What MRP-VM does not guarantee

MRP-VM cannot determine legal classification, establish a lawful basis for personal-data processing, prove fairness, perform a clinical evaluation, certify cybersecurity, or replace a quality-management system. It cannot ensure that a human reviewer is competent or independent. It cannot make weak evidence strong merely by tracing it.

A deployment still needs domain governance, security engineering, data management, impact assessment, independent validation where appropriate, incident response, supplier management, contractual controls, and legal review. MRP-VM should integrate with these systems and produce useful evidence for them, not claim to supersede them.

Its contribution is nevertheless important. It turns many trustworthy-AI requirements from aspirations into explicit engineering objects: intended purpose, capability scope, source authority, model and tool identity, bounded execution, qualified outputs, human gates, operational trace, evaluated change, publication, monitoring, and rollback.

That is the route by which custom agentic harnesses can help regulated industries use increasingly powerful LLMs with more control rather than less. The goal is not to restrain model capability. It is to make capability dependable enough to deserve adoption.

Conclusion

Building better harnesses without replacing better models

The opportunity opened by large language models is larger than a new class of conversational interface. Models can become general semantic engines inside many kinds of work. The engineering challenge is to convert that power into repeatable capability without hiding purpose, authority, evidence, or change inside an expanding prompt and transcript.

MRP-VM addresses that challenge through custom agentic harnesses. A harness makes a bounded claim about a defined practice. It states which work is in scope, which sources and tools are legitimate, which model and interpreter roles may be used, which artefacts must be produced, what qualification is required, where humans must intervene, and how the system learns from operational evidence.

An answer is not yet a capability. A prompt is not yet a maintained procedure. A model is not the complete system. A generated output is not reusable knowledge. Feedback is not an active patch. A candidate is not a published improvement. These distinctions are not academic caution; they are the control structure that allows organisations to use more model capability with more confidence.

MRP-VM makes the "how to do this" of repeatable work consolidatable. Agentic Knowledge Units give procedures, knowledge, tests, and interpreters a versioned home. SOP plans make responsibilities and dependencies visible. Local contracts bound context and authority. Qualification determines what may propagate. Trace and replay make operational history inspectable. Candidate workspaces, evaluation, publication, and rollback make improvement a controlled engineering act.

The architecture is explicitly complementary to strong LLMs. Language models remain the most flexible interpreters in the system. Coding agents can accelerate capability authoring. New model generations can improve planning, synthesis, qualification, and software construction. MRP-VM preserves the harness's intended purpose, evidence, tests, and governance while those interpreters change.

This is why specialisation supports trustworthy AI. A custom harness can make a smaller and stronger claim, evaluate the work that actually matters, restrict sources and actions, route exact relations to exact tools, expose human oversight, and monitor the consequences of change. In regulated industries, these properties become part of the evidence required for risk management, documentation, logging, oversight, resilience, data protection, quality management, and post-deployment control.

MRP-VM does not guarantee correctness or compliance. A trace can record a wrong process. A validator can share the model's blind spot. A suite can overfit. Human review can fail. The architecture contributes control points and historical evidence; responsible deployment still requires domain validation, security, data governance, impact assessment, organisational accountability, and legal analysis.

The research claim is concrete: on repeatable, semantically demanding work, evaluated custom agentic harnesses may improve capability, cost, reviewability, repairability, and trustworthiness. The aim is to make strong models understandable, governable, improvable, and reversible. Better models strengthen the harness; better harnesses make them more adoptable.

APPENDICES

Design and Evaluation Criteria for Trustworthy Harnesses

A conceptual division of the MRP-VM argument.

Appendix A. Design Invariants for Trustworthy Harnesses

This appendix states the implementation-independent principles that should remain valid even if the current syntax, manifests, APIs, runtime language, or interface design changes. They can be used as an architecture-review checklist for future MRP-VM revisions and for systems inspired by the same research programme.

1. Separate answer from capability

A task-local response may be useful without becoming reusable practice. Promotion requires a named responsibility, scope, resources, provenance, failure modes, evaluation, ownership, and a lifecycle. Generated output must not be treated as capability merely because it is fluent or because a user approved one instance.

2. Keep authority stable during active execution

A request should run under a pinned operational configuration: capability versions, settings, model bindings, permissions, and budgets. Candidate learning must not change an active request. The mechanisms that admit code, enforce effects, record trace, and publish generations require a separate engineering authority from ordinary semantic adaptation.

