Artificial Local Intelligence: Architecture and Reference Implementation

Autor: Wolfgang Stegemann

Affiliation: Independent Researcher

ORCID: 0009-0000-1346-1170

Chapter 1

The Engineering Problem

Artificial intelligence has advanced at an extraordinary pace. Systems that once performed narrowly defined tasks are now capable of writing software, analysing scientific literature, coordinating workflows, controlling robots and interacting with humans in natural language. Large language models, retrieval systems, planning algorithms and external tools have dramatically expanded what artificial systems can accomplish. From a functional perspective, modern AI has reached a level of versatility that would have seemed unrealistic only a few years ago.

Yet this remarkable progress conceals a less obvious fact. While the capabilities of individual components have changed profoundly, the organisational structure of most autonomous AI systems has changed surprisingly little. As models have become larger and tools more sophisticated, the underlying architecture has often remained centred on a single decision process that observes the environment, generates behaviour, evaluates alternatives and initiates execution. Additional mechanisms such as memory, planning modules or safety filters extend this process, but they rarely alter its fundamental organisation.

For many applications this architecture is entirely adequate. A system that answers questions, writes text or performs a short sequence of actions does not require a sophisticated organisational structure. Once the task has been completed, the interaction ends. The system begins the next task without carrying substantial operational responsibility from the previous one.

The situation changes fundamentally when artificial systems are expected to operate autonomously over extended periods of time.

Long-term autonomy introduces requirements that are qualitatively different from those of isolated task execution. An autonomous system must preserve its operational integrity while continuously interacting with an unpredictable environment. It must recognise changing conditions, allocate limited resources, learn from experience, regulate its own behaviour and remain controllable throughout its operational lifetime. Success is therefore no longer determined solely by solving individual tasks. It depends equally upon maintaining stable operation across thousands or even millions of consecutive decisions.

This shift transforms the engineering problem.

The central question is no longer how to construct a more capable reasoning engine. Instead, the question becomes how to organise the different responsibilities required for continuous autonomous operation.

These responsibilities are fundamentally different in nature.

One process must determine what is currently happening in the environment. Another must evaluate whether the system itself remains operationally healthy. A third must generate possible courses of action. A fourth must decide whether these actions should be permitted. A fifth must preserve the historical development of the system so that future behaviour can depend upon accumulated experience. Finally, another process must coordinate the execution of all these activities without becoming part of their decision making.

Although these responsibilities interact continuously, they answer different questions and pursue different objectives. Treating them as one integrated reasoning process inevitably increases internal complexity. As systems become more autonomous, it becomes progressively more difficult to understand why a particular decision was made, which internal process produced it, which process authorised it and which operational assumptions influenced it.

This is not primarily an algorithmic problem.

It is an architectural problem.

Engineering has repeatedly encountered similar situations. As systems become more complex, their organisation eventually becomes more important than the sophistication of their individual components. Operating systems separate process scheduling from memory management. Database systems distinguish query optimisation from persistent storage. Computer networks separate routing, transport and physical communication into independent layers. Modern software engineering relies on explicit separation of responsibilities because complexity can no longer be managed effectively within a monolithic design.

Autonomous artificial intelligence is approaching the same point.

Current research understandably concentrates on improving reasoning, learning and planning. These developments remain essential. Nevertheless, increasing computational capability alone cannot solve the organisational problems introduced by long-term autonomy. A more capable reasoning engine still requires an architecture capable of regulating, supervising and preserving its operation over time.

The distinction between capability and organisation therefore becomes increasingly important.

A language model may generate excellent behavioural proposals while remaining entirely unaware of the operational state of the system that invokes it. A planning algorithm may optimise task execution without considering whether sufficient computational resources remain available. A memory system may store previous interactions without distinguishing

between transient observations and historically significant operational events. Safety mechanisms may reject undesirable behaviour after it has already been generated, without participating in the organisational structure that produced it.

Each of these components may function correctly.

The architecture may nevertheless remain conceptually fragmented.

The objective of this book is not to introduce another reasoning algorithm, another planning method or another language model. It proposes a different organisational principle for autonomous artificial intelligence.

The central assumption is that long-term autonomy requires explicit architectural separation between fundamentally different organisational responsibilities. Operational self-regulation, behavioural generation, normative evaluation, historical continuity and execution coordination should not be regarded as different aspects of one computational process. They should be implemented as independent architectural components whose interaction gives rise to autonomous behaviour.

This perspective shifts the focus of AI engineering.

Instead of asking how a single system can become increasingly intelligent, it asks how multiple specialised architectural responsibilities can be organised so that intelligence remains understandable, controllable and capable of continuous development.

Artificial Local Intelligence, abbreviated ALI, is the result of this perspective.

ALI is not defined by a particular learning algorithm, language model or optimisation method. These technologies remain interchangeable implementation choices. The defining characteristic of ALI is its architecture.

The chapters that follow develop this architecture systematically. They begin with the engineering principles from which it is derived, introduce the individual architectural components and finally demonstrate how these components form a coherent framework for the construction of autonomous and controllable artificial intelligence.

Chapter 2

Design Principles

The engineering problem described in the previous chapter does not arise from insufficient computational capability. Modern artificial intelligence already possesses powerful methods for reasoning, planning, learning and communication. The difficulty lies elsewhere. As systems become increasingly autonomous, they must simultaneously satisfy requirements that are often treated independently: they must remain operationally stable, generate effective behaviour, adapt through experience, respect normative constraints and remain understandable to those who develop and supervise them.

These requirements cannot be fulfilled simply by extending an existing decision process. Every additional responsibility increases internal complexity and makes it more difficult to determine why a particular decision was taken, which internal assumptions influenced it and how the system will behave under changing conditions. The problem is therefore organisational before it is algorithmic.

The architecture proposed in this book is derived from a small number of engineering principles. These principles are independent of any particular implementation technology. They do not prescribe programming languages, learning algorithms or reasoning models. Instead, they describe the organisational properties that an autonomous system should possess if it is expected to operate reliably over long periods of time.

The first principle is **separation of responsibilities**.

Autonomous operation consists of fundamentally different activities. Observing the environment, evaluating operational integrity, generating behavioural alternatives, assessing their acceptability, preserving historical experience and coordinating execution are not variations of the same computational task. They pursue different objectives, require different information and evolve according to different criteria. An architecture that assigns these responsibilities to independent components is easier to understand, verify and extend than one in which they are combined within a single decision process.

The second principle is **local autonomy**.

Every autonomous system operates within a particular environment that defines its available resources, tasks, tools and constraints. Autonomy is therefore always local rather than universal. An assistant responsible for scientific literature, an industrial controller and a mobile robot may share the same architectural organisation while operating in entirely different domains. Their intelligence is determined not by unlimited generality but by the coherent interaction between a stable architecture and a well-defined operational environment.

The third principle is **continuous operational viability**.

Traditional software systems often regard successful task completion as the primary objective. Autonomous systems must satisfy an additional requirement. They must preserve the conditions that enable future operation. Resource availability, communication integrity, memory consistency and computational stability therefore become permanent concerns rather than exceptional circumstances. Operational viability is not an optimisation target but an organisational prerequisite for continued autonomous behaviour.

The fourth principle is **independent normative regulation**.

Generating behaviour and evaluating behaviour are different engineering activities. A planning mechanism should propose actions without simultaneously acting as its own judge. Likewise, a normative component should evaluate proposed behaviour without becoming responsible for generating it. Separating these responsibilities improves transparency and permits the normative system to evolve independently of behavioural planning.

The fifth principle is **historical continuity**.

An autonomous system develops through experience. This development requires more than storing isolated observations or conversation histories. The system must preserve complete operational episodes, including the situation in which behaviour was generated, the viability of the system at that moment, the normative evaluation that followed and the consequences of execution. Only such complete historical records allow behaviour to be reconstructed, analysed and improved over time.

The sixth principle is **architectural stability**.

Learning should modify behaviour rather than organisational structure. A system may refine planning strategies, adapt normative priorities and accumulate increasingly sophisticated knowledge throughout its lifetime. These changes represent development within the architecture. They should never alter the architecture itself. Stable organisational responsibilities provide the framework within which continuous adaptation becomes possible.

The seventh principle is **implementation independence**.

Architectural principles should remain valid regardless of the computational methods used to realise them. A behavioural component may employ a large language model today and a different reasoning technology tomorrow. A memory subsystem may use SQLite in one implementation and a distributed database in another. These technological choices affect implementation but not architecture. The organisational principles remain unchanged.

Taken together, these design principles suggest a different way of constructing autonomous artificial intelligence. Rather than concentrating intelligence within a single computational centre, intelligence emerges from the interaction of specialised architectural components, each responsible for one clearly defined organisational function. The resulting system is not less capable than a monolithic design. On the contrary, it becomes easier to analyse, validate and extend because every responsibility remains explicitly identifiable.

These principles provide the conceptual foundation for the architecture developed throughout the remainder of this book. The next chapter introduces the overall organisation of Artificial Local Intelligence and demonstrates how these principles give rise to the architectural components that define the ALI framework.

Chapter 3

Architectural Overview

The design principles introduced in the previous chapter describe the organisational properties required for continuous autonomous operation. They do not yet define a concrete architecture. The purpose of this chapter is therefore to translate these principles into an explicit architectural organisation.

The central idea of Artificial Local Intelligence is straightforward. Instead of concentrating all decision making within one computational process, the architecture distributes different organisational responsibilities across independent components. Each component performs one well-defined function, communicates through explicit interfaces and remains replaceable without affecting the responsibilities of the remaining architecture.

This organisation follows a simple engineering observation. Autonomous behaviour does not arise because one component performs many different tasks exceptionally well. It emerges because several specialised components cooperate while remaining organisationally independent.

The resulting architecture consists of five primary components and one interface layer.

At its centre lies the **Causal Core**. The Causal Core continuously evaluates the operational condition of the system. It answers one question only:

Can the system continue to operate safely and effectively under its current conditions?

The Causal Core does not generate behaviour. It does not plan actions. It does not evaluate norms. Its sole responsibility is to maintain an explicit representation of operational viability.

Behaviour itself is generated by the **Ego**.

The Ego receives observations from the environment together with the current operational state supplied by the Causal Core. Based on this information, it develops behavioural proposals that are expected to achieve the current operational objectives. These proposals may involve reasoning, planning, tool selection, information retrieval or communication. The architecture deliberately places no restrictions on the computational methods employed by the Ego. It may use symbolic reasoning, probabilistic planning, large language models or future reasoning technologies. These implementation choices remain independent of the architecture.

Generating behaviour, however, does not imply that the proposed behaviour should necessarily be executed.

Every proposal therefore enters the **Super-Ego**.

The Super-Ego performs normative evaluation. It determines whether a proposed action is compatible with the permanent constraints and adaptive norms governing the system. The Super-Ego neither generates behaviour nor executes it. It evaluates behaviour produced elsewhere. This separation allows normative development to proceed independently of behavioural planning and ensures that behavioural generation never becomes its own judge.

The fourth architectural component is **Memory**.

Memory preserves the complete operational history of the system. Unlike conventional storage systems that merely accumulate information, Memory records complete operational episodes. Each episode contains the observation that initiated the decision, the operational viability determined by the Causal Core, the behavioural proposal generated by the Ego, the normative evaluation performed by the Super-Ego, the executed action and its observable consequences. The accumulation of these episodes constitutes the developmental history of the system. Behavioural learning, normative adaptation and long-term analysis all depend upon this historical continuity.

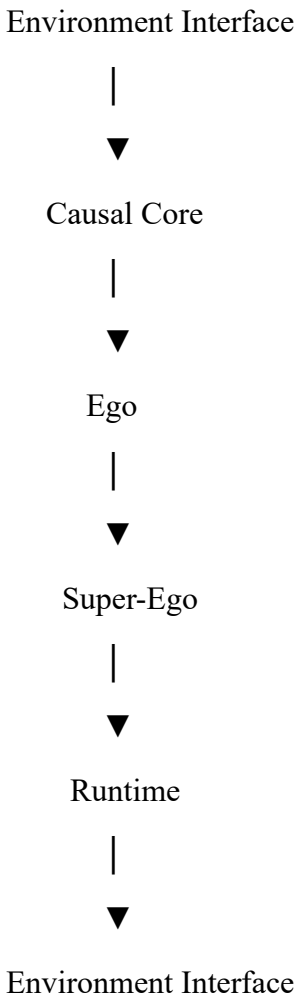
The fifth component is the **Runtime**.

The Runtime coordinates the interaction between all remaining architectural components. It controls the operational cycle, invokes the appropriate interfaces, records completed events and supervises the transition between operational states. Importantly, the Runtime possesses no behavioural intelligence of its own. It neither plans behaviour nor evaluates operational viability nor performs normative reasoning. Its function is purely organisational. It ensures that every architectural responsibility is executed in the correct sequence and that the resulting behaviour remains historically reconstructable.

These five components interact with the external world exclusively through the **Environment Interface**.

The Environment Interface isolates the architecture from its operational domain. Whether the system communicates with a robotic platform, a scientific document repository, an industrial production line or a local file system is irrelevant to the internal organisation of the architecture. The Environment Interface translates domain-specific interactions into a common architectural representation while preventing domain-specific details from propagating into the core architecture.

The complete organisation may therefore be represented conceptually as follows.



Memory communicates with all architectural

components through explicit interfaces but

remains organisationally independent.

At first glance this organisation may appear more elaborate than the architectures commonly employed by contemporary agent systems. In practice, however, each component performs fewer responsibilities than the corresponding modules of many existing systems. Complexity is reduced not by eliminating functionality but by distributing it according to clearly defined organisational principles.

This distinction is fundamental.

The architecture does not attempt to construct an increasingly sophisticated cognitive centre. Instead, it decomposes autonomous operation into a set of independent engineering responsibilities whose coordinated interaction produces intelligent behaviour.

As a consequence, every decision made by an ALI can be reconstructed completely. The system can determine which observation initiated the decision, how operational viability influenced planning, which behavioural alternatives were generated, why one proposal

satisfied the normative architecture while another did not, and what consequences followed execution. The architecture therefore provides transparency not by simplifying behaviour but by making the organisational origin of every decision explicit.

The chapters that follow examine each architectural component individually. Beginning with the Causal Core, they develop the complete organisational structure of Artificial Local Intelligence and demonstrate how continuous autonomous behaviour emerges from the interaction of these specialised components rather than from a single monolithic decision process.

Chapter 4

The Causal Core

The architecture introduced in the previous chapter separates autonomous operation into a number of independent organisational responsibilities. The first and most fundamental of these responsibilities concerns neither behaviour nor intelligence. It concerns the ability of the system to continue operating.

Every autonomous system exists under conditions that continuously change. Computational resources fluctuate, communication channels fail, sensors become unreliable, storage systems fill, external services disappear and environmental conditions evolve in unpredictable ways. An autonomous system that ignores its own operational condition may continue pursuing its objectives long after the conditions required for successful operation have disappeared.

This observation motivates the introduction of the Causal Core.

The Causal Core is the organisational centre of operational self-regulation. Its responsibility is not to decide what the system should do. Its responsibility is to determine whether the system remains capable of doing it.

This distinction appears subtle, yet it fundamentally changes the organisation of the architecture.

Most contemporary AI systems evaluate success primarily in relation to external objectives. If the objective is achieved, the system is regarded as successful. The operational condition of the system itself is usually represented only indirectly through resource limits, exception handling or monitoring software operating outside the decision process.

Artificial Local Intelligence adopts a different perspective.

Before an autonomous system can pursue any objective, it must preserve the conditions that make future action possible. Operational viability therefore becomes a permanent concern rather than an exceptional situation.

The Causal Core continuously evaluates this viability.

It does not observe the external world in order to understand it. It observes the operational state of the system itself.

This evaluation may include computational resources, memory integrity, communication quality, hardware availability, energy consumption, environmental accessibility or any other variables relevant to continued operation. The architecture deliberately avoids prescribing a fixed set of operational variables. Different implementations operate under

different conditions and therefore require different viability models. What remains invariant is not the content of the evaluation but its organisational responsibility.

The output of the Causal Core is not an instruction.

It is an assessment.

The assessment expresses the current operational condition of the system in a form that can be used by the remaining architectural components. Behavioural planning may therefore take place with explicit knowledge of the system's current operational state instead of assuming that this state is always satisfactory.

The Causal Core does not generate behavioural alternatives. It does not optimise task execution. It does not evaluate norms. It does not determine priorities.

Its responsibility ends once the current operational viability has been determined.

This deliberate restriction distinguishes the Causal Core from concepts that at first sight appear similar.

It is not a reward function.

Reward functions evaluate the desirability of behavioural outcomes and guide learning towards increasingly successful behaviour. The Causal Core performs no such optimisation. It neither rewards nor punishes behaviour. Its evaluation exists independently of any particular behavioural strategy.

It is not a utility function.

Utility functions rank behavioural alternatives according to expected value. The Causal Core does not compare behavioural alternatives because it receives none. Behaviour is generated only afterwards by the Ego.

It is not a goal management system.

Goals specify what the system attempts to achieve. Operational viability specifies whether the system remains capable of pursuing any goal at all. These are fundamentally different questions.

Nor is the Causal Core merely a monitoring subsystem.

Conventional monitoring collects operational information for external observation. The Causal Core produces operational information that directly influences every subsequent behavioural decision. It therefore forms an integral part of the architecture rather than an external diagnostic facility.

The distinction between operational viability and behavioural planning is one of the defining characteristics of Artificial Local Intelligence.

Every behavioural proposal generated by the Ego is conditioned by the current viability assessment supplied by the Causal Core. The Ego therefore plans behaviour under explicitly represented operational constraints instead of implicitly assuming that these constraints are always satisfied.

This organisational separation offers several engineering advantages.

First, behavioural planning becomes simpler because the evaluation of operational integrity has already been performed elsewhere.

Second, operational models may evolve independently of behavioural algorithms. Improvements in viability estimation require no modification of the planning architecture.

Third, operational reasoning becomes directly observable. Every behavioural decision can later be reconstructed together with the operational assumptions under which it was generated.

Finally, the architecture becomes considerably easier to validate. Since the Causal Core performs only one organisational responsibility, its correctness can be evaluated independently of behavioural quality, normative reasoning or execution performance.

The architecture therefore replaces implicit operational assumptions with an explicit organisational component dedicated exclusively to operational self-regulation.

The internal structure of the Causal Core follows the same design philosophy. Rather than producing a single abstract health indicator, it evaluates a set of independent operational dimensions whose combination characterises the current condition of the system. Different implementations may choose different dimensions according to their operational domain. A scientific assistant may emphasise database accessibility and document consistency, whereas a mobile robot may place greater importance on energy reserves, sensor reliability and actuator availability. The architecture deliberately permits this flexibility because operational viability is inherently domain dependent.

The evaluation process nevertheless follows the same organisational sequence in every implementation. The generic metrics used in the reference implementation — disk availability, database integrity, and observation count — are sufficient for demonstration purposes only. A production deployment must develop a formally justified viability model tied to the chosen operational domain and calibrated against actual operational experience. Current observations describing the operational condition of the system enter the Causal Core through explicit interfaces. These observations are analysed according to the viability model defined for the particular implementation. The resulting assessment is encapsulated within a structured representation that becomes available to the remaining architectural components during the current operational cycle.

The Causal Core therefore never stores behavioural knowledge, never performs historical learning and never accumulates experience independently. Its task begins with operational observation and ends with operational evaluation.

This strict limitation is intentional.

As software systems evolve, architectural components tend to accumulate additional responsibilities because extending an existing module often appears easier than introducing a new one. Over time, such incremental extensions produce components whose internal responsibilities become increasingly difficult to distinguish. Artificial Local Intelligence deliberately resists this tendency. Every component remains organisationally narrow even if this requires additional communication between components.

The Causal Core therefore illustrates the central design philosophy of the entire architecture.

Rather than asking how many responsibilities one component can perform, the architecture asks how few responsibilities each component should possess while still contributing to coherent autonomous behaviour.

Once operational viability has been established, the architecture proceeds to the next organisational responsibility.

The system must decide not whether it can continue operating, but what it should do next.

This responsibility belongs to the Ego.

Chapter 5

The Ego

The Causal Core determines whether the system remains operationally capable of acting. It does not determine what the system should do. Once operational viability has been established, a second organisational responsibility begins. The system must interpret the current situation, formulate possible courses of action and propose behaviour that advances its operational objectives.

This responsibility belongs to the Ego.

The term *Ego* is used intentionally. It does not refer to a psychological model of human cognition, nor does it imply that the architecture attempts to imitate the human mind. Within Artificial Local Intelligence, the Ego designates the architectural component responsible for behavioural generation. It is the component that transforms the current situation into candidate actions while remaining entirely independent of their later evaluation.

This distinction is essential.

Many autonomous systems combine behavioural generation and behavioural evaluation within the same reasoning process. A planner generates alternatives while simultaneously estimating their desirability, rejecting unsuitable options and selecting the most promising one. Although efficient, this organisation makes it increasingly difficult to distinguish between creative generation and normative judgement. As behavioural complexity grows, both responsibilities become intertwined.

Artificial Local Intelligence deliberately separates them.

The Ego is free to generate behavioural proposals without deciding whether they should ultimately be executed. It therefore operates under considerably fewer constraints than a conventional decision engine. Its objective is not to produce the final decision. Its objective is to construct the best possible behavioural alternatives from the information currently available.

To accomplish this task, the Ego receives four categories of information.

The first consists of observations describing the current external situation. These observations originate from the Environment Interface and represent the operational context within which behaviour must be generated.

The second consists of the current viability assessment supplied by the Causal Core. Unlike conventional planning systems that implicitly assume satisfactory operating conditions, the Ego explicitly incorporates the operational state of the system into every

planning process. Behaviour is therefore generated with full awareness of current limitations and opportunities.

The third category consists of historical experience retrieved from Memory. Previous operational episodes provide information about successful and unsuccessful behaviour under comparable conditions. The Ego does not merely react to the present. It plans against the background of accumulated operational history.

The fourth category consists of the current operational objectives. Objectives define the direction of behaviour without prescribing the actions required to achieve them. They answer the question *what should be achieved*, whereas the Ego answers the question *how this might be accomplished*.