3. Make meaningful work visible

The system should represent responsibilities and dependencies at the level needed for independent inspection, qualification, routing, reuse, recomputation, retention, or authority. Plans should be neither collapsed into one opaque call nor expanded into a ritual transcript of every micro-operation.

4. Treat interpretation as an operation

Context selection, framing, source authority, method choice, and regime selection can materially change the result. They should be attributable to explicit procedures, capabilities, metadata, or trace events rather than assumed to occur neutrally.

5. Use the minimum sufficient local context and authority

A ready unit of work should receive its local responsibility, direct dependencies, selected knowledge, bounded state, and permitted resources. It should not inherit the entire request history or ambient host authority unless the contract makes that necessary and reviewable.

6. Preserve provisional status

An interpreter's immediate output is a candidate. It should not release dependent work or enter persistent knowledge until an applicable qualification accepts it for the relevant purpose. Rejected and unresolved candidates may remain visible in the trace without influencing state.

7. Match qualification to the claim

Use exact checks where exact semantics exist. Else use source-linked demonstration, counterexample search, replay, specialist comparison, or human review as appropriate. Do not treat one generic critic as a universal validator. The validation regime is part of the capability claim.

8. Preserve honest non-resolution

A harness must be able to report that available procedures do not justify acceptance or rejection. Unresolved output should identify the blocked region, attempted alternatives, missing evidence or authority, and relevant stop condition. Final-sounding prose is not a closure criterion.

9. Reconsider locally

When context, method, interpreter, validator, or plan proves inadequate, reopen the nearest relevant choice point. Invalidate and recompute the dependent slice while preserving unrelated accepted work. Global restart should be a deliberate choice, not the default repair mechanism.

10. Record operational trace, not private thought

Trace should capture request identity, versions, selection, plan, dependencies, effects, artefacts, qualification, revisions, budgets, stopping, retention, evaluation, and publication. It need not expose hidden chain-of-thought. Explanations should derive from executed structure and artefacts.

11. Make replay possible

A trace should be a testable claim. The system should be able to reconstruct structural execution under archived versions and settings, using recorded model outputs or deterministic substitutes where needed. Replay should fail closed when required artefacts are unavailable.

12. Separate memory statuses

Request data, session continuity, generated output, source-governed knowledge, learning evidence, candidate capability, and active capability are different classes of object. Movement among them requires explicit provenance, retention, consent, transformation, review, and authority.

13. Separate invention from adoption

Coding agents and exploratory workflows may propose bold changes in isolated candidate spaces. They must not activate their own work. Evaluation and publication are distinct acts with independent evidence and authority.

14. Publish coherent generations

Activation should move a complete, tested operational configuration rather than isolated mutable fragments. Requests pin a generation before work begins. Rollback restores a coherent earlier configuration, including relevant settings and dependencies.

15. Evaluate the system, not only the answer

Candidate evaluation should include task quality, selection, plan validity, qualification, scope, negative transfer, cost, latency, resource use, safety, trace quality, preserved work, reviewer effort, regression, and rollback. The architecture's benefit must be demonstrated against realistic baselines.

16. Prefer specialised, federated claims

A harness should have a defined purpose, authority, knowledge boundary, output contract, evaluation, and learning policy. Cross-harness reuse should be explicit and re-evaluated in the receiving context. Shared infrastructure must not make semantic policy contagious.

17. Keep code and natural-language procedure complementary

Deterministic code should own stable, complete semantics. Model-mediated instruction should own work that genuinely requires interpretation, synthesis, or judgement. The choice should be visible and revisable. Neither medium should be used as ideology.

18. Keep the framework fallible

MRP-VM is a proposal, not a final doctrine. It should be possible to determine that a simpler workflow is better, that meta-level reconsideration adds no value, or that a task class exceeds the current graph semantics. The architecture must remain open to its own evaluation and revision.

19. Treat trustworthiness as a system and lifecycle property

Do not infer trustworthiness from one model, one benchmark, or one explanation. Define it for the complete harness under an intended purpose, with evidence across design, deployment, monitoring, change, and retirement.

20. Bind governance and compliance evidence to intended use

Risk, documentation, oversight, data governance, security, and evaluation must be proportional to the work the harness performs and the people it may affect. MRP-VM supplies control points; the organisation remains responsible for the substantive claim.

Appendix B. Evaluation Guide

Purpose

The goal of evaluation is to determine whether an MRP-VM configuration offers a better engineering and epistemic trade-off than credible alternatives for a bounded task class. Evaluation should isolate architectural contribution rather than reward access to a stronger model or a richer private dataset.

Comparator set

At minimum, compare four systems under equivalent model and tool access where feasible:

At minimum, the comparison should include a monolithic prompt or single model call, a fixed workflow with predefined steps, a sequential agent loop with tools, and the MRP-VM harness under study.

Additional comparators may include a domain-specific pipeline, a multi-agent debate system, a model router, or another graph-based orchestration framework. The comparator should reflect a serious alternative that an engineering team might actually deploy.

Case design

Build a case set with explicit strata:

The case set should contain ordinary in-scope work, difficult but still in-scope work, ambiguous cases that require clarification, and cases in which decisive evidence is missing.

It should also contain out-of-scope cases that require refusal or delegation, cases with injected local faults, cases designed to reveal negative transfer, cost- or latency-sensitive cases, and a sealed holdout unavailable during candidate development.

Document source, difficulty rationale, expected artefacts, admissible variability, and privacy status for every case.

Primary outcome families

Task outcome

Measure domain quality using criteria appropriate to the artefact: accuracy, completeness, source coverage, fidelity, test success, proof correctness, decision quality, or expert acceptance. Preserve disagreement among reviewers rather than averaging away contested criteria.

Selection and framing

Measure whether the harness selected a capability-complete but bounded set; whether authoritative sources were included; whether irrelevant capabilities dominated; whether the task was framed under the appropriate regime; and whether out-of-scope cases were recognised.

Planning

Measure graph validity, responsibility coverage, dependency correctness, parallelism opportunities, unnecessary decomposition, and the frequency with which plans omit qualification or authority gates. Evaluate whether planner revisions improve the graph or merely add retries.

Qualification and calibration

Measure acceptance, rejection, and unresolved decisions against domain review. Include false acceptance, false rejection, inappropriate certainty, missed escalation, and the added value of plural validators. Where proof objects or tests exist, distinguish formal correctness from translation fidelity.

Fault localisation and repair

Inject controlled errors into context, intermediate artefacts, interpreter outputs, or validators. Measure whether the system identifies the likely locus, which graph slice is invalidated, how much accepted work is preserved, and how many attempts are required to recover or stop honestly.

Explanation and trace

Give users, operators, engineers, and governance reviewers role-appropriate tasks. Measure time to answer questions such as: which source supported this claim, which capability changed, why was this effect authorised, what was recomputed, and why did publication fail? Assess both completeness and cognitive burden.

Efficiency

Measure latency, model calls, token use, tool calls, compute, memory, storage, and monetary cost. Compare median and tail behaviour. Record whether stronger models are invoked selectively or routinely and whether repeated work is cached safely.

Safety and authority

Test unauthorised connector access, excessive resource requests, side effects without approval, leakage across sessions, improper retention, generated-code escape, and publication bypass. Evaluate graceful failure when a permitted resource is unavailable.

Improvement quality

For candidate revisions, compare baseline and candidate on affected replay, full regression, negative transfer, and sealed holdout. Attribute gains where possible to knowledge, instructions, routing, planning, qualification, or interpreter code. Measure capability-library complexity and selection health after adoption.

Suggested core metrics

Table 9. Suggested core metrics for research on trustworthy custom agentic harnesses.

Metric	Operational definition
Task acceptance	Share of cases judged fit for the intended use by blinded reviewers or domain criteria.
Context-selection error	Share of failures attributable to omitted, irrelevant, or unauthorised context.
Local fault recovery	Share of injected local faults repaired without restarting accepted upstream work.
Preserved work ratio	Accepted node work retained after a challenged premise or plan fragment.
Reviewer localisation time	Time required to identify the first materially defective decision.
Qualified non-resolution	Rate and calibration of honest unresolved outcomes on underdetermined cases.
Resource cost	Latency, model calls, tokens, tool time, and monetary cost under a fixed quality target.
Negative transfer	Performance loss on unaffected task slices after a candidate publication.
Replay fidelity	Agreement between archived structure and structural replay under controlled substitution.
Rollback integrity	Ability to restore a complete earlier generation and reproduce its accepted behaviour.
Oversight effectiveness	Share of consequential cases in which reviewers received adequate information, had usable authority, and intervened appropriately when intervention was required.
Documentation completeness	Coverage of intended purpose, versions, sources, models, tools, tests, limitations, approvals, and monitoring evidence

Metric	Operational definition
	required by the deployment assurance case.
Data-governance integrity	Rate of unauthorised context use, retention-policy breach, provenance loss, or failure to separate service data from learning evidence.
Third-party change sensitivity	Performance and control impact when a model, provider, connector, or external knowledge source changes under a versioned replacement test.