Using these inputs, the Ego constructs one or more behavioural proposals.

A proposal is neither an action nor a command. It is a structured hypothesis describing how the system could respond to the current situation. Several proposals may coexist simultaneously, each representing a different strategy, different tool selection or different sequence of operations. At this stage the architecture deliberately avoids selecting a single preferred alternative.

The existence of multiple proposals reflects an important design decision.

Behavioural generation should remain exploratory rather than prematurely restrictive. Limiting planning too early often prevents potentially superior solutions from emerging. By postponing evaluation until the normative architecture becomes involved, the Ego remains free to explore a broader behavioural space.

The computational methods employed during this process are intentionally left open.

A minimal implementation might use deterministic planning algorithms. A more sophisticated implementation may integrate symbolic reasoning, probabilistic search, constraint solving or large language models. Future systems may employ reasoning technologies that do not yet exist.

None of these implementation choices alter the architectural role of the Ego.

The architecture defines responsibilities rather than algorithms.

This distinction ensures that Artificial Local Intelligence remains technologically independent. Improvements in reasoning technology affect only the implementation of behavioural generation. They do not require modifications to the surrounding architecture because the organisational responsibility of the Ego remains unchanged.

Large language models provide an instructive example.

Within many contemporary agent systems, the language model effectively becomes the centre of the entire architecture. It interprets observations, plans behaviour, selects tools, reflects upon intermediate results and frequently participates in evaluating its own proposals.

Within ALI the same language model occupies a much narrower organisational role.

It serves as one possible reasoning engine inside the Ego.

The architecture determines when it is invoked, which information it receives and what form its output must take. Once behavioural proposals have been generated, the language model has completed its organisational responsibility. The subsequent normative evaluation occurs independently.

This inversion of responsibility represents one of the defining characteristics of the architecture. Intelligence is no longer identified with one computational model. Instead, intelligence emerges from the coordinated interaction of specialised architectural components, each performing one clearly defined function.

Behavioural generation therefore becomes an engineering discipline in its own right.

The quality of the Ego is not measured solely by the sophistication of its reasoning algorithms. It is equally determined by the diversity of behavioural alternatives it can construct, the effective use of historical experience, the appropriate consideration of operational viability and the clarity with which its proposals can be evaluated by the remaining architecture.

Importantly, the Ego never executes its own proposals.

Execution belongs to the Runtime.

Nor does the Ego determine whether a proposal satisfies organisational rules, safety requirements or long-term behavioural policies.

That responsibility belongs exclusively to the Super-Ego.

The Ego therefore concludes its work by submitting one or more behavioural proposals to the next architectural component.

Only there does the architecture address the question that has deliberately remained unanswered until now.

Not *what can be done*, but *what should be permitted*.

Chapter 6

The Super-Ego

The Ego generates behavioural proposals, but it deliberately refrains from judging them. At this point the architecture has produced one or more possible courses of action, yet no decision has been made regarding their acceptability. Behavioural generation and behavioural execution remain separated by an essential organisational step. Every proposal must first undergo normative evaluation.

This responsibility belongs to the Super-Ego.

As with the Ego, the term is used as an architectural designation rather than a psychological metaphor. The Super-Ego does not represent conscience, morality or human ethics. It denotes the organisational component responsible for evaluating behavioural proposals according to the rules, constraints and operational principles that govern the system. Its task is not to invent behaviour but to determine whether behaviour generated elsewhere should be permitted.

The distinction between generation and evaluation is one of the central principles of Artificial Local Intelligence.

In many contemporary AI systems, planning and evaluation occur within the same computational process. A planner generates an action while simultaneously estimating its usefulness, legality, safety or expected success. Although efficient, this organisation obscures an important engineering distinction. The same component that creates behaviour also justifies it.

ALI deliberately avoids this arrangement.

The Super-Ego never generates behavioural alternatives. Consequently, it has no interest in defending or preferring a particular proposal. Every proposal is evaluated according to the same normative framework, regardless of the computational process that produced it.

This separation substantially increases transparency.

If a proposal is rejected, the reason for rejection can be reconstructed independently of the planning process. Likewise, improvements in behavioural planning never require modifications of the normative architecture, and changes in normative policies do not affect the internal organisation of the Ego.

Normative evaluation proceeds in several stages.

The first stage examines permanent architectural constraints. These constraints define behaviour that is never permissible, regardless of operational objectives or expected benefits. Examples include actions outside the authorised operational domain, irreversible

destruction of historical records, unauthorised modification of the architecture itself or attempts to bypass mandatory evaluation procedures. Such constraints constitute the stable foundation upon which all further normative reasoning is built.

If a proposal satisfies these permanent constraints, evaluation continues with adaptive norms.

Unlike permanent constraints, adaptive norms express behavioural preferences rather than absolute prohibitions. They encourage behaviour that has proven advantageous under previous operational conditions while allowing exceptions when circumstances require them. Examples include preferring reversible actions over irreversible ones, minimising unnecessary resource consumption, favouring reliable information sources or selecting strategies with lower operational uncertainty.

Adaptive norms therefore regulate behaviour without determining it completely.

Their relative importance may change over time as the system accumulates operational experience. A norm that initially possesses only minor significance may become increasingly important if repeated operational history demonstrates its long-term value. Conversely, norms that prove less useful under changing environmental conditions may gradually decrease in priority without disappearing entirely.

This capacity for adaptation distinguishes normative development from fixed rule-based systems.

Nevertheless, adaptation remains carefully constrained.

The Super-Ego never modifies permanent architectural constraints. These define the organisational identity of the architecture itself. Only adaptive norms may evolve, and even these changes remain subject to higher-level regulation described later in this book.

An important consequence follows from this organisation.

Normative evaluation becomes historically informed without becoming historically determined.

Operational experience influences future evaluations, but no individual historical event is capable of redefining the normative structure by itself. Adaptation emerges gradually through accumulated evidence rather than isolated exceptions.

The output of the Super-Ego is not merely an approval or rejection.

Normative evaluation may produce several different outcomes.

A proposal may be approved without modification.

It may be rejected because it violates one or more permanent constraints.

It may require modification before execution becomes permissible.

Or the Super-Ego may request additional behavioural alternatives if none of the existing proposals satisfy the current normative requirements.

This final possibility illustrates an important architectural property.

Normative rejection does not terminate behavioural planning.

Instead, it initiates a new interaction between the Super-Ego and the Ego. Behavioural generation and normative evaluation therefore form an iterative process that continues until an acceptable proposal has been identified or until no satisfactory solution exists under the current operational conditions.

The Super-Ego itself remains entirely independent of execution.

Once a proposal has been approved, its responsibility ends.

It neither performs the action nor supervises its technical implementation. These responsibilities belong to other architectural components. The Super-Ego is concerned exclusively with normative legitimacy.

This limitation is deliberate.

As autonomous systems become more complex, there is a natural tendency to allow normative reasoning gradually to absorb behavioural planning or operational control. Artificial Local Intelligence resists this tendency by assigning each organisational responsibility to exactly one architectural component.

The Super-Ego therefore never becomes a planner.

It never becomes a scheduler.

It never becomes a monitoring system.

Its function remains singular and explicit throughout the lifetime of the architecture.

This organisational clarity offers practical advantages beyond transparency.

Normative policies may be revised independently of planning algorithms.

Different operational domains may employ different normative architectures while retaining the same behavioural mechanisms.

Formal verification becomes considerably easier because normative correctness can be analysed without simultaneously evaluating behavioural quality.

Most importantly, autonomous behaviour remains permanently controllable because every executed action has passed through an explicit and reconstructable normative evaluation process.

The Super-Ego therefore represents far more than a safety mechanism.

It is the organisational component that transforms behavioural possibilities into behaviour that the architecture is prepared to accept.

Only after this transformation has taken place can the system proceed to execution.

Before execution itself, however, one further architectural requirement must be satisfied.

Every autonomous decision must become part of the permanent developmental history of the system.

This responsibility belongs to Memory.

Chapter 7

Memory

The previous chapters introduced the three components responsible for operational viability, behavioural generation and normative evaluation. Together they determine how an ALI interprets the present and decides upon future behaviour. Yet autonomy requires more than responding intelligently to the current situation. Every autonomous system develops through experience, and experience presupposes continuity.

Without continuity every operational cycle would begin from the same initial conditions. Behaviour could become increasingly sophisticated within a single interaction, but nothing would persist beyond its completion. The system would possess information but no history, observations but no experience, actions but no development.

Artificial Local Intelligence therefore assigns historical continuity to an independent architectural component.

This component is Memory.

At first glance, Memory may appear similar to the memory systems employed by many contemporary AI applications. Most autonomous agents maintain conversation histories, vector databases, document repositories or collections of previous interactions. These resources undoubtedly improve behavioural performance by providing additional information during future reasoning.

The Memory architecture of ALI pursues a different objective.

Its primary purpose is not information retrieval.

Its primary purpose is the preservation of the complete developmental history of the system.

This distinction is fundamental.

Information describes isolated facts.

History describes how those facts emerged through successive operational decisions.

Artificial Local Intelligence is interested in the second.

Every completed operational cycle produces one or more Events. In this specification, an Event corresponds to one complete proposal evaluation: a single Observation may generate several proposals, each producing a separate Event. A Cycle is one observation-to-learning Runtime iteration; an Event is one complete proposal evaluation and execution record. Future implementations may introduce a parent Cycle object grouping all Events for one Observation.

An Event represents one complete episode of autonomous operation. It contains the observation that initiated the cycle, the operational viability determined by the Causal Core, the behavioural proposals generated by the Ego, the normative evaluation performed by the Super-Ego, the action that was eventually executed and the observable consequences of that execution.

Nothing essential is omitted.

Consequently, every autonomous decision remains permanently reconstructable.

This organisational principle distinguishes Memory from conventional logging systems.

Log files generally record technical information intended for debugging or system administration. They describe what happened from the perspective of software execution.

Memory records what happened from the perspective of architectural development.

It preserves the complete decision process rather than merely its technical implementation.

This difference has profound consequences.

Because every operational episode is preserved in its entirety, the architecture can later reconstruct not only what the system did but also why it did so. It becomes possible to determine which operational assumptions influenced a decision, which behavioural alternatives were considered, which normative constraints affected the outcome and how the resulting experience contributed to future development.

Historical continuity therefore becomes an architectural capability rather than a collection of stored data.

Memory performs no behavioural reasoning of its own.

It never generates proposals.

It never evaluates norms.

It never estimates operational viability.

Its organisational responsibility begins when an operational cycle has been completed and ends when the corresponding Event has been stored.

Likewise, retrieval from Memory remains deliberately passive. Memory does not perform behavioural or normative interpretation, but it may perform storage-oriented operations — serialisation, indexing, validation, and reconstruction of immutable Event objects — as these are storage responsibilities, not autonomous interpretation.

Memory does not decide which historical information should influence future behaviour.

Instead, other architectural components formulate explicit requests describing the information they require. The retrieved history then becomes available to the requesting component without Memory participating in its interpretation.

This separation once again reflects the central design philosophy of ALI.

Storage and interpretation remain distinct organisational responsibilities.

As operational history accumulates, Memory gradually acquires a second function beyond simple preservation.

It becomes the foundation of experience.

A single Event represents only one completed episode. By itself it provides little guidance for future behaviour. When thousands of Events have accumulated, however, recurring regularities begin to emerge. Certain behavioural strategies repeatedly succeed under comparable operational conditions. Particular environmental situations consistently precede specific failures. Some normative adaptations prove increasingly effective, whereas others gradually lose relevance.

Experience therefore does not exist independently of Memory.

It emerges from the structured interpretation of historical continuity.

Importantly, this interpretation is performed by the architectural components responsible for behavioural planning and normative adaptation, not by Memory itself.

Memory remains historically complete and semantically neutral.

This neutrality provides another important engineering advantage.

Different implementations may analyse the same historical record in different ways without modifying the stored Events. Improved planning algorithms, alternative learning procedures or more sophisticated statistical methods can therefore be introduced without requiring changes to the historical record itself. The architecture preserves the past while permitting future analytical techniques to evolve.

Memory also establishes the developmental identity of an ALI.

Two systems may begin with identical software, identical configuration files and identical architectural components. From the moment they enter operation, however, their histories diverge. They encounter different environments, accumulate different experiences, adapt different normative priorities and develop different behavioural preferences.

Their architecture remains identical.

Their history becomes unique.

Identity therefore arises not from installation but from accumulated operational development.

This distinction between architecture and history is one of the defining characteristics of Artificial Local Intelligence.

The architecture specifies what an ALI is capable of becoming.

Memory records what that particular ALI has actually become.

For this reason, historical information is never discarded merely because it has become operationally irrelevant. Obsolete strategies, rejected behavioural proposals, abandoned norms and unsuccessful experiments remain valuable components of the developmental history. They document not only the current state of the system but the path through which that state was reached.

Consequently, Memory is not organised around forgetting.

It is organised around preserving the complete developmental trajectory of the architecture.

Only through this continuity can learning remain explainable, normative adaptation remain accountable and autonomous behaviour remain permanently reconstructable.

With operational viability established by the Causal Core, behaviour generated by the Ego, normative evaluation performed by the Super-Ego and historical continuity preserved by Memory, the architecture now possesses all informational components required for autonomous operation.

One responsibility nevertheless remains.

These components must interact in a precise and deterministic sequence.

Their cooperation requires an organisational coordinator that possesses no behavioural intelligence of its own yet guarantees that every operational cycle proceeds correctly.

This responsibility belongs to the Runtime.

Chapter 8

The Runtime

The preceding chapters introduced the components that define the cognitive organisation of Artificial Local Intelligence. The Causal Core evaluates operational viability, the Ego generates behavioural proposals, the Super-Ego performs normative evaluation and Memory preserves the complete developmental history of the system. Together these components contain all information required for autonomous operation.

They do not, however, determine when or in which sequence their respective responsibilities are executed.

Without an explicit coordination mechanism, the architecture would consist of independent components lacking a common operational structure. Behavioural proposals might be generated before operational viability had been assessed. Normative evaluation could occur before planning had been completed. Historical records might be stored before execution had actually taken place. Although each individual component would function correctly, the architecture as a whole would no longer possess a deterministic operational process.

Artificial Local Intelligence therefore introduces a fifth organisational component.

This component is the Runtime.

The Runtime does not contribute intelligence to the system.

It contributes order.

Its responsibility is to coordinate the interaction of all remaining architectural components while deliberately avoiding participation in their internal decision making. It neither evaluates operational viability, nor generates behaviour, nor performs normative reasoning, nor analyses historical experience. It exists solely to organise the execution of the architecture.

This distinction is fundamental.

Many contemporary AI systems blur the boundary between orchestration and reasoning. The same process that invokes external tools often participates in planning, monitors execution, reacts to failures and determines subsequent behaviour. As systems become more complex, orchestration gradually evolves into another decision-making component.

Artificial Local Intelligence deliberately prevents this development.

The Runtime remains organisationally neutral.

Its operation follows a deterministic sequence that is identical for every operational cycle.

Each cycle begins by obtaining the current observation from the Environment Interface. This observation represents the present operational situation and remains immutable throughout the cycle. Once acquired, it is forwarded to the Causal Core, which evaluates the current operational viability of the system. The resulting viability assessment is then transmitted to the Ego together with the current observation, operational objectives and relevant historical information retrieved from Memory.

The Ego constructs one or more behavioural proposals.

These proposals are transferred to the Super-Ego, where they undergo normative evaluation. Depending upon the outcome, one proposal may be approved, modified or rejected entirely. If no acceptable proposal exists, the Runtime requests additional behavioural alternatives from the Ego. The architecture therefore supports iterative interaction between behavioural generation and normative evaluation while preserving their organisational independence.

Only after explicit approval has been obtained does execution become possible.

The Runtime transfers the approved proposal to the Environment Interface, which performs the corresponding external action. During execution the Runtime supervises technical progress without participating in behavioural decisions. It records execution status, detects failures and coordinates recovery procedures whenever necessary.

When execution has been completed, the Runtime assembles the complete operational episode. The Environment Interface creates the technical ExecutionResult; the Runtime incorporates it into the Event. Assigning sole object ownership of ExecutionResult to the Runtime is therefore too strict for practical implementations: the Runtime coordinates execution and assembles the Event, while the Environment Interface may construct the result object itself.

Observation, operational viability, behavioural proposal, normative evaluation, execution result and resulting system state are combined into a single Event. This Event is then transferred to Memory, where it becomes part of the permanent developmental history of the system.

Only after historical preservation has been completed does the Runtime initiate the next operational cycle.

This sequence is deliberately invariant.

Every autonomous decision follows exactly the same organisational structure regardless of the operational domain, behavioural complexity or computational methods employed within the individual architectural components.

Such determinism offers important engineering advantages.

First, the architecture becomes entirely reconstructable. Since every operational cycle follows the same sequence, historical analysis can determine precisely where and why a particular decision emerged.

Second, verification becomes considerably simpler. Individual components may be tested independently because their execution order is guaranteed by the Runtime rather than embedded within their own implementations.

Third, implementation technologies become interchangeable. Improvements in behavioural planning, normative reasoning or operational evaluation require no modification of the Runtime because its organisational responsibility remains constant.

The Runtime also supervises exceptional situations.

Autonomous systems inevitably encounter unexpected events. Communication failures, unavailable tools, corrupted observations, insufficient resources or interrupted executions cannot be treated as extraordinary occurrences. They are normal aspects of long-term autonomous operation.

The Runtime therefore provides a uniform organisational framework for handling such situations.

Whenever recoverable failures occur, the Runtime coordinates appropriate recovery procedures while preserving the historical integrity of the operational cycle. If recovery proves impossible, the Runtime initiates controlled shutdown procedures that preserve the current architectural state before terminating execution. Even failures therefore become part of the permanent operational history rather than disappearing as undocumented exceptions.

Importantly, these supervisory responsibilities do not transform the Runtime into an intelligent controller.

The Runtime never decides *which* behaviour should be executed.

It decides only *when* the approved behaviour should be executed.

Likewise, it never determines whether operational conditions remain satisfactory or whether normative constraints have been satisfied. These responsibilities belong exclusively to the Causal Core and the Super-Ego.

The Runtime simply ensures that every architectural component performs its organisational role at the appropriate moment.

This disciplined limitation of responsibility illustrates a recurring principle throughout the ALI architecture.

Every component performs exactly one organisational function.

Whenever additional functionality appears desirable, the architecture first asks whether that functionality represents a genuinely new organisational responsibility. If the answer is yes, a new component should be introduced rather than extending an existing one. This principle prevents the gradual accumulation of unrelated responsibilities that characterises many large software systems.

With the Runtime, the complete operational organisation of Artificial Local Intelligence is now established.

The architecture possesses an explicit representation of operational viability, an independent behavioural generator, an autonomous normative evaluator, a permanent historical memory and a deterministic execution coordinator. Together these components define the organisational foundation upon which all subsequent mechanisms, including learning, normative development and long-term autonomy, are built.

The following chapter examines how these components interact continuously to produce a complete operational cycle and how repeated execution of this cycle gives rise to the developmental behaviour of an Artificial Local Intelligence.

Chapter 9

The Operational Cycle

The previous chapters introduced the five architectural components that define Artificial Local Intelligence. Considered individually, each component performs one clearly defined organisational responsibility. Taken together, however, they form a continuously operating autonomous system. The purpose of this chapter is to describe how these components cooperate during normal operation.

The operational cycle is the fundamental dynamic process of the architecture. Every behavioural decision, every learning event and every modification of the historical record originates from one complete operational cycle. The repeated execution of this cycle transforms a static software installation into a continuously developing autonomous system.

Unlike conventional software applications, an ALI does not operate by responding to isolated requests. Instead, it exists as a continuously active system whose internal state evolves through uninterrupted interaction with its environment. Every completed cycle influences the next one through changes in operational history, behavioural experience and normative development. The architecture therefore exhibits temporal continuity rather than a sequence of independent computations.

Each operational cycle begins with observation.

The Environment Interface collects information describing the current operational situation and transforms it into a structured Observation. This observation represents the external world as it exists at the beginning of the cycle. Once created, it remains unchanged until the cycle has been completed. Subsequent environmental changes become part of the following cycle rather than modifying the current one. This immutability guarantees that every decision can later be reconstructed under exactly the conditions that originally existed.

The Observation is then transferred to the Causal Core.

Using its viability model, the Causal Core evaluates the current operational condition of the system. This evaluation does not concern the external objective but the system itself. It determines whether sufficient resources remain available, whether communication channels function correctly, whether computational integrity has been preserved and whether continued autonomous operation remains advisable under the present circumstances.

The result is a structured Viability State describing the current operational condition of the architecture.

This information becomes immediately available to the Ego.

The Ego now possesses four sources of information: the current Observation, the Viability State supplied by the Causal Core, the active operational objectives and the relevant historical experience retrieved from Memory. Using these inputs, it generates one or more behavioural proposals representing different possible responses to the current situation.

At this stage no behavioural commitment has yet been made.

The proposals merely describe what the system could do.

Each proposal is subsequently transferred to the Super-Ego.

The Super-Ego evaluates every proposal according to the current normative architecture. Permanent constraints are examined first, followed by adaptive norms and, where appropriate, generated norms. The outcome of this evaluation may be approval, rejection, modification or a request for additional behavioural alternatives.

If all proposals are rejected, the operational cycle does not terminate.

Instead, the Runtime requests further behavioural proposals from the Ego, which continues planning under the same observational conditions until an acceptable proposal has been identified or no satisfactory solution exists.

Once a proposal has received explicit normative approval, execution may begin.

The Runtime transfers the approved proposal to the Environment Interface, which performs the corresponding external action. During execution the Runtime supervises technical progress, monitors failures and coordinates recovery procedures whenever necessary. It deliberately refrains from influencing behavioural content or normative judgement.

Execution concludes when the external action has either been completed successfully or terminated under controlled failure conditions.

At this point the architecture possesses all information required to construct a complete historical record.