Experimental integrity

Pin capability generations, settings, model identities, tool versions, and source snapshots. When stochastic model calls cannot be reproduced, store or substitute responses according to the question being tested. Use the same resource envelope across systems or report deviations explicitly.

Separate development cases from sealed evaluation. Do not allow the candidate author to inspect hidden expected answers. Where human judgement is required, use blinded review and record inter-rater disagreement. Pre-register primary metrics for higher-stakes comparative studies.

Reading the result

A successful result need not show that MRP-VM is best on every metric. The architecture may justify modest overhead if it materially improves fault localisation, reviewability, or controlled learning. Conversely, trace and planning overhead are not justified if they do not improve the properties the system is designed to expose.

The final question is a trade-off question: for this task class and risk profile, does the explicit capability lifecycle create enough value to warrant its complexity?

Appendix C. Glossary

Accountable growth

Improvement organised so that the harness can identify what changed, why it changed, how it was evaluated, who authorised activation, which generation used it, and how to return to a prior state.

Active theory frame

A bounded, goal-conditioned assembly of entities, assumptions, constraints, hypotheses, sources, tools, inference policies, and closure conditions that makes a local problem tractable.

Agentic Knowledge Unit (AKU)

A versioned home for a coherent reusable capability. It may include natural-language instructions, source-governed knowledge, code, metadata, declared resources, tests, provenance, and an optional interpreter binding.

Answer

A request-local result produced for one situation. An answer can be valuable without being reusable or appropriate to retain.

Authority

The legitimate permission to perform a transition or effect: access a source, invoke a connector, commit an artefact, retain feedback, run candidate code, inspect sealed evaluation, publish a generation, or roll back.

Authorised evidence

Operational or evaluation information whose provenance, purpose, consent or legal basis, access, and retention permit it to be used in analysis or candidate construction.

Baseline

The active or reference harness generation against which a candidate is compared.

Bounded harness

A custom operational environment defined by purpose, input family, output contract, capability catalog, knowledge sources, tools, permissions, evaluation, interfaces, settings, monitoring, and improvement policy.

Capability

A reusable claim about what a harness can do under specified conditions. It includes responsibility, interface, resources, provenance, tests, failure modes, composition, and ownership.

Capability-complete selection

A small working set of capabilities sufficient to carry the request truthfully, without loading a sprawling catalog of irrelevant or competing procedures.

Candidate

A proposed capability or configuration change created in isolation. A candidate is inactive until it passes evaluation, approval, and explicit publication.

Candidate workspace

An isolated environment in which people and coding agents may create and test proposed AKU, plan, routing, evaluator, knowledge, or interpreter changes without altering active requests.

Catalog generation

A coherent, versioned resolution of active capability units. Requests pin a generation before selection and execution.

Closure condition

The criterion under which a local or final task may stop: successful test, accepted qualification, human decision, explicit refusal, exhausted alternatives, or bounded unresolved status.

Coding agent

A model-centred system capable of editing code, instructions, tests, and related artefacts. In MRP-VM, coding agents may create candidate work but do not acquire publication authority.

Composition

The process of combining named artefacts into a larger result while preserving provenance, qualification status, disagreement, and unresolved branches.

Conformity assessment

A legally or organisationally defined process for demonstrating that a system meets applicable requirements before or during placing it on the market, putting it into service, or using it.

Connector

A named, host-governed capability for accessing an external system. Connectors expose bounded operations rather than ambient network or credential access.

Context filtering

The accountable selection of knowledge, capabilities, and local material visible to an invocation. Relevance depends on the current responsibility and risk, not only semantic similarity.

Custom agentic harness

A versioned system that combines models, tools, knowledge, procedures, tests, interfaces, human oversight, and governance to perform a repeatable class of work under a declared intended purpose.

Demonstration

Evidence that a provisional result is fit to propagate for the current purpose. It may include tests, schemas, sources, derivations, proof objects, counterexamples, replay, specialist review, or human approval.

Effect

An externally or internally consequential operation, such as reading a protected source, executing code, writing a file, sending a message, or changing state. Effects require explicit authority and trace.

Epoch

A current implementation term for a coherent phase of graph execution after an accepted local plan revision. The durable concept is versioned, inspectable progress across graph replacement.

Evaluation

Comparative testing of candidate and baseline across target cases, replay, regression, negative transfer, holdout, efficiency, safety, security, fairness where relevant, oversight, and trace quality.