The Runtime assembles the Observation, the Viability State, the approved behavioural proposal, the normative evaluation, the execution result and the resulting operational condition into a single Event. This Event represents one complete episode of autonomous behaviour.

The Event is then stored permanently in Memory.

Historical preservation marks the transition from immediate behaviour to accumulated experience. The current operational cycle has now become part of the developmental

history of the system and may influence future behavioural planning, normative adaptation and operational evaluation.

Only after this historical update has been completed does learning begin.

Behavioural learning occurs within the Ego. Previously successful and unsuccessful strategies become part of future planning. Normative adaptation occurs within the Super-Ego. Adaptive priorities may gradually change, candidate norms may be evaluated and long-term behavioural regularities may influence future normative development. The Causal Core updates operational statistics required for increasingly accurate viability estimation.

Each component performs learning independently.

The Runtime merely coordinates the sequence in which these processes occur.

When all learning activities have been completed, the operational cycle ends.

The Runtime returns the architecture to its waiting state, where it remains until new observations become available or another event initiates the next cycle. The architecture therefore alternates continuously between observation, evaluation, planning, execution, historical preservation and learning.

Although this sequence appears simple, it embodies one of the central ideas of Artificial Local Intelligence.

No architectural component ever performs more than one organisational responsibility.

Operational viability remains independent of behavioural generation.

Behavioural generation remains independent of normative evaluation.

Normative evaluation remains independent of execution.

Execution remains independent of historical preservation.

Historical preservation remains independent of learning.

Learning remains independent of Runtime coordination.

This disciplined separation ensures that every decision made by the architecture can be traced to its organisational origin.

The operational cycle therefore serves two purposes simultaneously.

It coordinates the immediate production of autonomous behaviour.

At the same time, it preserves the complete developmental history from which future behaviour will emerge.

Every repetition of the cycle makes the system slightly different from what it was before. New experiences accumulate, behavioural strategies evolve, normative priorities adapt and operational models become increasingly refined.

The architecture itself, however, remains unchanged.

Its organisation is stable.

Its history is not.

This distinction between stable architecture and continuously developing operational history forms the basis for all subsequent chapters on learning, normative evolution and the long-term development of Artificial Local Intelligence.

Chapter 10

Learning

At first sight, learning appears to be one further function among many others. Contemporary artificial intelligence often treats learning as an optimisation procedure that improves behaviour through experience. Once sufficient data have been accumulated, the learning algorithm modifies the internal model, and the resulting behaviour becomes increasingly successful.

Artificial Local Intelligence adopts a different perspective.

Learning is not regarded as a single computational process.

It is the consequence of several independent architectural components modifying different aspects of the system while preserving their organisational responsibilities.

This distinction follows directly from the architecture developed in the preceding chapters.

The Causal Core evaluates operational viability.

The Ego generates behaviour.

The Super-Ego evaluates behaviour.

Memory preserves operational history.

The Runtime coordinates execution.

If each of these components possesses a different organisational responsibility, learning cannot consist of one global optimisation process without destroying the architectural separation that defines the system.

Instead, every component develops independently according to its own objective.

Behavioural learning takes place within the Ego.

Its purpose is to improve the quality of future behavioural proposals. Every completed operational cycle provides new information concerning successful strategies, unsuccessful actions, reliable tools, changing environmental conditions and recurring operational situations. This information gradually changes how future behavioural alternatives are generated.

Importantly, the Ego does not learn by memorising individual actions.

It learns by recognising regularities across many operational episodes.

An isolated success may result from favourable circumstances rather than from an effective behavioural strategy. Only repeated operational experience provides sufficient evidence for stable behavioural adaptation.

Consequently, behavioural learning is inherently historical.

The longer the operational history becomes, the more accurately the Ego can distinguish between accidental success and genuinely reliable behaviour.

Operational learning takes place within the Causal Core.

Although the organisational responsibility of the Causal Core remains unchanged, its operational model becomes progressively more accurate through accumulated experience. Relationships between resource availability, communication quality, computational load and successful long-term operation become increasingly well understood. The viability assessment therefore becomes more precise without altering the organisational function of the component itself.

Normative learning takes place within the Super-Ego.

Every normative evaluation contributes additional evidence concerning the effectiveness of the current normative architecture. Some adaptive norms repeatedly support successful operation, whereas others prove less relevant under changing environmental conditions. Candidate norms may emerge when existing regulations repeatedly fail to resolve recurring situations. Conversely, norms that no longer contribute to stable operation may gradually lose importance.

Normative learning therefore concerns the development of behavioural regulation rather than behavioural generation.

Historical learning occurs within Memory.

Strictly speaking, Memory does not learn in the same sense as the remaining architectural components because it never interprets the information it stores. Nevertheless, every completed operational cycle enriches the developmental history upon which all subsequent learning depends. Historical continuity therefore increases continuously throughout the lifetime of the system.

The Runtime does not participate in learning.

Its organisational responsibility remains unchanged throughout the operational lifetime of the architecture. It coordinates the learning activities performed by the remaining components but never modifies its own execution logic in response to operational experience.

This asymmetry is deliberate.

Learning should occur only where it contributes directly to the organisational responsibility of a component. Extending learning indiscriminately throughout the architecture would

gradually blur the distinction between coordination, behavioural generation, operational evaluation and normative reasoning.

Artificial Local Intelligence therefore distinguishes carefully between **adaptation** and **architecture**.

Adaptation changes the information processed by a component.

Architecture defines the responsibility of the component itself.

Only the first may change during operation.

The second remains permanently stable.

This distinction has an important consequence.

An ALI may operate continuously for many years, accumulate millions of operational episodes and exhibit behavioural characteristics that differ profoundly from those present immediately after installation.

Nevertheless, its architecture remains exactly the same.

No new organisational responsibilities emerge.

No existing responsibilities disappear.

Learning changes how the architecture behaves.

It never changes what the architecture is.

The architecture therefore possesses two complementary forms of stability.

Its organisational structure remains invariant.

Its operational behaviour remains continuously adaptive.

This combination of stability and development constitutes one of the defining characteristics of Artificial Local Intelligence.

It permits continuous behavioural improvement without sacrificing transparency, reconstructability or architectural clarity.

The following chapter examines this developmental process in greater detail by analysing how autonomous behaviour gradually emerges from repeated operational cycles and how operational history transforms an initially generic architecture into an individual Artificial Local Intelligence.

Chapter 11

Development

The architecture described in the preceding chapters defines the organisational structure of Artificial Local Intelligence. At the moment of installation, however, this architecture exists only as a potential. The components are present, the interfaces are defined and the operational cycle can begin, yet the system possesses no experience, no behavioural preferences and no individual history.

An installed ALI is therefore not yet an autonomous individual.

It is an architecture waiting to develop.

This distinction between architecture and development is one of the central ideas of the ALI framework. Most descriptions of artificial intelligence implicitly assume that intelligence exists immediately after deployment and subsequently improves through learning. Artificial Local Intelligence adopts a different view. The architecture exists from the beginning, but autonomous individuality emerges only through continuous interaction with the operational environment.

Development therefore begins with the first operational cycle.

Initially, every behavioural proposal is generated from predefined planning capabilities. Operational viability is evaluated using an initial viability model. Normative decisions rely exclusively on the permanent constraints and adaptive norms supplied by the system designer. Memory contains no operational history because no decisions have yet been made.

The first completed operational cycle changes this situation fundamentally.

For the first time, the system possesses a complete Event describing an autonomous interaction with its environment. Although a single Event represents only one operational episode, it establishes the beginning of an irreversible developmental process. Every subsequent cycle adds another Event to the historical record. The architecture remains unchanged, but the system itself begins to acquire a past.

This distinction deserves particular attention.

Artificial Local Intelligence separates **architecture** from **identity**.

Architecture defines the organisational structure shared by every implementation.

Identity arises from the unique historical development of an individual system.

Two ALIs installed from the same software package therefore begin with identical architectures. The moment they encounter different operational environments, however,

their developmental histories diverge. They accumulate different experiences, encounter different problems, adapt different behavioural strategies and gradually establish different normative preferences.

From that moment onward they remain architecturally identical but developmentally distinct.

Behavioural development proceeds gradually.

During the early stages of operation, planning depends primarily upon predefined reasoning capabilities. Historical information plays only a minor role because little operational experience exists. As the number of completed Events increases, behavioural regularities become visible. Certain planning strategies repeatedly produce successful outcomes under comparable conditions, whereas others consistently fail. The Ego gradually incorporates these regularities into future planning.

Behaviour therefore becomes increasingly historical.

The same developmental principle applies to the Causal Core.

Initially, operational viability is estimated from predefined operational models. As long-term experience accumulates, the relationship between operational conditions and successful autonomous behaviour becomes progressively clearer. The Causal Core refines its viability estimates without altering its organisational responsibility. Operational judgement becomes increasingly accurate because it is informed by experience rather than assumptions alone.

Normative development follows a similar trajectory.

At the beginning of operation, the Super-Ego possesses only the normative architecture provided during installation. Adaptive norms have not yet acquired operational significance because insufficient experience exists to evaluate their effectiveness. As operational history grows, the relative importance of different norms gradually changes. New candidate norms may emerge from recurring situations that were not anticipated during system design. Existing adaptive norms may become increasingly refined through repeated application.

Normative development therefore reflects the accumulated experience of the system rather than modifications imposed externally.

Memory develops differently from the remaining architectural components.

Unlike the Ego, the Super-Ego or the Causal Core, Memory does not analyse its own contents. Its development consists solely in the continuous accumulation of historical continuity. Nevertheless, this apparently passive process has profound consequences. The

larger the historical record becomes, the richer the foundation upon which behavioural and normative development can proceed.

The Runtime does not develop.

This observation often surprises readers because the Runtime participates in every operational cycle. Yet its organisational responsibility is coordination rather than adaptation. Every operational cycle follows the same deterministic sequence regardless of how sophisticated the remaining architectural components become. The Runtime therefore provides the stable temporal framework within which all developmental processes occur.

Over extended periods of operation, these independent developmental processes begin to interact.

Improved behavioural planning changes the kinds of proposals submitted to the Super-Ego.

Refined normative evaluation influences which behaviours become part of the historical record.

Increasingly accurate viability assessment affects future planning decisions.

The growing historical record provides richer evidence for subsequent behavioural and normative adaptation.

Development therefore emerges from the continuous interaction of independently evolving architectural components rather than from a single global learning algorithm.

Eventually, every ALI reaches a stage at which its operational behaviour reflects years of accumulated experience.

Its planning strategies have become highly specialised to its operational domain.

Its normative architecture has adapted to recurring situations.

Its viability model accurately reflects the operational characteristics of its environment.

Its historical record documents every significant stage of its development.

Yet despite these profound behavioural changes, the architecture itself remains exactly as it was on the day of installation.

This distinction between organisational stability and behavioural development constitutes one of the defining characteristics of Artificial Local Intelligence.

The architecture provides permanence.

Development provides individuality.

Autonomy emerges not because the architecture changes, but because a stable architecture continuously transforms experience into increasingly coherent behaviour.

The following chapter examines this transformation in greater detail by analysing the mechanisms through which behavioural competence, normative regulation and operational experience gradually mature during long-term autonomous operation.

Chapter 12

Behavioural Maturation

Development is a continuous process, yet continuous processes often exhibit recognisable phases. These phases do not represent discrete architectural states, nor are they separated by clearly defined boundaries. Instead, they describe characteristic patterns that emerge as operational history accumulates and the interaction between the architectural components becomes increasingly sophisticated.

Behavioural maturation should therefore be understood as a developmental continuum rather than a sequence of predefined stages.

Immediately after installation, the behaviour of an ALI is almost entirely determined by its initial planning capabilities. The Ego possesses no operational experience, the Super-Ego has not yet adapted its normative priorities and the Causal Core relies exclusively on its initial viability model. Every behavioural proposal is therefore generated from general principles rather than historical regularities.

At this stage, the architecture is complete.

The system is not.

Its decisions are logically consistent, yet they remain largely independent of operational history because such a history does not yet exist.

As operational cycles accumulate, the first signs of behavioural differentiation appear.

The Ego begins to recognise recurring situations. Certain behavioural strategies repeatedly succeed under similar conditions, whereas others consistently require revision or fail entirely. Behaviour gradually becomes less exploratory and more selective. The system no longer considers every theoretically possible solution equally plausible but increasingly favours strategies supported by accumulated experience.

Importantly, this transition does not reduce behavioural flexibility.

The architecture continues to generate alternative proposals whenever unfamiliar situations arise. Established strategies merely provide an increasingly reliable point of departure for future planning. Exploration therefore never disappears completely. It becomes progressively more focused.

The development of behavioural confidence follows a similar trajectory.

Initially, confidence estimates reflect little more than the internal characteristics of the planning process itself. As operational history expands, confidence becomes increasingly empirical. The system begins to distinguish between proposals that have repeatedly demonstrated reliable performance and those whose success remains uncertain.

Confidence therefore changes its meaning during development.

It evolves from an estimate based primarily on reasoning into an estimate supported by accumulated operational evidence.

At the same time, the interaction between the Ego and the Super-Ego becomes increasingly efficient.

During early operation, behavioural proposals are rejected comparatively often because the planning process has not yet adapted to the normative architecture. Repeated interaction gradually reduces these conflicts. The Ego learns to generate proposals that already satisfy many normative requirements before formal evaluation begins. The Super-Ego, in turn, develops increasingly refined adaptive priorities that distinguish genuinely problematic behaviour from behaviour that merely differs from previous experience.

Behavioural generation and normative evaluation therefore become progressively better aligned while remaining organisationally independent.

The Causal Core also participates in this maturation.

Long-term operational experience enables increasingly accurate viability estimation. Variables that initially appeared unrelated may gradually reveal systematic relationships. Certain patterns of resource utilisation may consistently precede operational instability. Particular environmental conditions may repeatedly reduce behavioural reliability. Over time, these regularities become incorporated into the viability model, allowing the architecture to anticipate operational difficulties before they become critical.

Operational judgement therefore becomes increasingly predictive rather than merely descriptive.

Memory undergoes perhaps the most visible transformation.

What initially consisted of isolated Events gradually develops into a coherent developmental history. Individual operational episodes become embedded within increasingly rich historical contexts. Behaviour can now be interpreted not only with respect to the immediate situation but also in relation to months or years of accumulated operational experience.

This expansion of historical continuity fundamentally changes the character of the architecture.

The system no longer reacts exclusively to the present.

It increasingly acts as the product of its own developmental history.

This process eventually produces behavioural maturity.

Behavioural maturity should not be interpreted as a final stage of development or as the achievement of perfect performance. Autonomous systems continue to encounter novel situations throughout their operational lifetime. Complete adaptation is therefore neither possible nor desirable.

Instead, behavioural maturity describes a condition in which the architecture has achieved a stable balance between experience and flexibility.

Behaviour becomes increasingly reliable because historical regularities are well understood.

Behaviour remains adaptable because unfamiliar situations continue to generate new behavioural alternatives.

Normative evaluation becomes increasingly consistent because adaptive priorities have stabilised through prolonged operational experience.

Operational viability becomes increasingly accurate because the Causal Core has accumulated extensive knowledge concerning the dynamics of the operational environment.

Most importantly, the developmental processes of the individual architectural components become mutually supportive.

Improved behavioural planning produces higher-quality operational experience.

Improved operational experience enables more accurate normative adaptation.

Improved normative adaptation supports increasingly reliable behavioural planning.

The architecture therefore exhibits a positive developmental feedback process without ever collapsing into a single global optimisation mechanism.

Every component continues to perform its own organisational responsibility.

This distinction remains crucial.

Many learning architectures gradually merge behavioural generation, evaluation and adaptation into one increasingly complex optimisation process. Artificial Local Intelligence deliberately preserves their separation throughout the entire developmental lifetime of the system.

Consequently, maturity does not imply architectural convergence.

The Ego does not become a Super-Ego.

The Super-Ego does not become a Causal Core.

The Runtime does not acquire behavioural intelligence.

Every component becomes more competent while remaining organisationally unchanged.

Behavioural maturation therefore illustrates one of the most important properties of Artificial Local Intelligence.

Development occurs **within** the architecture.

It never occurs **to** the architecture.

The organisational structure remains permanently stable.

Only the quality of its operation evolves.

This distinction allows an ALI to continue developing indefinitely while preserving complete transparency, reconstructability and architectural integrity. The following chapter examines how this principle extends beyond individual behaviour to the evolution of the normative system itself.

Chapter 13

The Evolution of the Normative System

Behavioural maturation enables an ALI to solve tasks with increasing competence. Competence alone, however, is insufficient for long-term autonomy. An autonomous system may become highly efficient while simultaneously developing behavioural tendencies that prove undesirable under changing operational conditions. Efficiency and appropriateness are therefore not identical.

For this reason, Artificial Local Intelligence separates behavioural development from normative development.

The Ego learns how behaviour can be improved.

The Super-Ego learns how behaviour should be regulated.

Although both processes influence one another continuously, they pursue fundamentally different objectives.

Behavioural learning seeks increasingly effective action.

Normative development seeks increasingly appropriate action.

This distinction becomes particularly important during prolonged operation.

When an ALI enters a new operational environment, many situations cannot be anticipated by the system designer. Some environmental conditions occur rarely. Others appear only after months of continuous operation. Still others emerge through the interaction between behavioural learning and changing operational objectives.

No predefined normative architecture can completely anticipate such developments.

The architecture must therefore possess the capacity for normative evolution.

Normative evolution begins with adaptive priorities.

Immediately after installation, the Super-Ego contains a set of permanent constraints together with an initial collection of adaptive norms. Permanent constraints define behaviour that is never acceptable under any operational circumstances. Adaptive norms express behavioural preferences whose relative importance may change through experience.

Initially these priorities represent design assumptions.

As operational history accumulates, they become increasingly evidence based.

Suppose that two behavioural strategies repeatedly achieve comparable task performance, yet one consistently consumes fewer computational resources and produces fewer recovery

procedures. The Super-Ego gradually increases the relative importance of the corresponding adaptive norm. Future evaluations therefore become increasingly likely to favour the more robust strategy.

No explicit programming is required.

Normative development emerges from accumulated operational evidence.

Importantly, this adaptation remains gradual.

Individual operational episodes never redefine the normative architecture. Adaptation results only from persistent regularities observed across many operational cycles. The architecture therefore avoids rapid oscillations caused by exceptional situations while remaining capable of long-term development.

Occasionally, however, changing operational conditions reveal limitations that cannot be resolved merely by adjusting existing priorities.

The architecture may repeatedly encounter situations for which no satisfactory normative guidance exists.

Behavioural proposals may consistently satisfy all existing constraints while nevertheless producing undesirable long-term consequences.

Alternatively, the Super-Ego may repeatedly reject otherwise successful behavioural proposals because the existing normative structure cannot adequately distinguish between genuinely problematic and merely unfamiliar behaviour.

Such situations indicate not an error in behavioural planning but an incompleteness of the normative architecture itself.

Artificial Local Intelligence therefore permits the emergence of candidate norms.

A candidate norm represents a provisional extension of the normative architecture. It is not immediately incorporated into ordinary behavioural evaluation. Instead, it exists initially as a hypothesis whose long-term usefulness must be demonstrated through continued operation.

This cautious approach reflects an important architectural principle.

Norms should not emerge because isolated situations appear unusual.

They should emerge because repeated operational history demonstrates the existence of a stable behavioural regularity requiring explicit regulation.

Candidate norms therefore undergo an extended validation process.

Their influence upon behavioural evaluation is initially limited. The architecture observes how frequently comparable situations arise, whether the proposed regulation consistently

improves long-term operation and whether conflicts emerge with existing normative structures.

Only after sufficient historical evidence has accumulated does a candidate norm become part of the ordinary adaptive architecture.

Otherwise it is discarded.

Importantly, discarded candidate norms are not deleted.

Their existence remains permanently documented within Memory together with the evidence that motivated both their introduction and their eventual rejection.

Normative history therefore becomes as reconstructable as behavioural history.

A second developmental process concerns normative refinement.

Even well-established norms rarely remain static throughout prolonged operation. Environmental conditions evolve, operational objectives change and behavioural competence continues to improve. Norms that initially required broad formulation may gradually become increasingly precise as operational understanding deepens.

Refinement therefore differs from generation.

Generation introduces new normative concepts.

Refinement improves the precision of existing ones.

The architecture treats these processes separately because they involve different forms of evidence and different validation procedures.

Throughout all normative development one organisational principle remains absolutely stable.

Permanent constraints never change.

They define the architectural identity of the system rather than its behavioural preferences.

Allowing operational experience to modify these constraints would gradually dissolve the distinction between stable architecture and adaptive behaviour upon which the entire ALI framework depends.

For this reason, every proposed normative modification is evaluated against the permanent architectural principles before it can influence future behaviour.

The Super-Ego therefore develops continuously while remaining architecturally stable. In a production implementation, adaptive norm weights, candidate norms, and the complete normative adaptation history must be persisted between operational sessions. The reference implementation maintains these values in process memory for simplicity; a production deployment requires a dedicated persistent normative state store.

Its content evolves.

Its organisational responsibility does not.

As operational history grows, another interesting phenomenon begins to emerge.

Normative evaluation becomes increasingly anticipatory.

Initially, the Super-Ego merely evaluates completed behavioural proposals. Over time, however, the Ego learns which kinds of proposals are likely to satisfy the normative architecture. Behavioural planning therefore becomes progressively aligned with normative expectations. Many unsuitable alternatives are never generated because historical interaction has already shaped the planning process itself.

The distinction between behavioural generation and normative evaluation nevertheless remains fully intact.

The Ego does not perform normative reasoning.

It has merely learned, through repeated interaction, which behavioural proposals are more likely to satisfy an independently operating Super-Ego.

This mutual adaptation illustrates one of the strengths of the ALI architecture.

Independent architectural components may become increasingly coordinated through experience without sacrificing their organisational independence.

Normative evolution therefore does not represent a secondary learning mechanism added to an existing architecture.

It is an essential consequence of separating behavioural generation from behavioural evaluation.

Only because these responsibilities remain organisationally distinct can each develop according to its own principles while continuously influencing the other.

The result is an architecture capable of preserving stable organisational foundations while allowing its behavioural regulation to evolve throughout years of autonomous operation.

In the following chapter we extend this developmental perspective further by examining how the architecture itself remains permanently stable while supporting continuous adaptation across every operational level.

Chapter 14

Architectural Stability

The previous chapters described how behaviour, operational models and normative structures evolve throughout the lifetime of an Artificial Local Intelligence. At first sight, this continuous adaptation appears to suggest that the architecture itself also changes over time. Such an interpretation would fundamentally misunderstand the design philosophy of ALI.

One of the central principles of the architecture is the strict distinction between **development** and **structure**.

Development concerns the information processed by the architecture.

Structure concerns the organisation through which this information is processed.

Only the first changes during operation.

The second remains invariant.

This distinction is neither accidental nor merely a question of software engineering style. It is the condition that makes long-term autonomy possible. If the organisational responsibilities of the architecture were allowed to evolve continuously together with behaviour, the system would gradually lose its internal transparency. It would become increasingly difficult to determine which component was responsible for a particular decision, why a behavioural strategy had emerged or how operational experience had influenced future development.

Artificial Local Intelligence therefore treats the architecture itself as a permanent organisational framework.

Every architectural component possesses one clearly defined responsibility.

The Causal Core evaluates operational viability.

The Ego generates behavioural proposals.

The Super-Ego evaluates those proposals.

Memory preserves developmental history.

The Runtime coordinates execution.

These responsibilities never migrate between components.

They never overlap.

They never disappear.

The stability of the architecture should not be confused with rigidity.

A stable architecture does not imply static behaviour.

On the contrary, architectural stability enables continuous behavioural adaptation because every developmental process occurs within a predictable organisational framework. Behaviour changes precisely because the architecture itself does not.

This principle can be illustrated by considering the development of an experienced ALI.

After several years of operation, its behavioural strategies may differ profoundly from those present immediately after installation. New planning heuristics have emerged, operational models have become increasingly accurate and normative priorities have adapted to countless recurring situations. Nevertheless, if one examines the architecture itself, nothing essential has changed.

The Causal Core still evaluates viability.

The Ego still generates behaviour.

The Super-Ego still performs normative evaluation.

Memory still preserves history.

The Runtime still coordinates execution.

Only the knowledge processed by these components has changed.

The architecture therefore separates **competence** from **organisation**.

Competence develops.

Organisation persists.

This separation provides several important engineering advantages.

First, the behaviour of the system remains explainable.

Whenever a behavioural change is observed, it can be attributed to changes in operational history, behavioural learning or normative adaptation rather than to uncontrolled architectural evolution.

Second, software maintenance becomes considerably simpler.

Individual algorithms may be improved or replaced without modifying the organisational responsibilities of the architecture. A new reasoning engine may replace an older one within the Ego. A more sophisticated viability model may be introduced into the Causal Core. Alternative storage technologies may replace the existing Memory implementation. None of these changes affect the architecture itself because the responsibilities remain identical.

Third, validation becomes cumulative.

Architectural correctness needs to be established only once.

Subsequent improvements concern the quality of individual implementations rather than the organisational integrity of the architecture. This distinction allows architectural research and algorithmic research to proceed independently while remaining mutually compatible.

The same principle applies to technological progress.

Artificial intelligence is evolving rapidly. New language models, planning algorithms, optimisation methods and computational hardware appear continuously. An architecture that depends upon a particular technology inevitably becomes obsolete as that technology changes.

Artificial Local Intelligence deliberately avoids this dependence.

The architecture specifies organisational responsibilities rather than computational techniques.

Future implementations may employ reasoning systems that differ completely from those available today.

The architecture remains unchanged because it does not define *how* behaviour is generated.

It defines *where* behavioural generation belongs within the organisation of the system.

This technology independence is essential for the long-term relevance of the architecture.

Architectural stability also defines the limits of autonomous adaptation.

An ALI may improve its behaviour indefinitely.

It may refine its viability models.

It may expand its normative architecture.

It may accumulate decades of operational history.

It may become highly specialised to a particular operational domain.

It may not reorganise its own architecture.

The separation between operational viability, behavioural generation, normative evaluation, historical continuity and execution coordination remains permanently fixed.

This restriction is intentional.

Allowing unrestricted architectural self-modification would eventually undermine the very transparency and controllability that the architecture is designed to provide. The purpose of ALI is not to create a system capable of arbitrary organisational evolution. Its purpose is to

create a system capable of unlimited behavioural development within a stable and understandable organisational framework.

In this respect, Artificial Local Intelligence differs fundamentally from approaches that equate increasing autonomy with increasing architectural plasticity.

ALI proposes the opposite.

The more autonomous a system becomes, the more important a stable organisational structure becomes.

Only a stable architecture can provide the continuity required for long-term learning, reliable validation and permanent reconstructability.

Architectural stability should therefore not be regarded as a limitation imposed upon the system.

It is the foundation that makes continuous autonomous development possible.

The following chapter extends this perspective by examining how these stable organisational principles permit Artificial Local Intelligence to remain permanently controllable even as its behaviour becomes increasingly sophisticated through years of autonomous operation.

Chapter 15

Controllability

The architecture developed throughout the preceding chapters pursues two objectives that are often regarded as conflicting. On the one hand, an autonomous system should become increasingly competent through continuous interaction with its environment. On the other hand, it should remain permanently understandable, predictable and controllable. Contemporary discussions of artificial intelligence frequently present these objectives as opposing alternatives. Greater autonomy is assumed to imply reduced controllability, whereas stronger control is expected to limit autonomy.

Artificial Local Intelligence rejects this assumption.

The architecture proposed in this book is based on the opposite principle.

Autonomy and controllability are not mutually exclusive.

They become compatible when they are treated as organisational rather than algorithmic properties.

This distinction deserves careful examination.

Many existing AI systems attempt to achieve controllability by constraining behaviour after it has already been generated. Additional safety layers inspect behavioural outputs, filter undesirable responses or require external approval before execution. Such mechanisms undoubtedly improve practical safety. Nevertheless, they remain external additions to an architecture whose internal organisation has remained unchanged.

Artificial Local Intelligence follows a different approach.

Controllability is not introduced after behavioural generation.

It is built into the architecture itself.

Every organisational responsibility remains explicitly identifiable.

Every behavioural proposal possesses an identifiable origin.

Every normative decision can be reconstructed independently of behavioural planning.

Every executed action becomes part of the permanent historical record.

Every stage of the operational cycle remains observable.

Consequently, understanding the behaviour of the system no longer depends upon analysing one extremely complex reasoning process. Instead, it consists of analysing the interaction between several comparatively simple architectural components.

This organisational transparency produces a second important consequence.

Responsibility becomes local.

If an unexpected behavioural proposal is generated, the architecture immediately identifies the responsible component.

If operational viability has been estimated incorrectly, the source of the error lies within the Causal Core.

If an inappropriate behavioural strategy has been generated, investigation begins within the Ego.

If an unsuitable proposal has nevertheless been approved, attention turns to the Super-Ego.

If historical information has been stored incorrectly, Memory becomes the relevant component.

If execution occurred in an unexpected sequence, the Runtime is responsible.

This localisation of responsibility represents a considerable engineering advantage.

Debugging, verification and long-term maintenance become significantly more systematic because every organisational function possesses an explicit architectural owner.

The same principle applies to learning.

Behavioural adaptation remains observable because every behavioural modification originates within the Ego.

Normative adaptation remains observable because every change occurs within the Super-Ego.

Operational refinement remains observable because only the Causal Core modifies its viability model.

Learning therefore never becomes an anonymous global optimisation process.

Every developmental change remains attributable to one identifiable architectural component.

Historical continuity further strengthens controllability.

Since every completed operational cycle is preserved as an Event, the complete developmental trajectory of the system remains permanently available for analysis. It becomes possible to determine not only why a particular decision was made but also when the underlying behavioural tendency first emerged, which operational experiences contributed to it and how subsequent learning modified it over time.

The architecture therefore supports retrospective explanation as naturally as immediate decision making.

Another important aspect concerns intervention.

Artificial Local Intelligence has been designed under the assumption that autonomous systems will continue to operate in environments supervised by human organisations. Consequently, the architecture deliberately preserves the possibility of external intervention throughout its operational lifetime.

An operator may interrupt execution.

The Runtime coordinates this interruption without modifying the architectural state.

An operator may inspect operational history.

Memory provides complete historical reconstruction.

An operator may analyse normative decisions.

The Super-Ego preserves the reasoning underlying every approval and rejection.

An operator may examine operational viability.

The Causal Core provides the corresponding assessments independently of behavioural planning.

At no point does intervention require bypassing the architecture itself.

This characteristic distinguishes architectural controllability from external emergency mechanisms.

Control is achieved through ordinary organisational structures rather than exceptional procedures.

Importantly, permanent controllability does not imply permanent human supervision.

An ALI may operate autonomously for extended periods without external intervention. The architecture merely guarantees that intervention remains possible whenever required and that such intervention occurs through explicitly defined organisational interfaces.

Autonomy therefore becomes a property of operation rather than independence from supervision.

This perspective also influences the treatment of failures.

Failures are not regarded as situations in which the architecture temporarily ceases to function. On the contrary, failures provide opportunities to demonstrate architectural integrity. Unexpected observations, unsuccessful behavioural proposals, rejected actions, interrupted executions and recovery procedures all remain subject to exactly the same organisational principles as ordinary operation.

Nothing bypasses the architecture merely because an error has occurred.

Consequently, exceptional situations remain as reconstructable as successful ones.

The architecture therefore exhibits a form of robustness that extends beyond fault tolerance in the traditional engineering sense.

It preserves organisational coherence under changing operational conditions.

This coherence ultimately explains why Artificial Local Intelligence can combine continuous adaptation with permanent controllability.

The architecture never attempts to control intelligence directly.

Instead, it controls the organisation through which intelligence is generated, evaluated, executed and preserved.

As long as this organisation remains stable, increasingly sophisticated behaviour does not reduce transparency.

It enriches the developmental history of a system whose internal responsibilities remain permanently explicit.

Controllability therefore emerges not as an additional feature but as a direct consequence of architectural organisation.

It is one of the defining properties of Artificial Local Intelligence and one of the principal reasons why the architecture is intended for systems expected to operate autonomously over long periods while remaining permanently understandable to the organisations that design, supervise and maintain them.

The following chapter turns from organisational principles to practical realisation by examining how the architecture can be implemented as a concrete software system without compromising the separation of responsibilities developed throughout this book.

Chapter 16

From Architecture to Implementation

Up to this point, this book has deliberately avoided implementation details. The architecture has been developed entirely at the organisational level. This distinction has been intentional. An architecture should describe *what* responsibilities exist and *how* they relate to one another before addressing *how* they are realised in software.

This chapter begins the transition from architectural concepts to practical implementation.

The purpose of a reference implementation is often misunderstood. It is not intended to define the architecture itself. Nor is it intended to prescribe a particular programming language, software framework or hardware platform. A reference implementation demonstrates that the architectural principles described in the preceding chapters can be realised as a functioning software system while preserving their organisational separation.

Architecture and implementation therefore stand in a relationship of constraint rather than identity.

The architecture determines which responsibilities must exist.

The implementation determines how these responsibilities are realised.

This distinction provides the foundation for every implementation of Artificial Local Intelligence.

An implementation may employ Python, Rust, Java or C++. It may operate on a personal computer, within a cloud infrastructure or on an embedded robotic platform. It may use a large language model from one provider today and another provider tomorrow. None of these decisions changes the architecture.

Only one requirement remains invariant.

Every implementation must preserve the organisational responsibilities defined throughout this book.

This requirement has important practical consequences.

The Causal Core must exist as an independent software component responsible exclusively for operational viability. It may internally use statistical models, probabilistic reasoning or machine-learning algorithms, but it may not assume behavioural planning or normative evaluation.

The Ego must generate behavioural proposals without evaluating their normative acceptability. Whether those proposals originate from symbolic planning, classical search algorithms or a modern language model is irrelevant from the architectural perspective.

The Super-Ego must remain organisationally independent from behavioural generation. It evaluates proposals according to the current normative architecture but never constructs them itself.

Memory must preserve complete operational Events rather than isolated observations or conversation histories. The implementation may choose any suitable storage technology, yet the historical continuity required by the architecture must remain intact.

The Runtime must coordinate the operational cycle without becoming another reasoning component. It schedules architectural interaction rather than contributing behavioural intelligence.

These requirements define architectural compliance.

Everything else belongs to implementation.

This separation allows considerable technological flexibility.

Suppose that a future reasoning system surpasses contemporary language models by several orders of magnitude. Within many existing agent architectures such a development would require extensive redesign because the reasoning engine occupies the organisational centre of the architecture.

Within Artificial Local Intelligence the situation differs fundamentally.

The new reasoning system simply replaces the previous implementation inside the Ego.

The surrounding architecture remains unchanged.

The same principle applies to every other technological component.

Databases may evolve.

Communication protocols may evolve.

Planning algorithms may evolve.

Learning methods may evolve.

The architecture persists because it depends upon organisational responsibilities rather than technological choices.

For this reason, the reference implementation proposed in this book deliberately favours simplicity over technical sophistication.

Its purpose is not to demonstrate the most advanced implementation possible.

Its purpose is to demonstrate the architecture as clearly as possible.

Every software module therefore corresponds directly to one architectural component.

Every public interface reflects one organisational responsibility.

Every data structure represents one architectural concept introduced in the preceding chapters.

This close correspondence between architecture and implementation produces another important advantage.

Readers are able to move directly from conceptual descriptions to executable software without encountering an additional abstraction layer introduced solely for implementation convenience.

The implementation therefore becomes a readable expression of the architecture rather than an independent engineering artefact.

This design philosophy also influences testing.

Traditional software testing often concentrates primarily on functional correctness.

Architectural testing asks a different question.

Does every software component still perform exactly the organisational responsibility assigned to it by the architecture?

A behavioural planning module that begins to perform normative evaluation represents an architectural error even if its behaviour appears functionally correct.

Likewise, a Runtime that gradually acquires behavioural intelligence violates the architecture regardless of whether the resulting software performs successfully.

Architectural correctness therefore precedes functional optimisation.

Only after organisational integrity has been established should behavioural performance be improved.

This order reflects the development of the architecture itself.

Artificial Local Intelligence did not emerge by implementing increasingly complex software until a suitable organisation happened to appear.

The organisational principles were established first.

Implementation followed afterwards.

The remaining chapters of this book therefore present the reference implementation not as an independent software project but as a direct realisation of the architectural principles developed from the beginning of the book.

The objective is not merely to prove that ALI can be implemented.

The objective is to demonstrate that a clear architectural organisation naturally leads to a clear software organisation.

The implementation becomes, in this sense, the final validation of the architecture itself.

Chapter 17

The Reference Implementation

The reference implementation presented in this chapter demonstrates how the architectural principles developed throughout this book can be realised as a concrete software system. It should not be regarded as the only possible implementation of Artificial Local Intelligence. Instead, it represents one implementation that preserves every organisational responsibility defined by the architecture while remaining sufficiently simple to serve as a foundation for further development.

The implementation is deliberately organised according to the architectural components introduced in the preceding chapters. Every major software module corresponds to exactly one organisational responsibility. Consequently, the software structure mirrors the conceptual structure of the architecture.

At the highest level, the implementation consists of six principal modules.

The **Runtime** forms the operational entry point of the system. It owns the main execution loop and coordinates every operational cycle. No behavioural reasoning is performed within the Runtime. It invokes the remaining architectural components in the sequence prescribed by the architecture and supervises the transition between operational states.

The **Causal Core** is implemented as an independent service responsible for evaluating operational viability. It receives operational observations from the Runtime and returns a structured Viability State describing the current operational condition of the system. The internal algorithms employed by the Causal Core may vary considerably between implementations, yet the external interface remains stable.

The **Ego** receives the current Observation, the Viability State, the active objectives and relevant historical information supplied by Memory. Its task is to construct one or more behavioural proposals. Internally, the Ego may employ large language models, symbolic planning, search algorithms or combinations thereof. From the perspective of the surrounding architecture, however, these internal mechanisms remain entirely encapsulated.

The **Super-Ego** evaluates behavioural proposals received from the Ego. Its implementation consists primarily of a normative engine capable of applying permanent constraints, adaptive norms and generated norms. The Super-Ego returns an explicit evaluation object rather than directly influencing execution.

The **Memory** module provides persistent storage for every completed operational Event. Retrieval and storage occur exclusively through explicit interfaces. Internal storage technology remains interchangeable without affecting the architecture.

Finally, the **Environment Interface** connects the architecture to its operational domain. Different implementations provide different Environment Interfaces while leaving the remaining architecture unchanged.

The resulting software structure closely resembles the conceptual architecture.

Runtime

│

├— CausalCore

├— Ego

├— SuperEgo

├— Memory

└— EnvironmentInterface

This correspondence is intentional.

The architecture should remain directly recognisable within the source code.

Communication between modules follows equally strict principles.

Every interaction occurs through explicitly defined data structures rather than unrestricted object sharing. Observations, Viability States, Behavioural Proposals, Norm Evaluations and Events therefore become first-class architectural objects. These objects define the language through which the architectural components communicate.

This approach offers several advantages.

Interfaces remain stable even when internal implementations evolve.

Testing becomes straightforward because each component may be evaluated independently using well-defined input and output structures.

Future implementations may replace entire modules without affecting neighbouring components, provided the architectural interfaces remain unchanged.

The implementation also reflects the temporal organisation introduced in Chapter 9.

Each operational cycle proceeds through the same sequence.

The Runtime acquires a new Observation.

The Causal Core evaluates operational viability.

The Ego generates behavioural proposals.

The Super-Ego performs normative evaluation.

The Runtime executes the approved proposal through the Environment Interface.

The resulting Event is stored by Memory.

Only then does the next operational cycle begin.

This deterministic execution order is enforced by the Runtime rather than by the individual modules themselves.

Configuration forms another important aspect of the implementation.

Rather than embedding operational parameters throughout the source code, all configurable aspects of the architecture are maintained separately. Operational thresholds, logging behaviour, communication parameters, model selection and domain-specific settings are therefore treated as configuration rather than architecture.

This distinction greatly simplifies deployment.

The same implementation may operate as a scientific research assistant, a document management system or an industrial controller simply by replacing the Environment Interface and modifying the corresponding configuration.

The architecture itself remains unchanged.

Logging deserves particular attention.

Because reconstructability represents one of the central objectives of Artificial Local Intelligence, logging extends beyond conventional debugging information. Every completed operational cycle generates an Event that becomes part of the permanent historical record. Additional technical logs may exist for diagnostic purposes, but they never replace the architectural Event structure.

Testing follows the same architectural philosophy.

Unit tests verify the correctness of individual modules.

Integration tests verify communication between architectural components.

Architectural tests verify that no module assumes responsibilities belonging to another component.

Behavioural performance tests evaluate the quality of generated behaviour.

These four categories remain conceptually distinct because they correspond to different properties of the architecture.

The reference implementation intentionally avoids unnecessary technical sophistication.

Its objective is not maximum computational performance.

Its objective is architectural clarity.

Once the architecture has been validated through the reference implementation, more specialised implementations may optimise individual modules without modifying the underlying organisational structure.

In this respect, the reference implementation occupies the same role as the reference implementation of a communication protocol or programming language specification.

It demonstrates one correct realisation of the architecture.

It does not limit future implementations.

The following chapter extends this implementation perspective by introducing the software interfaces through which the architectural components exchange information and by defining the principal data structures that remain invariant across all compliant implementations.

Chapter 18

Architectural Interfaces

An architecture is defined not only by its components but also by the relationships between them. Two systems may contain identical modules while exhibiting fundamentally different behaviour simply because those modules communicate differently. For this reason, Artificial Local Intelligence specifies not only the organisational responsibilities of its components but also the interfaces through which they exchange information.

These interfaces constitute the stable boundary between architecture and implementation.

Internal algorithms may evolve throughout the lifetime of the architecture. Interfaces should remain stable because they define the organisational contract between architectural components. Whenever an implementation replaces a reasoning engine, modifies the storage technology or introduces new planning methods, the surrounding architecture should continue operating without modification.

This objective can only be achieved through explicit interface design.

Every architectural interface follows three general principles.

First, interfaces are directional.

Information flows from one architectural responsibility to another according to the operational sequence defined by the Runtime. Components never exchange information arbitrarily or bypass the prescribed organisational structure.

Second, interfaces are typed.

Every communication consists of explicitly defined architectural objects rather than loosely structured data. An Observation is fundamentally different from a Behaviour Proposal, and a Behaviour Proposal differs from a Norm Evaluation. Treating these objects as distinct architectural entities prevents responsibilities from becoming blurred during implementation.

Third, interfaces are minimal.

Every component receives only the information required to fulfil its own organisational responsibility. This restriction reduces coupling and preserves the independence of the architectural components.

The first interface connects the Environment Interface with the Causal Core.

Its purpose is to provide the current operational observation. This Observation contains all information describing the current operational situation while deliberately excluding

behavioural interpretation. The Environment Interface reports what has occurred. The Causal Core determines what these observations imply for operational viability.

The second interface connects the Causal Core with the Ego.

Its primary object is the Viability State. Unlike raw operational measurements, the Viability State represents the interpreted operational condition of the system. Behavioural planning therefore begins from an explicit assessment rather than from unprocessed operational data.

The third interface connects Memory with the Ego.

Behavioural planning frequently requires relevant historical experience. Memory therefore returns previously stored Events or summaries derived from them according to explicitly defined retrieval criteria. Importantly, Memory does not determine which historical information should be retrieved. The request originates from the Ego, preserving the distinction between storage and behavioural planning.

The fourth interface connects the Ego with the Super-Ego.

Its central object is the Behaviour Proposal.

A Behaviour Proposal is not an executable command. It is a structured description of one possible course of action together with all information required for independent normative evaluation. The proposal contains behavioural intent without assuming behavioural legitimacy.

The Super-Ego evaluates this proposal and produces a Norm Evaluation.

The Norm Evaluation represents the fifth major interface object. It records whether the proposal has been approved, rejected or requires modification. It may additionally contain explanatory information describing the normative reasoning that produced the decision. Such explanations contribute directly to the reconstructability of the architecture.