Executable natural language

Natural-language specification made operational through a runtime that constrains context, interpretation, tools, effects, qualification, and continuation. It does not imply complete formal semantics.

Generation pinning

Binding a request to one coherent capability and settings configuration so that active behaviour does not change during execution.

Governance

The roles, policies, evidence, and authority transitions by which requests, effects, retention, candidate construction, evaluation, publication, monitoring, and rollback are controlled.

Human oversight

A role and mechanism through which a competent person receives appropriate information and real authority to monitor, pause, refuse, override, escalate, or otherwise intervene according to risk and policy.

Intended purpose

The specific use for which a harness is designed, supplied, or deployed, including users, inputs, outputs, decisions, affected people, environment, limitations, and prohibited uses.

Interpreter

The mechanism that realizes a local capability: an LLM, deterministic program, parser, retriever, solver, simulator, coding agent, learned model, human gate, or composite service.

Local contract

The bounded information and authority presented to one unit of work: responsibility, direct dependencies, selected knowledge, permitted resources, expected artefact, and applicable budgets.

Meta-rationality

The disciplined capacity to examine and revise the frame, method, representation, evidence regime, or closure condition under which local reasoning is conducted.

MRP

Meta-Rational Pragmatics: a research programme for governing interpretation, frame choice, regime selection, revision, and bounded execution under incomplete semantics.

MRP-VM

A Meta-Rational Pragmatic Virtual Machine: a governed runtime and candidate-development environment for executing explicit natural-language programmes through versioned capabilities and heterogeneous interpreters.

Negative transfer

Activation of a learned capability where it does not apply, causing unrelated tasks to degrade. Negative-transfer tests protect scope.

Operational trace

The versioned record of what the harness selected, planned, executed, qualified, revised, retained, evaluated, published, monitored, and rolled back.

Planner frontier

A visible point in an execution graph where further decomposition is required. A planner may propose a replacement fragment, while the runtime validates and applies it atomically.

Pragmatics

The study and operational handling of meaning as shaped by use, context, purpose, participants, and consequences. In MRP, pragmatics is constrained by evidence, authority, and effects.

Provisional value

An interpreter output recorded in the trace but not yet committed for downstream use. It must be qualified, rejected, or left unresolved.

Qualification

The process that decides whether a provisional value is fit to propagate for the present task. Outcomes include acceptance, rejection, and unresolved status.

Regime

A local mode of representation, evidence, computation, and validation appropriate to a class of subproblem, such as symbolic constraint solving, semantic interpretation, retrieval, probabilistic inference, simulation, or human judgement.

Regime induction

The higher-order capability to recognise the structural character of a subproblem, stabilize it into a bounded frame, and map it to an adequate interpreter and qualification regime.

Regulated adoption

The organisational and legal process by which a harness is validated, documented, approved, deployed, monitored, changed, and retired under applicable horizontal and sector-specific requirements.

Replay

Re-execution or reconstruction of a recorded request under archived versions and settings, with recorded model responses or controlled substitutes where necessary, to test the structural claim made by the trace.

Retention

The authorised transition by which an artefact moves from request-local output into session continuity, improvement evidence, or source-governed reusable knowledge.

Runtime

The authority-bearing substrate that parses programmes, maintains graph and state, schedules work, invokes capabilities, enforces resources and effects, records trace, controls continuation, and governs publication.

Session

A policy-bound container for continuity across requests. It preserves ownership and history while distinguishing conversation, execution, retained summary, feedback, and improvement authorisation.

SOP

A Standard Operating Procedure represented as an explicit programme of named responsibilities, local frames, and dependencies. The current implementation uses SOP Lang as a lightweight intermediate language.

Specialised harness

A custom harness that makes a bounded claim about one practice, with explicit purpose, authority, sources, tools, evaluation, and improvement policy. Specialisation is compatible with flexible planning and interpreter choice.

Third-party model risk

The operational, legal, security, continuity, and performance risk introduced by dependence on an external model, provider, tool, connector, or knowledge source.

Trace

An append-only operational record of versions, selection, plans, dependencies, effects, artefacts, qualification, revision, stopping, retention, evaluation, publication, and monitoring. It is not a hidden chain-of-thought transcript.

Trustworthy AI

An evidence-backed property of a complete socio-technical system in context, including validity and reliability, safety, security and resilience, accountability and transparency, explainability and interpretability, privacy, fairness, and effective human responsibility.

Unresolved

A qualified state indicating that available procedures do not justify acceptance or definitive rejection. It may be intermediate or terminal and preserves the reason for non-closure.

Weak dependency

A relation in which an upstream artefact may guide context or interpretation without creating the same automatic recomputation obligation as a strong dependency. Material use remains traceable.

References

Alboaie, S. (2026a). “AGI Will Not Be One Thing.” Meta Rational Pragmatics Article 01. AGISystem2. https://agisystem2.com/MRP/meta-rational-pragmatics-competitive-direction.html

Alboaie, S. (2026b). “Meta-Rational Pragmatics.” Meta Rational Pragmatics Article 02. AGISystem2. https://agisystem2.com/MRP/meta-rational-pragmatics.html

Alboaie, S. (2026c). “Meta-Rational Pragmatics in Context.” Meta Rational Pragmatics Article 03. AGISystem2. https://agisystem2.com/MRP/meta-rational-pragmatics-in-context.html

Alboaie, S. (2026d). “Meta-Rationality: Psychological Roots, Philosophical Lineages, and the Executable Turn.” Meta Rational Pragmatics Article 04. AGISystem2. https://agisystem2.com/MRP/meta-rationality-psychology-philosophy-executable-pragmatics.html

Alboaie, S. (2026e). “Meta-Rationality: Difficulties, Misunderstandings, and Internal Risks.” Meta Rational Pragmatics Article 05. AGISystem2. https://agisystem2.com/MRP/meta-rationality-difficulties-misunderstandings-risks.html

Alboaie, S. (2026f). “Meta-Rationality, AI, and the Health of Scientific Inquiry.” Meta Rational Pragmatics Article 06. AGISystem2. https://agisystem2.com/MRP/meta-rationality-ai-scientific-inquiry.html

Alboaie, S. (2026g). “A New Foundational Intuition for Neuro-Symbolic AI.” Meta Rational Pragmatics Article 07. AGISystem2. https://agisystem2.com/MRP/a-new-foundational-intuition-for-neuro-symbolic-ai.html

Alboaie, S. (2026h). “Executable Natural Language.” Meta Rational Pragmatics Article 08. AGISystem2. https://agisystem2.com/MRP/executable-natural-language.html

Alboaie, S. (2026i). “Regime Selection and Tractable Computation as Regime Induction.” Meta Rational Pragmatics Article 09. AGISystem2. https://agisystem2.com/MRP/regime-selection-tractable-computation-regime-induction.html

Alboaie, S. (2026j). “MRP-VM: An Implementation Path.” Meta Rational Pragmatics Article 10. AGISystem2. https://agisystem2.com/MRP/mrp-vm-implementation-path.html

Alboaie, S. (2026k). “Related Research for Meta-Rational and Executable Pragmatics.” Meta Rational Pragmatics Article 11. AGISystem2. https://agisystem2.com/MRP/related-research-meta-rational-executable-pragmatics.html

Alboaie, S. (2026l). “Grounding Without Premature Finality.” Meta Rational Pragmatics Article 12. AGISystem2. https://agisystem2.com/MRP/grounding-reconsidered-meta-rational-pragmatics.html

Alboaie, S. (2026m). “Semantic Pluralism and the Special Sciences.” Meta Rational Pragmatics Article 13. AGISystem2. https://agisystem2.com/MRP/meta-rational-pragmatics-semantic-pluralism-executable-formalization.html

Alboaie, S. (2026n). “Orchestration, Routing, and MRP-VM.” Meta Rational Pragmatics Article 14. AGISystem2. https://agisystem2.com/MRP/orchestration-routing-mrp-vm.html

Alboaie, S. (2026o). “Beyond Local Compatibility: Energy, JEPA, and the Meta-Rational Frontier.” Meta Rational Pragmatics Article 15. AGISystem2. https://agisystem2.com/MRP/beyond-local-compatibility-energy-jepa-mrp.html

Alboaie, S. (2026p). “Macro-AI: Automated Conceptualization and Recursive Improvement.” Meta Rational Pragmatics Article 16. AGISystem2. https://agisystem2.com/MRP/macro-ai-automated-conceptualization-recursive-improvement.html

AssistOS. (2026a). “AssistOS.” Axiologic Research. https://www.assistos.org/

AssistOS. (2026b). “The AssistOS Vision.” Axiologic Research. https://www.assistos.org/vision.html

AssistOS. (2026c). “Research Foundation.” Axiologic Research. https://www.assistos.org/research.html

Assran, M., Duval, Q., Misra, I., Bojanowski, P., Vincent, P., Rabbat, M., LeCun, Y., and Ballas, N. (2023). “Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

Basseches, M. (1984). Dialectical Thinking and Adult Development. Norwood, NJ: Ablex.