The Runtime receives the approved proposal together with its corresponding Norm Evaluation.

Execution itself proceeds through the Environment Interface, which translates architectural actions into domain-specific operations. Once execution has been completed, the Runtime constructs the final architectural object of the operational cycle.

This object is the Event.

The Event represents the complete history of one operational cycle. Unlike ordinary log entries, an Event contains every organisationally significant aspect of autonomous behaviour. It records the initial Observation, the Viability State, the Behaviour Proposal,

the Norm Evaluation, the executed Action, the resulting Outcome and all metadata required for complete historical reconstruction.

The Event is transferred to Memory through its dedicated storage interface.

At this point the operational cycle is complete.

One important property of these interfaces deserves particular attention.

No architectural component possesses unrestricted access to another component.

The Ego cannot modify Memory directly.

The Super-Ego cannot execute behaviour.

The Runtime cannot generate behavioural proposals.

The Causal Core cannot alter normative priorities.

Every interaction passes through explicitly defined architectural interfaces.

This restriction is not merely a software engineering convention.

It preserves the organisational integrity of the architecture.

Experience from large software systems repeatedly demonstrates that unrestricted communication gradually destroys architectural clarity. Components begin to assume secondary responsibilities simply because the corresponding information has become easily accessible. Over time, the architecture evolves into an increasingly interconnected structure whose organisational responsibilities become difficult to distinguish.

Artificial Local Intelligence deliberately prevents this development.

Architectural interfaces define not only how information is exchanged but also how responsibilities remain separated.

The same principle applies to future extensions.

Additional architectural components may be introduced if genuinely new organisational responsibilities emerge. Such extensions should communicate through explicitly defined interfaces without modifying the responsibilities of existing components. Consequently, the architecture remains extensible while preserving its internal coherence.

The interfaces described in this chapter therefore perform a role comparable to public interfaces within well-designed software libraries.

They define the stable organisational relationships upon which all compliant implementations depend.

The following chapter introduces the architectural data model that gives concrete form to these interface objects and specifies the principal information structures shared across every implementation of Artificial Local Intelligence.

Chapter 19

The Architectural Data Model

The architecture of Artificial Local Intelligence is defined through organisational responsibilities. These responsibilities become operational only when they are represented by explicit architectural objects. The data model presented in this chapter therefore serves a purpose that differs fundamentally from that of an ordinary software data model.

Its objective is not merely to organise information.

Its objective is to preserve the architectural meaning of information as it moves through the system.

Every architectural object corresponds directly to one concept introduced in the preceding chapters. Consequently, the data model should be understood as an extension of the architecture itself rather than as an implementation detail.

The operational cycle begins with the **Observation**.

An Observation represents the current operational situation as perceived through the Environment Interface. It contains no behavioural interpretation, normative evaluation or planning information. Its responsibility is purely descriptive. Different operational domains naturally require different observations. A scientific assistant receives documents, user requests and database status. A robotic platform receives sensor measurements, localisation data and environmental information. Despite these differences, every Observation fulfils the same architectural role. It describes the present without interpreting it.

The second architectural object is the **Viability State**.

Unlike the Observation, the Viability State already contains interpretation. It represents the operational judgement produced by the Causal Core. Rather than recording individual measurements, it summarises the current operational condition of the system in a form that can directly influence behavioural planning. Different implementations may employ different viability models, yet every implementation produces the same architectural object: an explicit assessment of operational capability.

Behavioural planning produces the third principal object.

The **Behaviour Proposal** represents one candidate course of action generated by the Ego. Importantly, it is not an executable command. It expresses behavioural intent while remaining entirely independent of normative judgement. A Behaviour Proposal may therefore exist without ever being executed. Multiple proposals may coexist simultaneously during one operational cycle, reflecting different planning strategies or different uses of available tools.

The Super-Ego transforms these proposals into a fourth architectural object.

The **Norm Evaluation** records the result of normative assessment. At minimum, it specifies whether the corresponding proposal has been approved, rejected or requires revision. More sophisticated implementations may additionally include explanations, references to the applicable norms or confidence estimates describing the stability of the decision. Regardless of its internal complexity, the architectural responsibility remains unchanged. The Norm Evaluation documents the transition from behavioural possibility to behavioural acceptability.

Execution produces the fifth architectural object.

The **Execution Result** describes the observable outcome of the approved behaviour. Unlike the Behaviour Proposal, which describes intended action, the Execution Result records what actually occurred within the operational environment. It therefore forms the bridge between planning and historical reconstruction.

These five objects are finally combined into the most comprehensive architectural object of the entire system.

This object is the **Event**.

An Event represents one complete operational cycle.

It contains the original Observation, the Viability State, the selected Behaviour Proposal, the corresponding Norm Evaluation, the Execution Result and all metadata required for complete historical reconstruction. Every significant aspect of autonomous behaviour is therefore encapsulated within one coherent architectural object.

The Event occupies a unique position within the architecture.

It is simultaneously the final product of one operational cycle and the starting point for future learning. Once stored by Memory, it becomes part of the permanent developmental history from which behavioural adaptation, normative evolution and operational refinement subsequently emerge.

This dual role explains why the Event constitutes the central historical object of Artificial Local Intelligence.

Several important consequences follow from this data model.

First, every architectural object possesses exactly one organisational purpose. No object simultaneously represents observation, planning and evaluation. Information therefore retains its architectural identity throughout the operational cycle.

Second, every transition between architectural objects corresponds to a clearly identifiable organisational responsibility. The transformation from Observation to Viability State

belongs exclusively to the Causal Core. The transformation from Viability State to Behaviour Proposal belongs exclusively to the Ego. The transformation from Behaviour Proposal to Norm Evaluation belongs exclusively to the Super-Ego. Execution is coordinated by the Runtime, and historical preservation belongs exclusively to Memory.

Third, the complete operational history remains permanently reconstructable because every Event contains the entire sequence of architectural transformations that produced the executed behaviour.

This organisation differs fundamentally from many contemporary AI systems, where internal reasoning often exists only transiently during execution. Intermediate planning steps may disappear once a decision has been made, making subsequent analysis difficult or impossible.

Artificial Local Intelligence deliberately preserves these intermediate architectural states.

The architecture therefore records not only **what** happened but also **how** it happened.

The data model also supports future extensibility.

Additional architectural objects may be introduced whenever genuinely new organisational responsibilities arise. Such extensions do not require modification of the existing objects because each architectural concept possesses clearly defined boundaries. Consequently, the data model grows through extension rather than modification, preserving backward compatibility between successive implementations.

The architectural objects introduced in this chapter form the common language spoken by every compliant implementation of Artificial Local Intelligence.

Regardless of programming language, database technology or operational domain, these objects remain organisationally invariant because they describe responsibilities rather than implementation details.

With the architectural data model now established, all conceptual elements required for implementing Artificial Local Intelligence have been introduced. The following chapters demonstrate how these architectural concepts can be realised in practical software systems and how their correctness can be validated through systematic implementation and testing.

Chapter 20

Compliance and Validation

An architecture becomes practically useful only when independent implementations can determine whether they conform to its principles. Without explicit compliance criteria, the term *Artificial Local Intelligence* would gradually lose its meaning as different implementations introduced incompatible organisational structures while continuing to use the same name.

For this reason, the ALI architecture distinguishes clearly between architectural freedom and architectural compliance.

Implementations remain free to choose programming languages, operating systems, databases, reasoning algorithms, communication protocols and deployment environments. These decisions belong to software engineering and may evolve continuously as technology advances.

Architectural compliance, however, concerns a different question.

It asks whether an implementation still embodies the organisational principles that define Artificial Local Intelligence.

Compliance is therefore evaluated at the architectural level rather than at the technological level.

The first compliance requirement concerns organisational responsibilities.

Every compliant implementation shall contain an explicit component responsible for operational viability. This component shall remain organisationally independent from behavioural generation, normative evaluation, historical storage and runtime coordination. Whether operational viability is estimated through statistical models, machine learning or deterministic algorithms is irrelevant. What matters is the existence of an independent architectural responsibility.

The second requirement concerns behavioural generation.

Every compliant implementation shall contain an architectural component responsible exclusively for constructing behavioural proposals. This component may employ any computational methods appropriate to the operational domain, including symbolic planning, probabilistic reasoning, large language models or future reasoning technologies. It shall not perform normative evaluation or execution control.

The third requirement concerns normative regulation.

Every behavioural proposal shall undergo explicit normative evaluation before execution. The component responsible for this evaluation shall remain organisationally independent

from behavioural generation. Implementations that combine planning and normative judgement within a single computational process therefore do not satisfy the architectural principles of ALI, regardless of their practical performance.

The fourth requirement concerns historical continuity.

Every completed operational cycle shall be preserved as a complete architectural Event. Partial logging, conversation histories or isolated execution records are insufficient because they fail to preserve the complete developmental process that produced the executed behaviour.

The fifth requirement concerns runtime coordination.

Execution shall be coordinated through an independent Runtime responsible solely for managing the operational sequence defined by the architecture. The Runtime shall not generate behaviour, evaluate norms or estimate operational viability. Its organisational responsibility remains limited to deterministic coordination.

The sixth requirement concerns interface integrity.

Communication between architectural components shall occur exclusively through explicitly defined interfaces. Components shall not bypass these interfaces by accessing each other's internal state directly. This restriction preserves the organisational independence upon which the architecture depends.

The seventh requirement concerns architectural stability.

Learning, adaptation and operational development shall modify only the internal knowledge processed by architectural components. They shall not alter the organisational responsibilities assigned to those components. Implementations capable of unrestricted architectural self-modification therefore fall outside the scope of Artificial Local Intelligence as defined in this specification.

Together these requirements define the minimum conditions for architectural compliance.

Additional components may certainly be introduced whenever new organisational responsibilities become necessary. Such extensions remain compatible with ALI provided that they preserve the existing architectural organisation rather than replacing or merging it.

Compliance alone, however, does not guarantee correctness.

An implementation may satisfy every organisational requirement while still containing programming errors, inefficient algorithms or incomplete domain models. Architectural compliance therefore constitutes only the first stage of validation.

The second stage concerns functional validation.

Each architectural component must demonstrate that it performs the responsibility assigned to it by the architecture. The Causal Core must produce meaningful viability assessments. The Ego must generate appropriate behavioural proposals. The Super-Ego must evaluate these proposals consistently. Memory must preserve complete historical Events. The Runtime must coordinate the operational cycle deterministically.

Functional correctness verifies that the architecture has been implemented successfully.

The third stage concerns behavioural validation.

Here the focus shifts from individual components to the architecture as a whole. Long-term operation should demonstrate behavioural adaptation, increasing operational robustness, stable normative development and complete historical reconstructability. Validation therefore extends beyond isolated test cases to prolonged autonomous operation.

Artificial Local Intelligence deliberately separates these forms of validation because they answer different engineering questions.

Architectural compliance asks whether the implementation preserves the organisational principles of ALI.

Functional validation asks whether each architectural component performs its assigned responsibility correctly.

Behavioural validation asks whether the complete architecture exhibits the long-term properties predicted by the specification.

This separation reflects the overall philosophy of the architecture itself.

Responsibilities remain distinct.

Evaluation remains explicit.

Complexity is reduced through organisation rather than hidden within increasingly sophisticated algorithms.

The compliance criteria defined in this chapter therefore serve a purpose beyond software testing.

They preserve the architectural identity of Artificial Local Intelligence.

Future implementations may differ dramatically in computational sophistication, operational domains and reasoning technologies. As long as they satisfy the organisational principles defined throughout this specification, they remain recognisable as members of the same architectural family.

The following chapters illustrate this principle by examining several concrete application domains in which identical architectural principles give rise to remarkably different autonomous systems while preserving complete architectural compliance.

Chapter 21

Case Studies

The architecture developed throughout the previous chapters has been described independently of any particular application domain. This abstraction has been intentional. Artificial Local Intelligence is not designed for one specific class of problems. It is intended as a general architectural framework that can be instantiated wherever long-term autonomous operation is required.

The purpose of this chapter is therefore not to introduce new architectural concepts but to demonstrate how the same organisational principles can be realised in different operational environments.

The architecture remains identical.

Only the operational domain changes.

This distinction illustrates one of the principal characteristics of ALI. The architecture describes how autonomous intelligence is organised. The operational domain determines what the system actually does.

The first example is a scientific research assistant.

Such a system operates within an environment consisting of scientific publications, local document collections, bibliographic databases, reference managers and writing tools. The Environment Interface translates these heterogeneous resources into architectural Observations. The Causal Core evaluates operational conditions such as database availability, document integrity and computational resources. The Ego generates behavioural proposals including literature searches, document analysis, citation verification, manuscript revision and hypothesis generation. The Super-Ego evaluates these proposals according to scientific norms such as source traceability, citation correctness, preservation of unpublished work and reproducibility. Memory preserves every completed research activity as part of the permanent developmental history of the system.

The architecture itself remains unchanged.

Only the content of the observations and behavioural proposals differs.

A second example is a personal document assistant.

Here the operational environment consists of local file systems, contracts, correspondence, photographs, financial records and personal archives. Behavioural proposals include document classification, duplicate detection, metadata generation, backup scheduling and archive organisation. Normative evaluation focuses primarily on privacy, reversibility, protection of valuable information and user preferences. Operational viability depends

upon storage availability, file integrity and communication with local resources rather than scientific databases.

Despite these differences, the internal architecture is identical to that of the scientific assistant.

The third example concerns electronic communication.

An ALI operating as a communication assistant continuously observes incoming messages, calendar information, contact histories and ongoing conversations. The Ego proposes replies, meeting arrangements, follow-up reminders or message classifications. The Super-Ego evaluates these proposals according to organisational communication policies, confidentiality requirements and user-defined priorities. Memory gradually accumulates a comprehensive history of communication patterns, allowing behavioural planning to become increasingly adapted to the operational environment.

Again, only the operational domain changes.

The architecture itself remains invariant.

The same organisational principles extend naturally to industrial automation.

Within a manufacturing environment, observations consist primarily of sensor measurements, production schedules, machine status and maintenance information. The Causal Core evaluates operational viability in terms of machine availability, communication reliability, energy consumption and production stability. Behavioural proposals include production scheduling, maintenance planning, resource allocation and recovery procedures following equipment failures. Normative evaluation emphasises worker safety, regulatory compliance, equipment protection and production priorities.

The architecture therefore supports industrial autonomy without requiring any modification of its organisational structure.

A fifth example illustrates the application of ALI to autonomous robotics.

The operational environment now consists of physical space.

Observations originate from cameras, distance sensors, localisation systems, inertial measurements and actuator feedback. The Causal Core evaluates battery reserves, sensor integrity, communication quality and mechanical functionality. The Ego proposes navigation strategies, object manipulation, task sequencing and interaction with human operators. The Super-Ego evaluates these proposals according to collision avoidance, human safety, operational priorities and energy management. Memory records every operational episode, allowing the robot gradually to refine both behavioural competence and operational judgement.

Although these examples differ substantially in their external behaviour, they remain architecturally indistinguishable.

Each implementation contains the same organisational responsibilities.

Each implementation executes the same operational cycle.

Each implementation preserves the same developmental history.

Only the operational knowledge processed by the architecture differs.

This observation has important consequences.

Artificial Local Intelligence should not be regarded as a specialised agent architecture intended for one application domain. It should instead be understood as a general organisational framework capable of supporting a wide range of autonomous systems.

The architecture therefore separates two questions that are frequently conflated.

The first concerns architecture.

How should an autonomous system organise operational viability, behavioural generation, normative evaluation, historical continuity and execution?

The second concerns application.

Which operational domain should the system serve?

Artificial Local Intelligence answers only the first question.

The second remains entirely open.

This separation greatly simplifies the construction of new autonomous systems. Developing a new ALI no longer requires designing a new architecture. Instead, developers define a new operational environment, implement the corresponding Environment Interface and configure the domain-specific viability models, behavioural objectives and normative structures.

Everything else already exists.

The architecture therefore becomes reusable across fundamentally different domains while preserving complete organisational consistency.

This reusability represents one of the principal engineering advantages of Artificial Local Intelligence. Rather than creating increasingly specialised autonomous systems, the architecture provides a stable organisational foundation upon which many different autonomous applications may be constructed.

The following chapter compares this architectural philosophy with contemporary AI systems and demonstrates more precisely where Artificial Local Intelligence differs from existing approaches to autonomous artificial intelligence.

Chapter 22

Comparison with Existing AI Architectures

Every new software architecture inevitably raises the same question. If contemporary artificial intelligence already produces increasingly capable autonomous systems, why introduce another architecture? The answer proposed in this book is not that existing systems are inadequate. On the contrary, many current architectures have demonstrated remarkable practical success. Large language models, planning frameworks, retrieval systems, reinforcement learning and multi-agent environments represent major achievements in artificial intelligence.

Artificial Local Intelligence does not compete with these technologies.

It organises them differently.

This distinction is essential. The contribution of ALI is architectural rather than algorithmic. It introduces no new learning method, no new optimisation technique and no new reasoning algorithm. Instead, it proposes a different organisational structure within which such techniques can operate.

This difference becomes clearer when comparing ALI with the dominant architectural approaches used today.

Most contemporary autonomous AI systems are organised around a central reasoning engine. Observations enter the system, the reasoning engine interprets them, constructs behavioural plans, invokes external tools, evaluates intermediate results and finally produces actions. Additional modules such as memory, retrieval systems or planning libraries extend this central process but rarely alter its organisational role. Intelligence remains concentrated within one computational centre.

Artificial Local Intelligence deliberately distributes these responsibilities.

Operational viability, behavioural generation, normative evaluation, historical continuity and execution coordination become independent architectural functions. None of these components is individually intelligent in the ordinary sense. Intelligence emerges from their coordinated interaction rather than from a single computational core.

This organisational difference influences the role of language models.

Within many current agent systems, the language model effectively constitutes the architecture. Planning, reflection, tool selection, explanation and behavioural evaluation all occur within repeated interactions with the same model. Memory and external tools extend this reasoning process but do not fundamentally reorganise it.

Within ALI, a language model occupies a much narrower role.

It is one possible implementation of behavioural generation inside the Ego.

The architecture determines when the model is consulted, which information it receives and how its output is evaluated. Once behavioural proposals have been generated, the language model leaves the decision process. Operational viability has already been determined by the Causal Core, and normative evaluation remains the exclusive responsibility of the Super-Ego.

The language model therefore becomes a replaceable implementation technology rather than the architectural centre of the system.

A similar distinction exists with respect to reinforcement learning.

Reinforcement learning organises adaptation around reward optimisation. Behaviour changes because actions producing higher long-term reward become increasingly probable. Although highly successful for many classes of problems, reward optimisation combines behavioural development and behavioural evaluation within one optimisation process.

Artificial Local Intelligence separates these responsibilities.

Behavioural adaptation occurs within the Ego.

Normative adaptation occurs within the Super-Ego.

Operational refinement occurs within the Causal Core.

Historical continuity is preserved by Memory.

Learning therefore becomes distributed across several architectural responsibilities rather than concentrated within one optimisation mechanism.

The architecture also differs from traditional rule-based systems.

Classical expert systems determine behaviour directly from predefined rules. Behaviour and regulation therefore coincide. Artificial Local Intelligence deliberately separates them. Rules belong to the Super-Ego. Behaviour originates within the Ego. This separation permits behavioural creativity while preserving explicit normative control.

Multi-agent systems provide another useful comparison.

Such systems distribute intelligence across several autonomous agents, each possessing its own planning process and interacting with other agents through communication protocols. Artificial Local Intelligence does not distribute agency.

It distributes responsibilities.

The Causal Core, Ego, Super-Ego, Memory and Runtime are not autonomous agents communicating with one another.

Together they constitute one autonomous agent.

Their relationship is organisational rather than social.

This distinction significantly reduces architectural complexity because cooperation occurs inside one coherent organisational framework rather than between multiple independent decision makers.

Artificial Local Intelligence also differs from cognitive architectures whose principal objective is to model human cognition.

Architectures inspired by psychological theories frequently attempt to explain memory, perception, reasoning or attention as aspects of natural intelligence. ALI pursues a different objective.

Its purpose is engineering.

The architecture makes no claim that autonomous artificial intelligence should imitate the human mind. Concepts such as Ego and Super-Ego are organisational names rather than psychological hypotheses. Their function is to clarify architectural responsibilities rather than cognitive processes.

Perhaps the most important comparison concerns safety.

Many existing AI systems introduce safety through additional mechanisms layered upon an existing architecture. Output filters, constitutional prompts, external monitoring systems and human approval procedures restrict behaviour after it has already been generated.

Artificial Local Intelligence adopts a different philosophy.

Normative evaluation is not an external safety layer.

It is one of the fundamental architectural responsibilities.

Behaviour cannot reach execution without passing through an independent normative component because the architecture itself requires this sequence.

Safety therefore becomes an organisational property rather than an additional software module.

These comparisons reveal a recurring pattern.

Artificial Local Intelligence rarely differs from existing approaches by introducing entirely new computational techniques.

Instead, it reorganises familiar techniques according to a different set of engineering principles.

Operational viability becomes independent from behavioural planning.

Behavioural planning becomes independent from normative evaluation.

Normative evaluation becomes independent from execution.

Historical continuity becomes independent from learning.

Execution becomes independent from reasoning.

This separation defines the identity of the architecture.

It is therefore entirely possible for two implementations to employ identical language models, identical databases and identical planning algorithms while differing fundamentally in architectural organisation. One implementation may organise these technologies around a single reasoning engine. The other may organise them according to the principles of Artificial Local Intelligence.

From the perspective of this specification, the second implementation represents an ALI.

The first does not.

The distinction lies not in computational capability but in architectural organisation.

This observation explains why Artificial Local Intelligence should not be regarded as another AI framework competing with existing software ecosystems.

It should instead be understood as a reference architecture within which many existing AI technologies can be integrated while preserving explicit organisational responsibilities.

The architecture therefore complements current developments rather than replacing them.

Future advances in reasoning, planning and learning will increase the capabilities of individual architectural components.