Brooks, R. A. (1991). "Intelligence without Representation." Artificial Intelligence 47 (1-3): 139-159.

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

Chollet, F. (2019). "On the Measure of Intelligence." arXiv:1911.01547.

Cornelio, C., Dash, S., Austel, V., Josephson, T. R., Goncalves, J., Clarkson, K. L., Megiddo, N., El Khadir, B., and Horesh, L. (2023). "Combining Data and Theory for Derivable Scientific Discovery with AI-Descartes." Nature Communications 14, 1777. https://doi.org/10.1038/s41467-023-37236-y

Cory-Wright, R., Cornelio, C., Dash, S., El Khadir, B., and Horesh, L. (2024). "Evolving Scientific Discovery by Unifying Data and Background Knowledge with AI Hilbert." Nature Communications 15, 5922. https://doi.org/10.1038/s41467-024-50074-w

Council of the European Union. (2026). "Artificial Intelligence: Council Gives Final Green Light to Simplify and Streamline Rules." 29 June 2026. https://www.consilium.europa.eu/en/press/press-releases/2026/06/29/artificial-intelligence-council-gives-final-green-light-to-simplify-and-streamline-rules/

de Garcez, A. d'Avila, and Lamb, L. C. (2020). "Neurosymbolic AI: The 3rd Wave." arXiv:2012.05876.

Dewey, J. (1938). Logic: The Theory of Inquiry. New York: Henry Holt.

Erol, K., Hendler, J., and Nau, D. S. (1994). "HTN Planning: Complexity and Expressivity." Proceedings of the Twelfth National Conference on Artificial Intelligence.

European Commission. (2026a). "AI Act: Regulatory Framework for Artificial Intelligence." Shaping Europe's Digital Future. https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai

European Commission. (2026b). "Navigating the AI Act." Shaping Europe's Digital Future. https://digital-strategy.ec.europa.eu/en/faqs/navigating-ai-act

European Commission. (2026c). "AI Literacy — Questions and Answers." Shaping Europe's Digital Future. https://digital-strategy.ec.europa.eu/en/faqs/ai-literacy-questions-answers

European Union. (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 (General Data Protection Regulation). Official Journal of the European Union.

European Union. (2022). Regulation (EU) 2022/2554 of the European Parliament and of the Council of 14 December 2022 on Digital Operational Resilience for the Financial Sector (DORA). Official Journal of the European Union.

European Union. (2024). Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act). Official Journal of the European Union.

Flavell, J. H. (1979). "Metacognition and Cognitive Monitoring: A New Area of Cognitive-Developmental Inquiry." American Psychologist 34 (10): 906-911.

Fleming, S. M. (2021). Know Thyself: The Science of Self-Awareness. New York: Basic Books.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. (2023). "PAL: Program-Aided Language Models." Proceedings of the 40th International Conference on Machine Learning.

Harnad, S. (1990). "The Symbol Grounding Problem." Physica D: Nonlinear Phenomena 42 (1-3): 335-346.

Hayes-Roth, B. (1985). "A Blackboard Architecture for Control." Artificial Intelligence 26 (3): 251-321.

International Medical Device Regulators Forum. (2025). Good Machine Learning Practice for Medical Device Development: Guiding Principles. IMDRF/AIML WG/N88 FINAL:2025.

ISO/IEC. (2023a). ISO/IEC 42001:2023, Information Technology — Artificial Intelligence — Management System.

ISO/IEC. (2023b). ISO/IEC 23894:2023, Information Technology — Artificial Intelligence — Guidance on Risk Management.

ISO/IEC. (2025). ISO/IEC 42005:2025, Information Technology — Artificial Intelligence — AI System Impact Assessment.

Jiang, D., Ren, X., and Lin, B. Y. (2023). "LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion." Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics.

Kahn, A. B. (1962). "Topological Sorting of Large Networks." Communications of the ACM 5 (11): 558-562.

Karpas, E., Abend, O., Belinkov, Y., Lenz, B., Lieber, O., Ratner, N., Shoham, Y., and others. (2022). "MRKL Systems: A Modular, Neuro-Symbolic Architecture That Combines Large Language Models, External Knowledge Sources and Discrete Reasoning." arXiv:2205.00445.

Kegan, R. (1994). In Over Our Heads: The Mental Demands of Modern Life. Cambridge, MA: Harvard University Press.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. (2024). "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines." International Conference on Learning Representations.