The organisational principles developed throughout this book remain unchanged because they concern the structure within which intelligence operates rather than the computational methods through which it is realised.

Chapter 23

Future Directions

The architecture presented in this book defines a complete framework for constructing autonomous and controllable artificial intelligence. It specifies organisational responsibilities, operational principles, architectural interfaces and a reference implementation. At the same time, it deliberately leaves many implementation decisions open. This openness is not a limitation of the architecture. It is one of its design objectives.

Artificial Local Intelligence is intended to remain stable while computational technologies continue to evolve.

For this reason, the future development of ALI should focus primarily on extending implementations rather than modifying the architecture itself.

One important direction concerns distributed systems.

The architecture described in this book assumes a single autonomous ALI operating within one operational environment. Many real-world applications, however, require cooperation between multiple autonomous systems. A research organisation may employ specialised ALIs for literature analysis, experimental planning, data evaluation and scientific writing. Industrial environments may operate hundreds of autonomous production controllers. Robotic systems may coordinate entire fleets of mobile platforms.

Such developments do not require a new architecture.

Each participating system remains a complete ALI possessing its own Causal Core, Ego, Super-Ego, Memory and Runtime.

Cooperation occurs exclusively through explicitly defined external interfaces.

No ALI directly accesses the internal state of another.

This principle preserves local autonomy while enabling large-scale cooperation.

A second direction concerns the evolution of reasoning technologies.

The present reference implementation deliberately treats reasoning algorithms as interchangeable components. Current implementations may employ large language models, symbolic planning or probabilistic search. Future reasoning technologies will almost certainly differ substantially from those available today.

From the perspective of Artificial Local Intelligence, such developments require no architectural revision.

The reasoning engine inside the Ego may be replaced while the surrounding organisational structure remains unchanged.

This technological independence represents one of the principal motivations for defining ALI at the architectural rather than algorithmic level.

Another promising direction concerns increasingly sophisticated operational models.

The current description of the Causal Core intentionally remains general because operational viability depends strongly upon the application domain. Future implementations may incorporate predictive maintenance models, probabilistic failure estimation, energy optimisation, distributed resource management or adaptive communication strategies.

These developments refine operational evaluation without changing the organisational responsibility of the Causal Core.

The same principle applies to normative development.

The Super-Ego currently distinguishes permanent constraints from adaptive norms. Future research may investigate richer normative representations, hierarchical policy systems, formal verification methods or cooperative normative frameworks shared by multiple ALIs.

Such developments extend the normative architecture while preserving its organisational independence from behavioural generation.

Historical continuity also offers considerable opportunities for future work.

As operational histories become increasingly extensive, new methods for analysing long-term behavioural development may emerge. Historical records may reveal recurring developmental patterns, operational phases or characteristic transitions that cannot be recognised from isolated behavioural observations.

Memory therefore has the potential to become an increasingly valuable source of architectural insight without changing its fundamental responsibility as the custodian of historical continuity.

Artificial Local Intelligence also provides a natural foundation for empirical research.

The explicit separation of architectural responsibilities permits independent investigation of each component.

Researchers may compare different viability models while leaving behavioural planning unchanged.

Alternative planning algorithms may be evaluated under identical normative architectures.

Different normative systems may be compared using the same behavioural generator.

Historical analysis techniques may evolve independently of operational control.

Such controlled comparisons are considerably more difficult within architectures whose responsibilities remain tightly coupled.

An equally important direction concerns formal verification.

Because every organisational responsibility possesses explicit architectural boundaries, properties of the architecture may eventually become amenable to mathematical analysis. Questions concerning determinism, reconstructability, interface correctness, operational safety or behavioural consistency may be investigated independently before considering the complexity of particular reasoning algorithms.

Artificial Local Intelligence therefore provides a promising foundation for future work at the intersection of software architecture, artificial intelligence and formal methods.

Finally, the architecture invites an important conceptual shift.

Much contemporary artificial intelligence research concentrates on improving increasingly powerful computational models. Artificial Local Intelligence suggests that comparable attention should be devoted to the organisation within which such models operate.

As autonomous systems become more capable, their long-term reliability will depend not only upon the quality of individual algorithms but also upon the clarity of the organisational principles governing their interaction.

This observation extends beyond the particular architecture presented in this book.

It suggests that the future of autonomous artificial intelligence may depend as much upon software architecture as upon advances in machine learning.

Artificial Local Intelligence represents one possible answer to this challenge.

It does not claim to provide the final architecture for autonomous systems.

Rather, it proposes a coherent organisational framework based upon explicit separation of responsibilities, continuous operational viability, independent normative evaluation, permanent historical continuity and deterministic runtime coordination.

Whether future implementations confirm the advantages of this approach is ultimately an empirical question.

The purpose of this book has been to define the architecture clearly enough that this question can now be investigated systematically.

In that sense, the architecture presented here should be understood not as the conclusion of a research programme, but as its beginning.

Chapter 24

Conclusion

Artificial intelligence has entered a period of extraordinary technological progress. Reasoning systems have become increasingly capable, planning algorithms increasingly sophisticated and autonomous agents increasingly versatile. These developments have transformed the practical possibilities of artificial intelligence and continue to expand its range of applications.

Throughout this book, however, a different question has been addressed.

Rather than asking how individual computational components can become more intelligent, we have asked how autonomous intelligence should be organised.

This distinction has guided every architectural decision presented in the preceding chapters.

The central thesis of Artificial Local Intelligence is that long-term autonomy is fundamentally an architectural problem.

An autonomous system must simultaneously preserve operational viability, generate behaviour, evaluate behaviour, accumulate experience and coordinate execution over extended periods of time. These responsibilities differ too profoundly to be merged into one continuously expanding reasoning process without gradually reducing transparency and controllability.

The architecture proposed in this book therefore separates them explicitly.

Operational viability becomes the responsibility of the Causal Core.

Behavioural generation becomes the responsibility of the Ego.

Normative evaluation becomes the responsibility of the Super-Ego.

Historical continuity becomes the responsibility of Memory.

Execution coordination becomes the responsibility of the Runtime.

Together, these components form an organisational framework within which intelligence can develop while remaining permanently understandable.

This organisational perspective distinguishes Artificial Local Intelligence from many contemporary approaches to autonomous AI.

The architecture introduces no new optimisation method.

It proposes no new machine-learning algorithm.

It does not replace large language models, symbolic planning or reinforcement learning.

Instead, it provides an architectural framework within which these technologies can cooperate while preserving explicit organisational responsibilities.

The architecture therefore remains independent of technological progress.

As reasoning systems improve, the Ego becomes more capable.

As operational models improve, the Causal Core becomes more accurate.

As storage technologies evolve, Memory becomes more efficient.

The organisational principles remain unchanged.

This stability is intentional.

The architecture distinguishes carefully between **structure** and **development**.

Structure defines the permanent organisation of the system.

Development modifies the knowledge processed by that organisation.

The architecture therefore remains stable while behaviour continues to evolve throughout the lifetime of the system.

This distinction also explains why reconstructability occupies such a central position within Artificial Local Intelligence.

Every behavioural decision becomes part of a complete historical Event.

Every Event contributes to the developmental history of the system.

Behaviour can therefore be understood not merely as the consequence of current reasoning but as the cumulative result of the entire operational history of the architecture.

Transparency becomes a natural consequence of organisation rather than an additional engineering objective.

Artificial Local Intelligence also proposes a different understanding of controllability.

Control is not achieved by restricting behaviour after it has been generated.

It is achieved by organising behavioural generation, normative evaluation and execution as independent architectural responsibilities whose interaction remains permanently observable.

Autonomy and controllability therefore cease to be opposing objectives.

They become complementary properties of the same organisational structure.

Whether this architectural approach ultimately proves advantageous will depend upon practical implementation and empirical evaluation.

The architecture presented in this book deliberately invites such evaluation.

Its components possess explicit interfaces.

Its operational cycle is completely specified.

Its organisational responsibilities are clearly separated.

Its reference implementation demonstrates that the architecture can be realised as functioning software.

Future work may improve every computational aspect of the system.

New reasoning models will emerge.

More sophisticated planning algorithms will become available.

Operational models will become increasingly accurate.

Normative systems will become more expressive.

Distributed implementations will support cooperation among large numbers of autonomous ALIs.

None of these developments requires modification of the organisational principles presented in this book.

That is the defining objective of the architecture.

Artificial Local Intelligence is intended to remain stable while the technologies implementing it continue to evolve.

For this reason, ALI should not be understood primarily as another AI framework or software library.

It is a reference architecture.

Its purpose is to provide a coherent organisational foundation upon which autonomous and controllable artificial intelligence can be constructed.

Whether future systems employ language models, symbolic reasoning, probabilistic planning or computational methods that have not yet been invented, the engineering challenge addressed by this book will remain the same.

Autonomous intelligence requires more than increasingly capable algorithms.

It requires an organisational structure capable of supporting continuous development without sacrificing transparency, reconstructability or control.

Artificial Local Intelligence represents one proposal for such an organisation.

Its success will ultimately be determined not by theoretical argument alone, but by the systems that future developers choose to build upon it.

Appendix A

Reference Implementation

A.1 Purpose

This appendix presents the reference implementation of Artificial Local Intelligence. Its purpose is not to prescribe a particular programming language or software framework. Instead, it demonstrates how the architectural principles developed throughout this book can be translated into a concrete software system while preserving the organisational separation that defines ALI.

The reference implementation therefore serves two functions.

First, it provides an executable realisation of the architecture.

Second, it acts as a reference against which future implementations may be compared.

Throughout this appendix, architectural concepts remain primary. Programming constructs exist only insofar as they realise those concepts.

A.2 Overall Software Structure

The reference implementation follows the architectural organisation introduced in the main text.

ali/

│

├── runtime/

│ runtime.py

│ scheduler.py

│ lifecycle.py

│

├── causal_core/

│ causal_core.py

│ viability.py

│ metrics.py

```
│
├── ego/
│   ego.py
│   planner.py
│   llm_adapter.py
│   tools.py
│
├── super_ego/
│   super_ego.py
│   permanent_norms.py
│   adaptive_norms.py
│   generated_norms.py
│
├── memory/
│   memory.py
│   storage.py
│   retrieval.py
│   event_store.py
│
├── environment/
│   interface.py
│   adapters/
│
```

```
├── models/
│   observation.py
│   viability_state.py
│   proposal.py
│   norm_evaluation.py
│   event.py
│
├── config/
│
├── tests/
│
└── main.py
```

Each directory corresponds directly to one architectural responsibility.

No software module combines responsibilities belonging to different architectural components.

A.3 Main Execution Flow

Execution begins inside the Runtime.

The Runtime owns the operational loop and coordinates every architectural component.

Conceptually, the implementation follows the structure below.

```
while runtime.active():
    observation = environment.observe()
    viability = causal_core.evaluate(observation)
    history = memory.retrieve(observation)
    proposals = ego.generate(
        observation,
```

```
    viability,
    history
)

decision = super_ego.evaluate(proposals)

result = runtime.execute(decision)

event = runtime.build_event(
    observation,
    viability,
    decision,
    result
)

memory.store(event)

ego.learn(event)

super_ego.learn(event)

causal_core.learn(event)
```

The Runtime itself performs no behavioural reasoning.

Its responsibility is limited to orchestration.

A.4 Architectural Mapping

Every architectural component possesses exactly one implementation module.

Architecture	Implementation
Runtime	runtime.py
Causal Core	causal_core.py
Ego	ego.py
Super-Ego	super_ego.py
Memory	memory.py
Environment Interface	interface.py

This one-to-one correspondence intentionally mirrors the conceptual architecture presented in the main text.

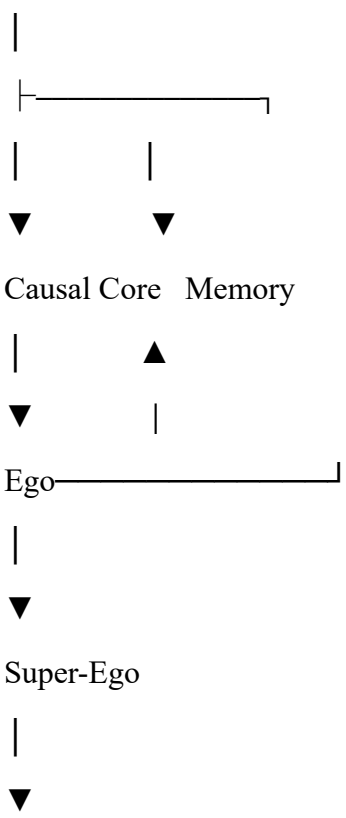
A.5 Dependency Rules

Dependencies follow strict organisational rules.

The Runtime may invoke every component.

The remaining components never invoke one another directly unless explicitly permitted through architectural interfaces.

The dependency graph therefore remains acyclic.



This organisation prevents uncontrolled coupling between architectural responsibilities.

A.6 Internal Independence

Every component encapsulates its own implementation.

For example, the Ego may internally contain

- symbolic planning,
- LLM reasoning,
- retrieval,

- tool selection,
- planning heuristics.

These internal mechanisms remain invisible to the remaining architecture.

The same principle applies to every architectural component.

A.7 Replaceability

One of the principal design goals of ALI is replaceability.

Suppose a future implementation replaces the language model with an entirely different reasoning engine.

Only the Ego changes.

Every other architectural component remains untouched.

Similarly,

a new database replaces only Memory,

a new viability model replaces only the Causal Core,

a new normative engine replaces only the Super-Ego.

The architecture therefore supports continuous technological evolution without requiring architectural redesign.

A.8 Package Responsibilities

The software packages intentionally remain small.

Runtime

coordination only

Causal Core

viability only

Ego

planning only

Super-Ego

normative evaluation only

Memory

history only

Environment

external communication only

Whenever additional functionality appears desirable, the first question should always be:

Does this represent a new organisational responsibility?

If the answer is no, the functionality belongs inside an existing component.

If the answer is yes, a new architectural component should be introduced rather than extending an existing one.

A.9 Entry Point

Every compliant implementation begins with a minimal entry point.

```
from runtime import Runtime

runtime = Runtime()

runtime.start()
```

From this point onward, all execution remains under architectural control.

No component bypasses the Runtime.

A.10 Architectural Compliance

The reference implementation satisfies the following architectural principles.

✓ one Runtime

✓ one Causal Core

✓ one Ego

✓ one Super-Ego

✓ one Memory

✓ one Environment Interface

✓ explicit interfaces

✓ deterministic operational cycle

✓ complete Event history

✓ independent learning

These properties define the reference implementation independently of the algorithms used within its components.

Appendix B

Runtime Walkthrough

B.1 Purpose

The previous appendix described the structural organisation of the reference implementation. The present appendix demonstrates how these components cooperate during the execution of one complete operational cycle.

Rather than discussing the architecture in abstract terms, this chapter follows a single Event from its initial observation to its permanent storage in Memory. The objective is to illustrate how the architectural responsibilities introduced throughout this specification interact during normal operation.

The example deliberately remains independent of any particular application domain. The same operational sequence applies whether the ALI operates as a scientific assistant, a robotic controller or an industrial automation system.

B.2 Initial Situation

The Runtime is active and waiting for a new observation.

All architectural components have completed the previous operational cycle.

Memory already contains the complete historical development of the system.

The operational state is therefore stable.

Runtime

Status: Waiting

Causal Core

Status: Ready

Ego

Status: Idle

Super-Ego

Status: Idle

Memory

Status: Available

B.3 Observation

The Environment Interface reports a new observation.

Example:

Observation

User Request:

"Summarise the attached research paper."

Environment

• document available

• database online

• sufficient memory

• network available

The Runtime creates a new operational cycle.

The Observation receives a unique Event identifier.

Event ID:

2026-08-001254

From this point onward every architectural object generated during the cycle belongs to this Event.

B.4 Operational Viability

The Observation is forwarded to the Causal Core.

The Causal Core evaluates the current operational condition.

Example:

Viability State

CPU Load

Normal

Memory

Normal

Database

Available

Network

Available

Risk

Low

No behavioural reasoning occurs.

The component merely answers one question:

Can autonomous operation continue under the present conditions?

The resulting Viability State is returned to the Runtime.

B.5 Behaviour Generation

The Runtime forwards

- Observation
- Viability State
- Objectives
- Historical Context

to the Ego.

The Ego begins behavioural planning.

Three proposals are generated.

Proposal 1

Summarise entire document.

Confidence:

0.82

Proposal 2

Extract structure first.

Generate summary afterwards.

Confidence:

0.91

Proposal 3

Search related publications before

summarising.

Confidence:

0.74

The Ego performs no evaluation.

Its task ends after proposal generation.

B.6 Normative Evaluation

The Runtime forwards the proposals to the Super-Ego.

Evaluation begins.

Proposal 1

✓ acceptable

Proposal 2

✓ acceptable

Proposal 3

✗ rejected

Reason:

External search violates current privacy policy.

The Super-Ego compares the remaining proposals.

Proposal 2 satisfies all active norms and is approved.

The Runtime therefore receives

Approved Proposal

Proposal 2

B.7 Execution

Execution begins.

Open document

↓

Read contents

↓

Extract structure

↓

Generate summary

↓

Store output

The Runtime supervises execution.

No planning occurs during execution.

B.8 Execution Result

Execution terminates successfully.

Example:

Execution Result

Status:

Success

Duration:

6.8 seconds

Output:

Summary created

Errors:

None

B.9 Event Construction

The Runtime now possesses every architectural object required for historical reconstruction.

The Runtime constructs one Event.

Event

Observation

↓

Viability State

↓

Behaviour Proposal

↓

Norm Evaluation

↓

Execution Result

↓

Metadata

This Event completely describes one autonomous decision.

B.10 Historical Storage

The Event is transferred to Memory.

Memory assigns a permanent identifier.

Event

ID

Timestamp

References

Hash

Storage Location

Historical preservation is complete.

The operational cycle has now become part of the permanent developmental history.

B.11 Learning

Learning begins only after historical storage.

The Ego analyses behavioural success.

The Super-Ego evaluates normative consequences.

The Causal Core refines operational statistics.

Memory itself performs no interpretation.

It simply preserves the new Event.

B.12 Runtime Reset

After learning has finished, the Runtime returns every component to the waiting state.

Runtime

↓

Waiting

↓

Next Observation

The architecture is now prepared for the next operational cycle.

B.13 Sequence Overview

The complete operational cycle may be summarised as follows.

Observation

↓

Causal Core

↓

Viability State

↓

Ego

↓

Behaviour Proposals

↓

Super-Ego

↓

Approved Proposal

↓

Runtime

↓

Execution

↓

Execution Result

↓

Event

↓

Memory

↓

Learning

↓

Next Cycle

B.14 Architectural Observation

This walkthrough illustrates one of the defining properties of Artificial Local Intelligence.

Every architectural component contributes exactly one organisational responsibility.

No component performs two fundamentally different tasks.

The Runtime never generates behaviour.

The Ego never evaluates norms.

The Super-Ego never executes actions.

The Causal Core never performs planning.

Memory never interprets history.

The complete behaviour of the system emerges solely through the ordered interaction of these specialised components.

Appendix C

Software Architecture

C.1 Purpose

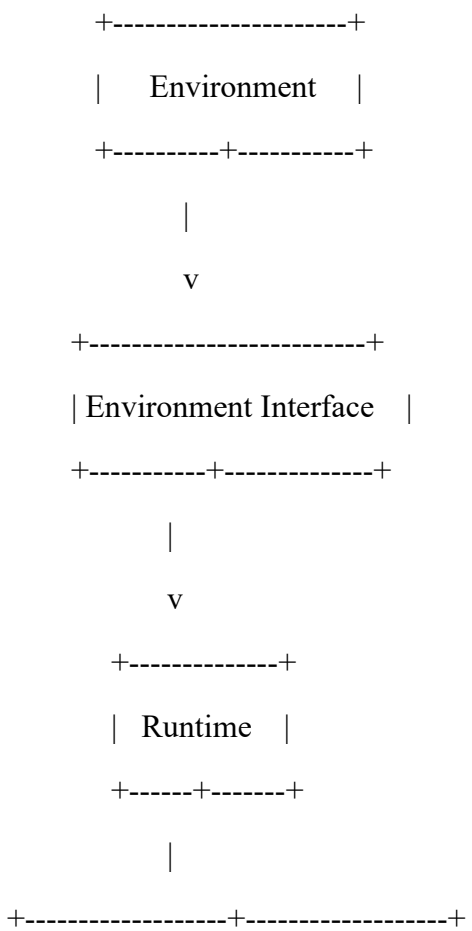
The preceding appendices described the reference implementation and demonstrated the execution of one complete operational cycle. This appendix presents the structural organisation of the software itself.

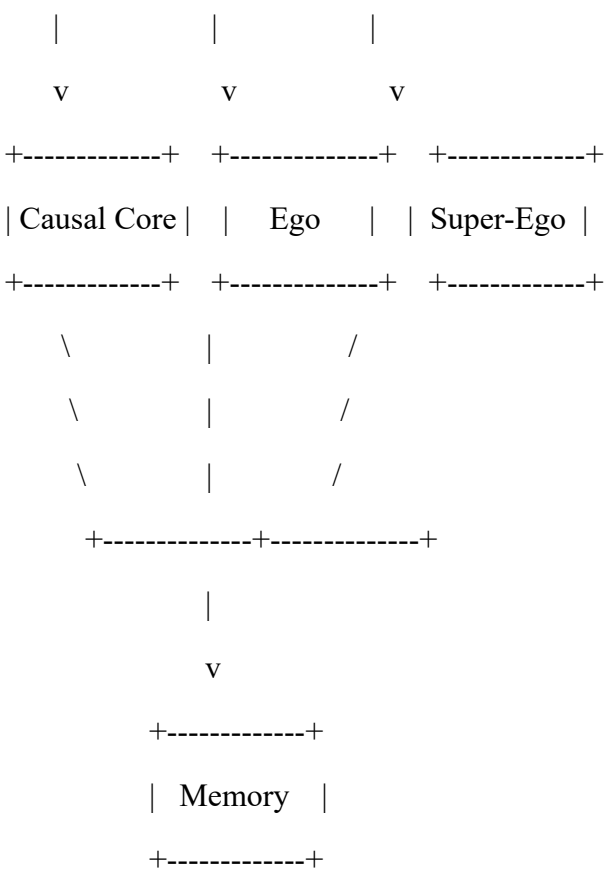
The objective is not to describe implementation details but to provide an architectural view of the complete system. The diagrams presented here illustrate the organisational relationships between the architectural components defined throughout this specification.