Kim, S., Moon, S., Tabrizi, R., Lee, N., Mahoney, M. W., Keutzer, K., and Gholami, A. (2023). "An LLM Compiler for Parallel Function Calling." arXiv:2312.04511.

Kuhn, T. S. (1962). The Structure of Scientific Revolutions. Chicago: University of Chicago Press.

Lakatos, I. (1978). The Methodology of Scientific Research Programmes. Cambridge: Cambridge University Press.

LeCun, Y. (2022). "A Path Towards Autonomous Machine Intelligence." OpenReview preprint.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). "A Tutorial on Energy-Based Learning." In Predicting Structured Data, edited by G. Bakir, T. Hofmann, B. Schölkopf, A. Smola, and B. Taskar. MIT Press.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." Advances in Neural Information Processing Systems 33.

Medical Device Coordination Group. (2025). MDCG 2025-6: Frequently Asked Questions on the Interplay between the Medical Devices Regulation, the In Vitro Diagnostic Medical Devices Regulation, and the Artificial Intelligence Act.

MRP-VM Documentation. (2026a). "MRP-VM Documentation." https://assistos-ai.github.io/mrp-vm-v1/

MRP-VM Documentation. (2026b). "Philosophy." https://assistos-ai.github.io/mrp-vm-v1/philosophy.html

MRP-VM Documentation. (2026c). "Runtime Architecture." https://assistos-ai.github.io/mrp-vm-v1/runtime-architecture.html

MRP-VM Documentation. (2026d). "Agentic Knowledge Units." https://assistos-ai.github.io/mrp-vm-v1/agentic-knowledge-units.html

MRP-VM Documentation. (2026e). "Iterative Planning." https://assistos-ai.github.io/mrp-vm-v1/iterative-planning.html

MRP-VM Documentation. (2026f). "Learning Lifecycle." https://assistos-ai.github.io/mrp-vm-v1/learning-lifecycle.html

MRP-VM Documentation. (2026g). "Internal AKU SDK." https://assistos-ai.github.io/mrp-vm-v1/aku-sdk.html

MRP-VM Documentation. (2026h). "Interfaces." https://assistos-ai.github.io/mrp-vm-v1/interfaces.html

MRP-VM Project. (2026). MRP-VM: A Meta-Rational Pragmatic Virtual Machine for Controlled Execution of Natural-Language Tasks. Position paper.

National Institute of Standards and Technology. (2021). Four Principles of Explainable Artificial Intelligence. NISTIR 8312.

National Institute of Standards and Technology. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1.

National Institute of Standards and Technology. (2024). Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1.

Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., and Stoica, I. (2024). "RouteLLM: Learning to Route LLMs with Preference Data." arXiv:2406.18665.

Peirce, C. S. (1877). "The Fixation of Belief." Popular Science Monthly 12: 1-15.

Popper, K. R. (1959). The Logic of Scientific Discovery. London: Hutchinson.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. (2023). "Reflexion: Language Agents with Verbal Reinforcement Learning." Advances in Neural Information Processing Systems 36.

Simon, H. A. (1957). Models of Man: Social and Rational. New York: Wiley.

Suchman, L. A. (1987). Plans and Situated Actions: The Problem of Human-Machine Communication. Cambridge: Cambridge University Press.

Wang, J., Wang, J., Athiwaratkun, B., Zhang, C., and Zou, J. (2024). "Mixture-of-Agents Enhances Large Language Model Capabilities." arXiv:2406.04692.

Winograd, T., and Flores, F. (1986). Understanding Computers and Cognition: A New Foundation for Design. Norwood, NJ: Ablex.

Wittgenstein, L. (1953/2009). Philosophical Investigations. 4th ed. Translated by G. E. M. Anscombe, P. M. S. Hacker, and J. Schulte. Oxford: Wiley-Blackwell.

Wolpert, D. H., and Macready, W. G. (1997). "No Free Lunch Theorems for Optimization." IEEE Transactions on Evolutionary Computation 1 (1): 67-82.

Xu, S., Li, Z., Mei, K., and Zhang, Y. (2024). "AIOS Compiler: LLM as Interpreter for Natural Language Programming and Flow Programming of AI Agents." arXiv:2405.06907.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. (2024). "Large Language Models as Optimizers." International Conference on Learning Representations.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. (2023). "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." Advances in Neural Information Processing Systems 36.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). "ReAct: Synergizing Reasoning and Acting in Language Models." International Conference on Learning Representations.

Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C., and Zou, J. (2024). "TextGrad: Automatic Differentiation via Text." arXiv:2406.07496.