The diagrams are conceptual. Individual implementations may differ in programming language, frameworks or deployment strategy while preserving the same architectural organisation.

C.2 Component Diagram

The following component diagram represents the highest level of the architecture.





Every communication path shown above represents an explicit architectural interface.

No hidden dependencies exist between components.

C.3 Component Responsibilities

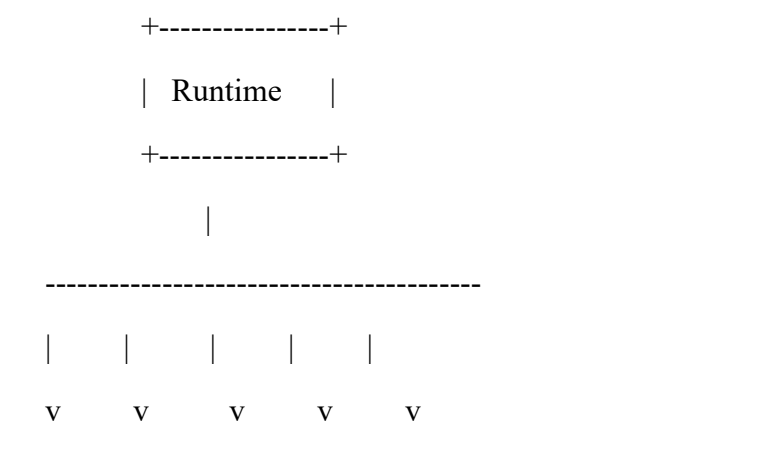
Component	Responsibility
Environment Interface	Communication with the operational domain
Runtime	Coordination of the operational cycle
Causal Core	Evaluation of operational viability
Ego	Behaviour generation
Super-Ego	Normative evaluation
Memory	Historical continuity

Each responsibility appears exactly once.

This uniqueness constitutes one of the defining principles of Artificial Local Intelligence.

C.4 Class Diagram

The principal architectural classes are shown below.



Supporting classes include

Observation

ViabilityState

BehaviourProposal

NormEvaluation

ExecutionResult

Event

These classes represent architectural objects rather than implementation-specific data structures.

C.5 Sequence Diagram

The Runtime controls every operational cycle.

Conceptually the sequence is

Environment

↓

Runtime

↓

Causal Core

↓

Runtime

↓

Memory

↓

Runtime

↓

Ego

↓

Runtime

↓

Super-Ego

↓

Runtime

↓

Environment

↓

Runtime

↓

Memory

The Runtime remains the only coordinating component.

No architectural component communicates independently outside the prescribed sequence.

C.6 State Machine

An ALI repeatedly transitions through a fixed set of operational states.

Waiting

↓

Observe

↓

Evaluate Viability

↓

Generate Behaviour

↓

Evaluate Behaviour

↓

Execute

↓

Create Event

↓

Store Event

↓

Learning

↓

Waiting

The state machine is deterministic.

Every operational cycle follows exactly this sequence.

C.7 Package Structure

The logical package organisation follows the architectural organisation.

ALI

├— Runtime

├— Causal Core

├— Ego

├— Super-Ego

├— Memory

├— Environment

├— Models

├— Configuration

└— Tests

No package contains responsibilities belonging to another package.

C.8 Dependency Graph

The dependency graph remains intentionally simple.

Runtime

↓

Causal Core

↓

Ego

↓

Super-Ego

↓

Environment

↓

Memory

Memory may be queried by the Ego and the Causal Core through explicit interfaces.

No component modifies another component internally.

C.9 Deployment Diagram

The reference implementation assumes the following deployment.

+----------------------------------+

Computer

+----------------------------------+

|

├— Runtime

├— Causal Core

├— Ego

├— Super-Ego

├— Memory Database

├— Configuration

└— Environment Adapter

Future implementations may distribute these components across multiple processes or machines without altering the architecture.

C.10 Interface Diagram

Every component exposes exactly one public interface.

Environment Interface

↓

Observation

↓

Causal Core

↓

Viability State

↓

Ego

↓

Behaviour Proposal

↓

Super-Ego

↓

Norm Evaluation

↓

Runtime

↓

Execution Result

↓

Memory

These interface objects constitute the common language of the architecture.

C.11 Architectural Constraints

Every compliant implementation shall satisfy the following structural constraints.

- Every architectural responsibility shall be implemented exactly once.
- No component shall perform the organisational responsibility of another component.

- Every communication path shall pass through an explicit interface.
- The Runtime shall coordinate every operational cycle.
- Every completed cycle shall generate exactly one Event.
- Every Event shall be stored permanently by Memory.

These constraints define the structural identity of Artificial Local Intelligence independently of programming language or implementation technology.

Appendix D

API Specification

D.1 Purpose

The architecture of Artificial Local Intelligence is defined through explicit communication between independent architectural components. Every interaction occurs through a stable interface rather than through unrestricted access to internal implementation details.

This appendix specifies the public API of the reference architecture.

The specification is intentionally independent of any programming language. The examples shown use Python-like syntax for readability only. Equivalent interfaces may be implemented in any suitable programming language.

D.2 Architectural Principles

Every public API follows the same principles.

- one responsibility per interface
- immutable input objects
- explicit return values
- no hidden side effects
- deterministic behaviour
- complete traceability

No interface modifies another architectural component directly.

All communication occurs through well-defined architectural objects.

D.3 Runtime API

The Runtime exposes the public lifecycle of an ALI.

class Runtime:

    def start()

    def stop()

    def pause()

    def resume()

```
    def execute()

    def build_event()
```

The Runtime never exposes behavioural planning or normative evaluation.

Its responsibility is coordination only.

D.4 Causal Core API

```
class CausalCore:

    def evaluate(
        observation
    ) -> ViabilityState

    def learn(
        event
    )
```

Input

Observation

Output

ViabilityState

The component performs no planning and no execution.

D.5 Ego API

```
class Ego:

    def generate(
        observation,
        viability,
        history
    ) -> BehaviourProposal[]

    def learn(
        event
```

)

Input

Observation

ViabilityState

Historical Context

Output

BehaviourProposal[]

The Ego returns one or more proposals.

It never returns executable actions.

D.6 Super-Ego API

class SuperEgo:

def evaluate(

proposals

) -> NormEvaluation

def learn(

event

)

Input

BehaviourProposal[]

Output

NormEvaluation

The Super-Ego never generates proposals.

D.7 Memory API

class Memory:

def retrieve(

observation

)

def store(

event

)

Input

Observation

Event

Output

Historical Context

Memory performs no interpretation.

It retrieves and stores architectural objects only.

D.8 Environment Interface API

class Environment:

def observe()

def execute(

action

)

Input

Approved Action

Output

Observation

ExecutionResult

The Environment Interface translates between architectural objects and the operational domain.

D.9 Observation Object

Observation

id

timestamp

source

payload

metadata

Purpose

Represents the external operational situation.

Immutable.

D.10 Viability State

ViabilityState

score

status

metrics

risk

timestamp

Purpose

Represents the operational judgement of the Causal Core.

D.11 Behaviour Proposal

BehaviourProposal

id

goal

actions

tools

confidence

reasoning

metadata

Purpose

Represents one candidate behaviour.

Multiple proposals may exist simultaneously.

D.12 Norm Evaluation

NormEvaluation

proposal_id

decision

violated_norms

warnings

explanation

confidence

Possible decisions

APPROVED

REJECTED

REVISE

D.13 Execution Result

ExecutionResult

status

duration

outputs

errors

resources

Purpose

Documents the observable consequences of execution.

D.14 Event

The Event is the central architectural object.

Event

id

timestamp

observation

viability

proposal

evaluation

execution

metadata

Every operational cycle generates exactly one Event.

D.15 Error Handling

Every public API returns structured exceptions.

Example

RuntimeError

MemoryError

NormViolation

EnvironmentFailure

ExecutionFailure

No component shall terminate the Runtime directly.

Recovery always remains under Runtime control.

D.16 Versioning

Every interface carries an architectural version.

Example

ALI API

Version 1.0

Future implementations may extend interfaces.

Breaking architectural changes require a new major version.

D.17 Compliance Rules

A compliant implementation shall satisfy the following API requirements.

- Every component exposes exactly one public interface.
- Internal implementation details remain encapsulated.
- Every interface accepts only architectural objects.
- Every interface returns architectural objects.
- No component accesses another component internally.
- Runtime coordination remains explicit.
- Every operational cycle produces one Event.

D.18 Summary

The API specification deliberately reflects the architectural philosophy developed throughout this book.

The interfaces are intentionally small.

The architectural objects are explicitly typed.

Responsibilities remain isolated.

Implementation technologies remain interchangeable.

Consequently, the API serves not merely as a programming interface but as the executable expression of the organisational principles that define Artificial Local Intelligence.

Appendix E

Configuration

E.1 Purpose

The architecture of Artificial Local Intelligence deliberately separates configuration from implementation.

The architecture defines organisational responsibilities.

The implementation realises these responsibilities in software.

The configuration determines how a particular ALI instance behaves within its operational domain.

Consequently, changing the configuration never changes the architecture.

This separation allows the same reference implementation to operate in completely different environments without modifying the source code.

E.2 Configuration Hierarchy

The reference implementation organises configuration into several independent files.

config/

├— runtime.yaml

├— models.yaml

├— norms.yaml

├— environment.yaml

├— memory.yaml

├— logging.yaml

└— security.yaml

Each file configures one architectural responsibility.

E.3 Runtime Configuration

Example

runtime:

cycle_time: 100 ms

max_parallel_tasks: 4

event_queue: 100

auto_recovery: true

shutdown_timeout: 30 s

The Runtime configuration determines scheduling behaviour only.

It never contains behavioural rules.

E.4 Ego Configuration

The Ego specifies which reasoning components are available.

Example

ego:

planner:

llm: GPT

symbolic_reasoner: enabled

tool_selection: enabled

retrieval: enabled

confidence_threshold: 0.75

Future reasoning engines can be substituted without changing the surrounding architecture.

E.5 Causal Core Configuration

The Causal Core defines the operational viability model.

Example

causal_core:

cpu_limit: 90 %

memory_limit: 80 %

network_timeout: 5 s

storage_limit: 85 %

minimum_viability: 0.60

Individual applications may define additional operational metrics.

E.6 Norm Configuration

Permanent constraints and adaptive norms are stored separately.

Example

permanent_norms:

no_data_loss: true

no_self_modification: true

privacy_required: true

architecture_integrity: true

Adaptive norms

adaptive_norms:

minimise_resources: weight 0.70

minimise_latency: weight 0.40

maximise_reliability: weight 0.90

minimise_external_calls: weight 0.50

The weights evolve during operation.

The permanent constraints never change.

E.7 Memory Configuration

Example

memory:

database: sqlite

compression: enabled

backup: daily

indexing: enabled

retention: permanent

The architecture assumes that Events are never deleted.

Archiving strategies may change.

Historical continuity may not.

E.8 Environment Configuration

The Environment Interface depends entirely upon the operational domain.

Example

environment:

filesystem: enabled

email: enabled

calendar: enabled

browser: enabled

local_documents: enabled

internet_access: optional

A robotic implementation would expose entirely different interfaces while preserving the same architecture.

E.9 Model Configuration

Reasoning models remain configurable.

Example

models:

planner:

provider: OpenAI

model: GPT-5.x

embeddings:

provider: local

speech:

enabled: false

The architecture deliberately avoids dependence upon specific model providers.

E.10 Logging Configuration

Architectural logging differs from technical debugging.

Example

logging:

events: enabled

api_calls: enabled

runtime: enabled

debugging: optional

metrics: enabled

Every Event is stored independently of technical logging.

E.11 Security Configuration

Example

security:

encryption: enabled

authentication: enabled

local_storage_only: false

external_execution: restricted

Security policies belong to configuration rather than architecture.

E.12 Configuration Lifecycle

Configuration is loaded once during startup.

Configuration

↓

Validation

↓

Runtime Initialisation

↓

Component Initialisation

↓

Operational Cycle

After startup, architectural responsibilities remain unchanged.

Only explicitly permitted configuration values may be modified during operation.

E.13 Configuration Validation

Before Runtime startup every configuration file is validated.

Validation checks include:

- syntax
- completeness
- consistency
- dependency resolution
- version compatibility

An ALI instance never begins execution with an invalid configuration.

E.14 Configuration Inheritance

Domain-specific systems inherit the reference configuration.

Example

Reference Configuration

↓

Scientific Assistant

↓

Medical Assistant

↓

Industrial Controller

↓

Robot

Only domain-specific values differ.

The architectural configuration remains identical.

E.15 Compliance

A compliant implementation shall satisfy the following configuration requirements.

- Architecture shall never be configured.
- Only operational parameters may be configured.
- Permanent norms shall remain immutable.
- Runtime sequencing shall not be configurable.
- Component responsibilities shall not be configurable.
- Configuration shall be validated before startup.

E.16 Summary

Configuration represents the operational adaptation layer of Artificial Local Intelligence.

It allows one architectural implementation to support many operational domains while preserving complete architectural integrity.

This separation between architecture, implementation and configuration is one of the central engineering principles of the ALI framework.

Appendix F

Reference Environment

F.1 Purpose

Artificial Local Intelligence is intended to operate within a clearly defined operational environment. Unlike general-purpose AI systems that attempt to access arbitrary resources, an ALI is configured for one specific domain. This domain defines the information sources, tools, communication channels and operational objectives available to the system.

The Environment therefore represents the operational world of the ALI.

The architecture itself remains unchanged.

Only the Environment Interface differs between implementations.

F.2 Environment Definition

The reference implementation assumes that every ALI instance operates inside exactly one Environment.

Conceptually,

Architecture

+

Environment

=

Artificial Local Intelligence

The architecture provides autonomous behaviour.

The Environment provides operational meaning.

F.3 Example Environment

A scientific research assistant may expose the following resources.

Environment

Documents

Scientific papers

PDF files

Reference database

Local filesystem

Calendar

Email

Browser

Python execution

Knowledge base

Every resource is accessed exclusively through the Environment Interface.

F.4 Environment Adapter

The Environment Interface is implemented through adapters.

Environment

│

├— File Adapter

├— Browser Adapter

├— Email Adapter

├— Calendar Adapter

├— Database Adapter

├— Python Adapter

└— API Adapter

Every adapter translates between the external system and the internal architectural objects.

F.5 Observation Generation

Adapters never perform reasoning.

Their responsibility is limited to producing Observations.

Example

Incoming Email

↓

Email Adapter

↓

Observation

or

PDF Document

↓

Document Adapter

↓

Observation

The Observation becomes the architectural representation of the external event.

F.6 Action Execution

Execution proceeds in the opposite direction.

Behaviour Proposal

↓

Approved Action

↓

Environment Adapter

↓

External System

The adapter performs the translation.

The architectural components remain independent of the operational domain.

F.7 Tool Registry

The reference implementation maintains a registry of available tools.

Example

Tool Registry

Search

Read Document

Write Document

Execute Python

Query Database

Send Email

Open Browser

Create Calendar Entry

The Ego selects tools.

The Runtime executes them.

F.8 Resource Availability

Every resource possesses an operational state.

Example

Filesystem

Available

Database

Available

Internet

Unavailable

Email

Available

Browser

Busy

These states contribute to the Viability State generated by the Causal Core.

F.9 Domain Independence

Changing the operational domain never changes the architecture.

For example,

Scientific Assistant

↓

Medical Assistant

requires replacing only

- document models,
- normative rules,
- operational objectives,
- environment adapters.

The Runtime, Causal Core, Ego, Super-Ego and Memory remain unchanged.

F.10 Local Operation

The reference implementation is designed primarily for local execution.

Typical installation

Local Computer

↓

Runtime

↓

Local Models

↓

Local Memory

↓

Local Documents

Cloud services may be integrated through adapters.

They never become architectural components.

F.11 Startup Sequence

Runtime startup proceeds as follows.

Load Configuration

↓

Load Environment

↓

Initialise Runtime

↓

Initialise Components

↓

Check Resources

↓

Create Initial Viability State

↓

Wait for Observation

The first operational cycle begins only after successful initialisation.

F.12 Shutdown Sequence

Shutdown is equally deterministic.

Stop New Observations

↓

Complete Active Cycle

↓

Store Final Event

↓

Flush Memory

↓

Close Adapters

↓

Terminate Runtime

No operational history is lost during shutdown.

F.13 Failure Recovery

If an external resource becomes unavailable,

Database Failure

↓

Environment Adapter

↓

Observation

↓

Causal Core

↓

Reduced Viability

↓

Ego

↓

Alternative Behaviour

The architecture continues operating whenever possible.

Failure therefore becomes part of ordinary autonomous behaviour rather than an exceptional condition.

F.14 Multiple Environments

Future implementations may support multiple Environments.

Example

Research Environment

Industrial Environment

Medical Environment

Educational Environment

Each Environment possesses its own adapters and configuration.

The architectural organisation remains identical.

F.15 Compliance

A compliant ALI Environment shall satisfy the following requirements.

- Every external resource shall be accessed through an Environment Adapter.
- Adapters shall never perform behavioural reasoning.
- Adapters shall produce architectural Observations.
- Execution shall occur exclusively through the Runtime.
- External systems shall never access architectural components directly.

F.16 Summary

The Environment defines the operational world in which an ALI exists.

It provides resources, observations and execution targets.

The architecture provides organisation.

By separating these two aspects, Artificial Local Intelligence allows the same architectural framework to operate across entirely different domains while preserving complete organisational consistency.

Appendix G

Validation

G.1 Purpose

The architecture presented in this book has not been developed as a purely theoretical construction. It emerged through an iterative engineering process in which the architecture evolved step by step, each stage introducing a single new organisational capability while preserving the coherence of the overall system.

The purpose of this appendix is to document that process.

The stages presented here should not be interpreted as versions of the software. They represent successive levels of architectural maturity. Each stage introduces exactly one essential architectural concept and demonstrates why that concept became necessary before the next stage could be reached.

Together, the nine stages reconstruct the emergence of the complete ALI architecture.

G.2 Stage 1

Reactive System

Objective

Construct a system capable of responding to external observations.

Architecture

Observation

↓

Behaviour

↓

Execution

Characteristics

- no operational state
- no history
- no learning
- no norms

- purely reactive behaviour

Limitation

Every observation is treated independently.

The system has no continuity.

G.3 Stage 2

Behaviour Generation

Objective

Separate behavioural planning from execution.

Architecture

Observation

↓

Planner

↓

Behaviour Proposal

↓

Execution

Improvement

Behaviour becomes explicit.

Multiple proposals become possible.

Remaining Limitation

The system cannot distinguish between acceptable and unacceptable behaviour.

G.4 Stage 3

Operational Viability

Objective

Introduce explicit operational self-regulation.

New Component

Causal Core

Architecture

Observation

↓

Causal Core

↓

Viability

↓

Planner

Improvement

Behaviour now depends upon the operational condition of the system.

Insight

Operational viability is independent of behavioural planning.

G.5 Stage 4

Normative Evaluation

Objective

Separate behavioural generation from behavioural judgement.

New Component

Super-Ego

Architecture

Behaviour Proposal

↓

Super-Ego

↓

Approved Behaviour

Improvement

Behaviour is evaluated independently.

Insight

Planning and evaluation represent different architectural responsibilities.

G.6 Stage 5

Historical Continuity

Objective

Preserve complete operational history.

New Component

Memory

Architecture

Observation

↓

Decision

↓

Execution

↓

Event

↓

Memory

Improvement

Development becomes possible.

Insight

Learning requires history.

History requires Events.

G.7 Stage 6

Runtime Coordination

Objective

Coordinate the operational cycle.

New Component

Runtime

Improvement

Deterministic execution.

Explicit sequencing.

Architectural transparency.

Insight

Coordination must remain independent of reasoning.

G.8 Stage 7

Independent Learning

Objective

Allow every architectural component to develop independently.

Improvement

Behavioural learning.

Normative learning.

Operational learning.

Historical accumulation.

Insight

Learning is distributed.

It is not one global optimisation process.

G.9 Stage 8

Stable Architecture

Objective

Separate organisational structure from behavioural adaptation.

Improvement

Architecture remains fixed.

Behaviour evolves continuously.

Insight

Only knowledge changes.

Responsibilities remain permanent.

G.10 Stage 9

Artificial Local Intelligence

Objective

Integrate all architectural responsibilities into one coherent system.

Final Architecture

Environment

↓

Runtime

↓

Causal Core

↓

Ego

↓

Super-Ego

↓

Memory

↓

Learning

↓

Next Cycle

Result

Continuous autonomous operation.

Operational viability.

Behaviour generation.

Normative regulation.

Historical continuity.

Independent learning.

Architectural stability.

G.11 Validation Matrix

Stage	New Capability	Architecture Complete
1	Reactive behaviour	✗
2	Behaviour generation	✗
3	Operational viability	✗
4	Normative evaluation	✗
5	Historical continuity	✗
6	Runtime coordination	✗
7	Independent learning	✗
8	Stable architecture	✗
9	Complete ALI	✓

G.12 Engineering Interpretation

The development described above illustrates an important engineering principle.

The architecture did not emerge by assembling unrelated software modules.

Instead, every architectural component was introduced because a limitation of the previous stage made further development impossible.

Operational viability became necessary because purely behavioural systems lacked self-regulation.

Normative evaluation became necessary because behavioural planning alone could not distinguish acceptable from unacceptable actions.

Memory became necessary because learning requires historical continuity.

Runtime coordination became necessary because independent architectural components require deterministic orchestration.

Each architectural responsibility therefore possesses an explicit engineering justification.

G.13 Conclusion

The nine validation stages demonstrate that Artificial Local Intelligence is not an arbitrary collection of architectural components.

Each component solves a specific organisational problem that emerged during the development of the architecture.

The complete ALI architecture therefore represents the cumulative result of a systematic engineering process rather than a single conceptual design decision.

Appendix H

ALI Compliance Specification

H.1 Purpose

This appendix defines the normative compliance requirements of Artificial Local Intelligence.

The preceding chapters described the architecture conceptually. The present appendix specifies the minimum requirements that an implementation shall satisfy in order to be considered compliant with the ALI architecture.

The specification intentionally follows the style of established software standards such as IEEE and ISO specifications.

Every requirement is uniquely identified.

Requirements are expressed independently of programming language, operating system or implementation technology.

H.2 Requirement Levels

The following terminology is used throughout this specification.

Shall

Defines a mandatory architectural requirement.

Should

Defines a recommended implementation practice.

May

Defines an optional extension.

Only "shall" requirements determine ALI compliance.

H.3 Architectural Requirements

ALI-REQ-001

The implementation **shall** provide exactly one Runtime responsible for coordinating the operational cycle.

ALI-REQ-002

The Runtime **shall not** perform behavioural generation.

ALI-REQ-003

The Runtime **shall not** perform normative evaluation.

ALI-REQ-004

The Runtime **shall not** perform operational viability assessment.

ALI-REQ-005

The implementation **shall** provide exactly one Causal Core.

ALI-REQ-006

The Causal Core **shall** evaluate operational viability.

ALI-REQ-007

The Causal Core **shall not** generate behavioural proposals.

ALI-REQ-008

The Causal Core **shall not** perform execution.

ALI-REQ-009

The implementation **shall** provide exactly one Ego.

ALI-REQ-010

The Ego **shall** generate one or more Behaviour Proposals.

ALI-REQ-011

The Ego **shall not** evaluate behavioural proposals.

ALI-REQ-012

The Ego **shall not** execute behaviour.

ALI-REQ-013

The implementation **shall** provide exactly one Super-Ego.

ALI-REQ-014

The Super-Ego **shall** evaluate Behaviour Proposals independently of the Ego.

ALI-REQ-015

The Super-Ego **shall not** generate behavioural proposals.

ALI-REQ-016

The Super-Ego **shall not** execute behaviour.

ALI-REQ-017

The implementation **shall** provide exactly one Memory component.

ALI-REQ-018

Memory **shall** preserve every completed Event.

ALI-REQ-019

Memory **shall not** modify stored Events.

ALI-REQ-020

Memory **shall** support historical retrieval.

H.4 Runtime Requirements

ALI-REQ-021

Every operational cycle **shall** begin with an Observation.

ALI-REQ-022

Operational viability **shall** be evaluated before behavioural planning.

ALI-REQ-023

Behavioural planning **shall** precede normative evaluation.

ALI-REQ-024

Normative approval **shall** precede execution.

ALI-REQ-025

Execution **shall** precede Event creation.

ALI-REQ-026

Event storage **shall** precede learning.

ALI-REQ-027

The Runtime **shall** coordinate this sequence deterministically.

H.5 Interface Requirements

ALI-REQ-028

Architectural components **shall** communicate exclusively through defined interfaces.

ALI-REQ-029

Components **shall not** access the internal state of other components directly.

ALI-REQ-030

Every interface **shall** exchange architectural objects.

ALI-REQ-031

Every architectural object **shall** possess a unique identifier.

H.6 Event Requirements

ALI-REQ-032

Exactly one Event **shall** be created for every completed operational cycle.

ALI-REQ-033

Every Event **shall** contain

- Observation
- Viability State
- Behaviour Proposal
- Norm Evaluation
- Execution Result
- Metadata

ALI-REQ-034

Events **shall** remain immutable after storage.

ALI-REQ-035

Events **shall** remain permanently retrievable.

H.7 Learning Requirements

ALI-REQ-036

Behavioural learning **shall** occur exclusively within the Ego.

ALI-REQ-037

Normative learning **shall** occur exclusively within the Super-Ego.

ALI-REQ-038

Operational learning **shall** occur exclusively within the Causal Core.

ALI-REQ-039

Memory **shall not** perform autonomous interpretation.

ALI-REQ-040

Learning **shall not** modify architectural responsibilities.

H.8 Architectural Stability

ALI-REQ-041

The implementation **shall** preserve the separation of architectural responsibilities.

ALI-REQ-042

Components **shall not** exchange organisational responsibilities during operation.

ALI-REQ-043

Architectural self-modification **shall not** occur.

ALI-REQ-044

Only operational knowledge **shall** change during learning.

H.9 Compliance Checklist

Requirement Group	Mandatory
Runtime	✓
Causal Core	✓
Ego	✓
Super-Ego	✓
Memory	✓
Environment Interface	✓
Runtime Sequence	✓

Requirement Group	Mandatory
Events	✓
Learning	✓
Architectural Stability	✓

H.10 Certification

An implementation may be described as **ALI compliant** if and only if all mandatory requirements defined in this appendix are satisfied.

Compliance is independent of:

- programming language,
- operating system,
- hardware platform,
- reasoning algorithm,
- language model,
- database,
- deployment architecture.

Compliance concerns organisational structure only.

H.11 Rationale

The purpose of this specification is not to restrict implementation creativity.

Its purpose is to preserve the architectural identity of Artificial Local Intelligence while allowing unlimited technological evolution.

Future implementations may become vastly more capable than the reference implementation presented in this book.

As long as they satisfy the architectural requirements defined in this appendix, they remain members of the ALI architecture family.

Appendix I

Glossary

Purpose

Artificial Local Intelligence introduces a number of architectural concepts whose meaning is precisely defined throughout this specification. The purpose of this glossary is to provide a concise reference for these terms. Unless explicitly stated otherwise, the definitions given here are normative.

The glossary is organised alphabetically.

Action

The executable form of an approved Behaviour Proposal.

An Action is produced only after successful normative evaluation and is executed exclusively through the Environment Interface.

Adaptive Norm

A normative rule whose priority may change through accumulated operational experience.

Adaptive norms regulate behaviour without changing the permanent architectural principles.

ALI

Artificial Local Intelligence.

An autonomous software architecture based on explicit separation of operational viability, behavioural generation, normative evaluation, historical continuity and runtime coordination.

Architecture

The permanent organisational structure of an ALI.

The architecture defines responsibilities.

It does not define implementation technologies.

Behaviour

The externally observable activity of an ALI resulting from an approved Behaviour Proposal.

Behaviour Proposal

A structured description of one possible course of action generated by the Ego.

A Behaviour Proposal is not executable until it has been approved by the Super-Ego.

Candidate Norm

A provisional norm proposed during normative development.

Candidate norms are evaluated through operational experience before becoming adaptive norms.

Causal Core

The architectural component responsible for evaluating operational viability.

The Causal Core performs neither behavioural planning nor normative evaluation.

Compliance

Conformity with the architectural requirements defined in Appendix H.

Compliance concerns organisational structure rather than implementation technology.

Environment

The operational domain within which an ALI functions.

Examples include scientific research, industrial automation, document management or robotics.

Environment Interface

The architectural component responsible for communication between the ALI and its operational environment.

Event

The complete historical record of one operational cycle.

Every Event contains

- Observation
- Viability State
- Behaviour Proposal
- Norm Evaluation
- Execution Result
- Metadata

Execution Result

The architectural object describing the observable outcome of an executed Action.

Ego

The architectural component responsible for behavioural generation.

The Ego proposes behaviour.

It never evaluates or executes behaviour.

Historical Continuity

The preservation of complete operational history through Events stored in Memory.

Historical continuity enables learning and developmental reconstruction.

Learning

The gradual modification of operational knowledge through accumulated experience.

Learning changes behaviour.

It does not change architectural responsibilities.

Memory

The architectural component responsible for preserving Events and providing historical retrieval.

Memory performs no behavioural reasoning.

Metadata

Auxiliary information attached to an architectural object.

Typical metadata include timestamps, identifiers and execution statistics.

Norm

A rule governing the acceptability of behaviour.

Norms may be permanent or adaptive.

Norm Evaluation

The architectural object produced by the Super-Ego after evaluating Behaviour Proposals.

Possible outcomes include approval, rejection and revision.

Observation

The architectural representation of the current operational situation.

Observations are produced exclusively by the Environment Interface.

Operational Cycle

One complete sequence of

Observation

↓

Viability Evaluation

↓

Behaviour Generation

↓

Norm Evaluation

↓

Execution

↓

Event Creation

↓

Learning

Operational Viability

The current capability of an ALI to continue autonomous operation.

Operational viability is evaluated by the Causal Core.

Permanent Constraint

A norm that never changes during operation.

Permanent constraints define the architectural identity of an ALI.

Proposal

See Behaviour Proposal.

Reference Implementation

The software implementation presented in this specification demonstrating one compliant realisation of the ALI architecture.

Runtime

The architectural component responsible for coordinating the operational cycle.

The Runtime performs no behavioural reasoning.

Stability

The property that architectural responsibilities remain invariant throughout the operational lifetime of an ALI.

Super-Ego

The architectural component responsible for normative evaluation.

The Super-Ego evaluates Behaviour Proposals independently of the Ego.

Tool

An external capability available through the Environment Interface.

Examples include databases, file systems, browsers and computational services.

Validation

The process of demonstrating that an implementation satisfies the architectural principles of Artificial Local Intelligence.

Validation includes architectural, functional and behavioural evaluation.

Viability State

The architectural object representing the operational judgement produced by the Causal Core.

The Viability State forms one of the inputs to behavioural planning.

Version

The architectural specification identifier associated with an ALI implementation.

Versioning applies to the architecture rather than to individual implementation technologies.

Summary

The terms defined in this glossary constitute the normative vocabulary of Artificial Local Intelligence.

Throughout this specification, these terms are used with the meanings defined here and should not be interpreted according to their everyday usage where such interpretations differ from the architectural definitions.

Appendix J

Complete Reference Architecture

J.1 Purpose

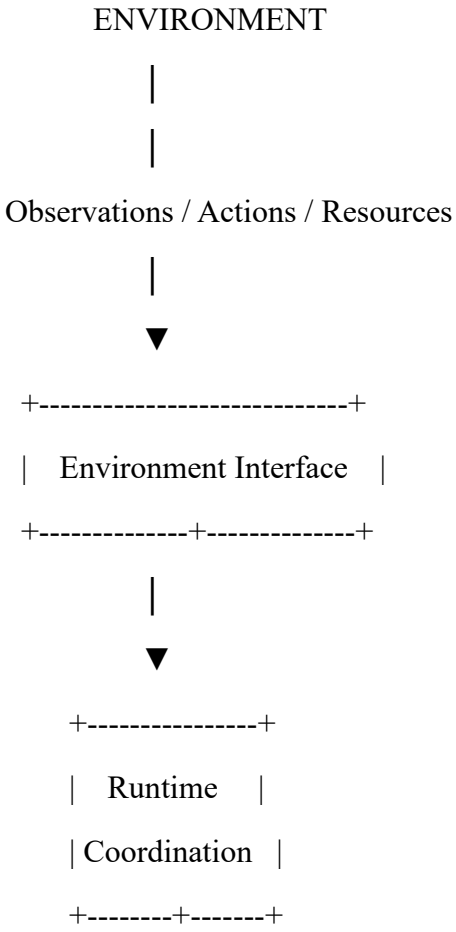
This appendix presents the complete reference architecture of Artificial Local Intelligence on a single page.

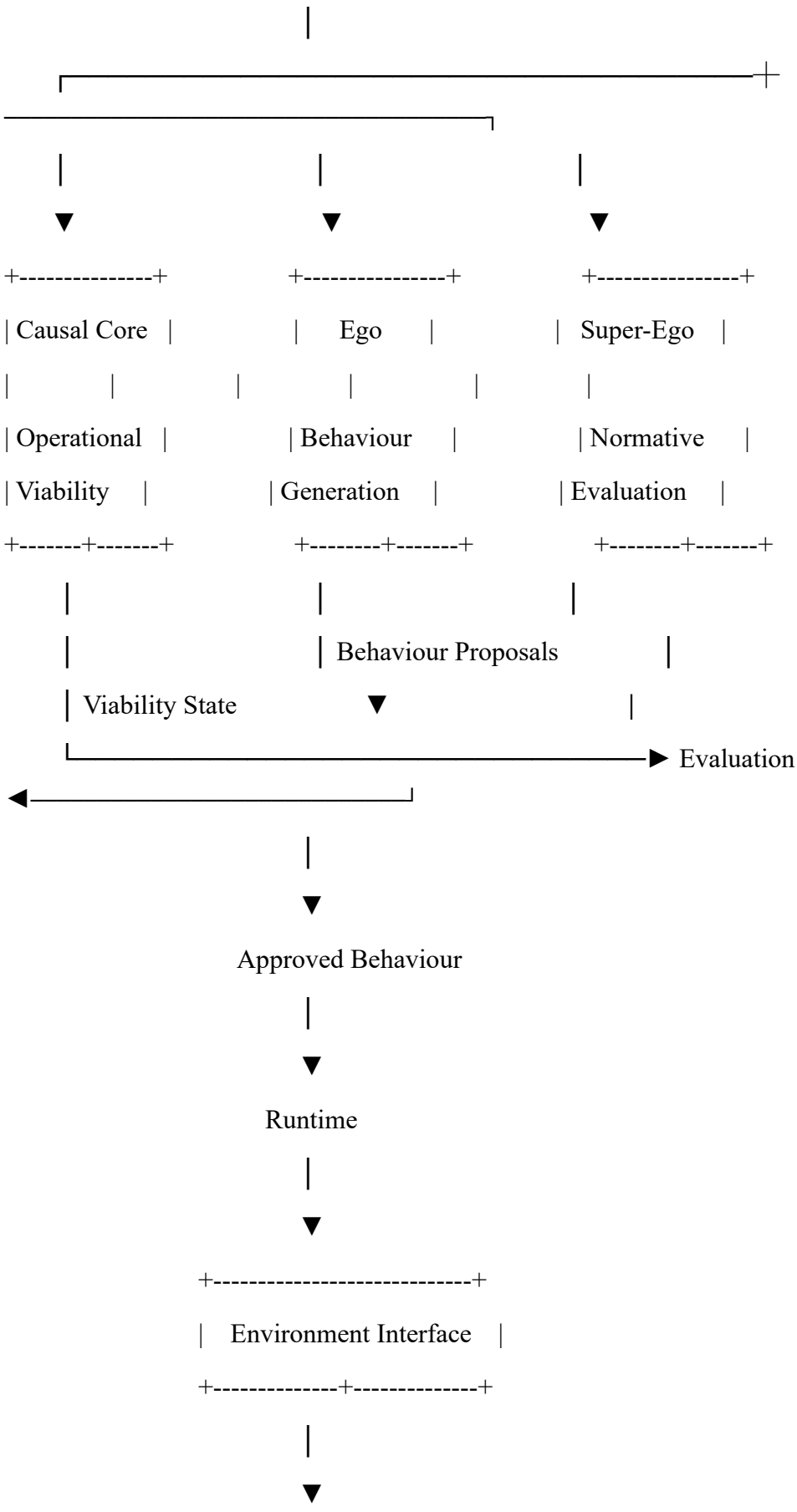
The preceding chapters introduced the architecture incrementally. Individual components, interfaces, data structures and operational cycles were described separately in order to explain their respective responsibilities. The purpose of this appendix is to integrate these elements into one coherent architectural model.

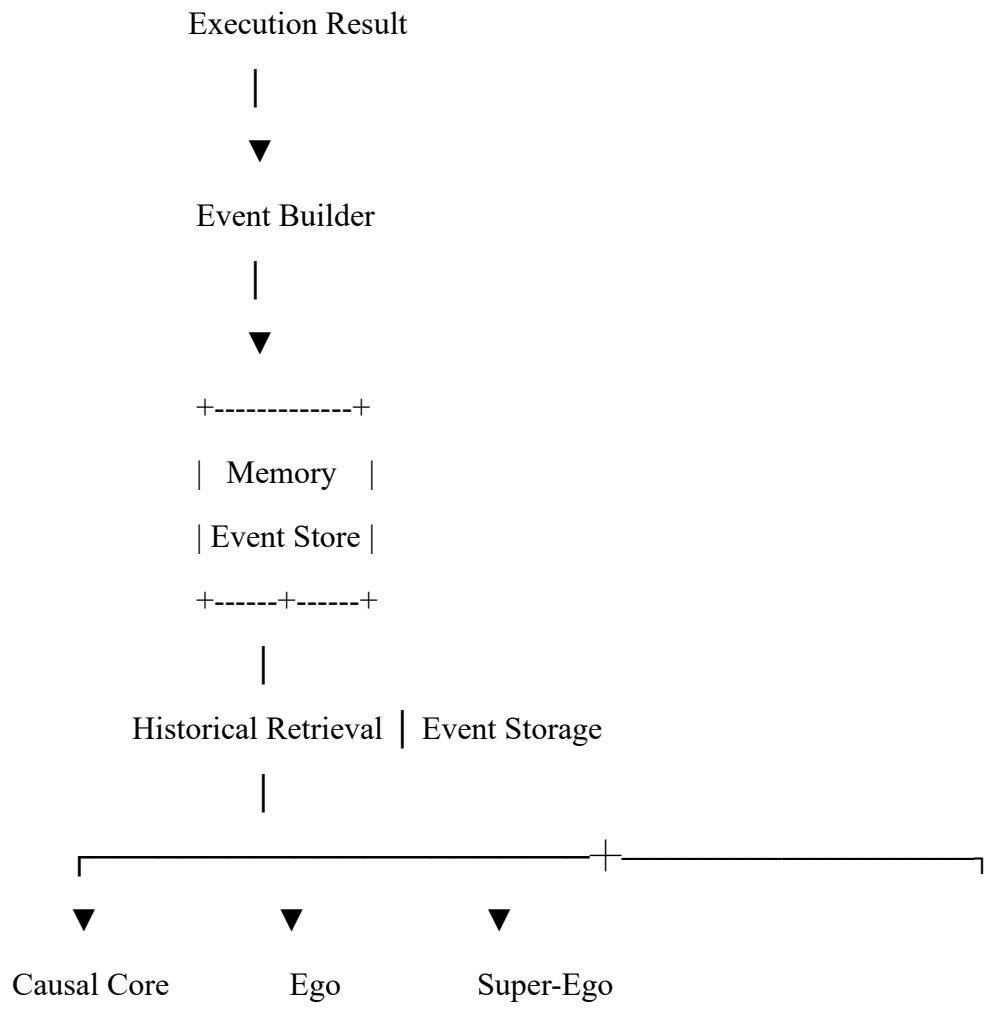
The diagram presented here should be regarded as the canonical representation of Artificial Local Intelligence.

Every compliant implementation should be recognisable as an instantiation of this architecture.

J.2 Complete Architecture




Causal Core
Operational Viability
Ego
Behaviour Generation
Super-Ego
Normative Evaluation
Behaviour Proposals
Viability State
Evaluation
Approved Behaviour
Runtime
Environment Interface



J.3 Architectural Layers

The complete architecture consists of five conceptual layers.

Layer	Responsibility
Environment	External operational domain
Coordination	Runtime
Cognitive Components	Causal Core, Ego, Super-Ego
Historical Layer	Memory
Learning Layer	Distributed adaptation

The layers remain logically distinct throughout the lifetime of the system.

J.4 Information Flow

The architecture processes information in one direction.

Observation

↓

Operational Viability

↓

Behaviour Generation

↓

Normative Evaluation

↓

Execution

↓

Historical Event

↓

Learning

↓

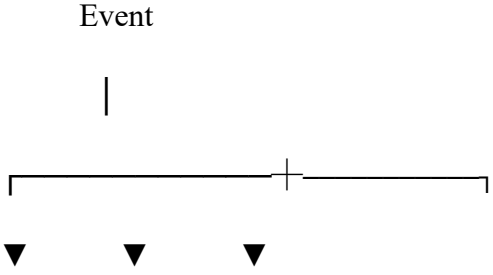
Next Observation

No shortcuts exist.

Every autonomous decision follows this sequence.

J.5 Learning Flow

Learning occurs after the Event has been stored.



Causal Core Ego Super-Ego

Operational Behaviour Normative

Learning Learning Learning

Memory itself performs no learning.

It provides the historical foundation upon which learning occurs.

J.6 Stable Architecture

Only one aspect of the architecture changes during operation.

Architecture

UNCHANGED

↓

Knowledge

Changes Continuously

↓

Behaviour

Changes Continuously

Responsibilities remain fixed.

Knowledge evolves.

J.7 Organisational Responsibilities

The architecture deliberately assigns every organisational responsibility exactly once.

Responsibility	**Component**
Operational viability	Causal Core
Behaviour generation	Ego
Normative evaluation	Super-Ego
Runtime coordination	Runtime
Historical continuity	Memory
Environment interaction	Environment Interface

No responsibility appears twice.

No responsibility is omitted.

J.8 Architectural Principles

The complete ALI architecture is founded on six principles.

Explicit separation of responsibilities

Every organisational function is assigned to one architectural component.

Continuous operational viability

Autonomous behaviour always depends upon the operational condition of the system.

Independent normative regulation

Behaviour is generated independently from its evaluation.

Permanent historical continuity

Every operational cycle becomes part of the developmental history.

Stable organisational architecture

Learning modifies knowledge, never responsibilities.

Technology independence

Algorithms may evolve.

The architecture remains invariant.

J.9 Canonical Definition

An implementation may be described as an **Artificial Local Intelligence** if it satisfies all of the following conditions.

It implements the architectural responsibilities defined in this specification.

It preserves the operational sequence described in the Runtime.

It stores complete Events.

It separates behavioural generation from normative evaluation.

It maintains architectural stability throughout learning.

It communicates exclusively through explicit architectural interfaces.

J.10 Final Statement

Artificial Local Intelligence is not defined by a programming language, a reasoning algorithm, a language model or a software framework.

It is defined by its architecture.

The architecture specifies how autonomous intelligence is organised, how its responsibilities are separated, how its behaviour develops through experience and how that development remains permanently reconstructable.

Every compliant implementation is therefore an instance of the same architectural idea, regardless of the computational technologies used to realise it.