MOSAIC: MODULAR ORCHESTRATION OF SPECIALISED AI COMPONENTS

Nabil Islam
Independent Researcher
w3nabil@gmail.com

November 4, 2026

ABSTRACT

Large frontier language models achieve strong general-purpose performance, but they are expensive to run. A natural question is whether a system of smaller, specialised components, combined with structured task decomposition, can match the accuracy of a frontier model on a representative set of tasks, while using frontier-scale inference only rarely. We propose MOSAIC, an architecture in which a query is resolved through a strict hierarchy of **hardcoded orchestration logic**, not through routing between pre-trained models of different sizes. Each level of the hierarchy receives a task specification, applies fixed control logic to decide whether and how to narrow it, and calls a specialised model directly once a sub-task is clearly scoped. We describe three mechanisms that stack together and are intended to reduce inference cost compared with a single dense model: model-scale substitution, early termination, and cumulative partial resolution. We state this as a single efficiency hypothesis (H1). We describe the architecture, state the hypothesis explicitly, formalise the structural guarantees of the hierarchy, and propose an evaluation plan to test the hypothesis directly. [1]

***K*eywords** hierarchical task decomposition, multi-agent orchestration, compute-efficient inference, small language models, bounded retry termination.

1 Introduction

1.1 Motivation

The main trend in large language model development has been to increase the number of parameters in order to gain general capability. This produces models that are highly capable but expensive to serve, whatever the difficulty of the query. This raises a natural question: for a given query, is the full capacity of a frontier-scale model really necessary? Or could several smaller models, supported by the right structure, give an equal or better answer at lower cost?

One way to approach this question is as a model-routing problem: a central orchestrator selects among several pre-trained specialised models based on domain, and a frontier model is kept as a limited escalation path. This framing treats routing accuracy (picking the right model) as the main mechanism of interest. It also treats capability gaps as something to measure (through the escalation rate) but not to explain.

Our work starts from a different point. The components that perform routing and decomposition are not models themselves. They are hardcoded control logic. They receive a task specification, decide how to narrow or split it, and call a model only when a sub-task is concrete enough to execute. This is not only a change of wording; it changes what the hierarchy is for. If the levels of the hierarchy were models of different capability, the hierarchy would work as a capability ladder: hard tasks go up, and easy tasks go down. Because the levels are orchestration logic, the hierarchy is better understood as a **decomposition pipeline**. Each level resolves one layer of ambiguity or scope and then passes a narrower task to the next level.

This difference motivates the central hypothesis of this report, which we present in Section 1.2.

[1] No experiments have been run yet. All claims in this report are hypotheses to be tested, not results.

1.2 Central Hypothesis

Our work is built around one central hypothesis. We state it here explicitly because the rest of the paper, and the proposed evaluation plan, are designed to test it.

H1 (Efficiency hypothesis): A system composed of several specialised models, combined with structured decomposition and selective frontier-model escalation, can achieve task performance comparable to a single large dense frontier-scale model, across a representative task distribution, while escalating to frontier-scale inference on a small minority of queries.

Section 4 describes the architectural mechanisms by which the design is intended to satisfy H1: model-scale substitution, early termination, and cumulative partial resolution. Together, they form a single governing efficiency principle. Section 8 proposes an evaluation to test H1 directly.

1.3 Contributions

Our work makes the following contributions:

1. An architecture description in which the hierarchy is made of hardcoded orchestration logic that performs task decomposition. It is not a routing table between pre-trained models of different sizes.
2. A formal statement of two structural guarantees that the hierarchy provides by construction: a **topological non-looping guarantee** (strict downward-only delegation) and a **retry-bounded termination guarantee** at the top-level control loop. We also explain why the two are complementary and not redundant.
3. A single governing efficiency principle (Section 4). It combines three stacking cost-reduction mechanisms built into the architecture, and it is the concrete basis for the efficiency hypothesis (H1).
4. An evaluation plan that tests H1 directly against a frontier-model baseline. It includes a proposed metric, call-depth distribution, that is specific to the internal mechanisms of the architecture.

1.4 Report Structure

Section 2 reviews related work. Section 3 describes the MOSAIC architecture: the delegation hierarchy, specialised models, corpora, the calling-permission structure between components, a comparison with existing approaches, and the difference between tools and assistants. Section 4 states MOSAIC's efficiency principle, which combines three stacking mechanisms, as the concrete basis for H1. Section 5 lists the capabilities that the specialised models and the Frontier Model should have. Section 6 states the structural guarantees of the architecture. Section 7 explains MOSAIC's position on confidence estimation. Section 8 proposes an evaluation plan for H1. Section 9 discusses limitations. Section 10 concludes. The appendices give a diagram reference, a glossary, a worked example, the eAI conjecture, and an illustrative estimate of deployment footprint and cost.

2 Background & Related Studies

Our work was developed while closely observing the following work.

2.1 Mixture-of-Experts Architectures

Mixture-of-experts (MoE) approaches send different inputs to different subsets of model parameters, instead of activating a full dense network for every input [1, 2]. MOSAIC shares the routing principle, but it differs in granularity and in kind. MoE routing selects among learned sub-networks inside one model. MOSAIC's hierarchy selects how to **decompose a task**, and only at the leaves does it select a model to call. The comparison with MoE is therefore looser than it may first appear: MOSAIC's higher levels do not choose between experts. They produce the inputs that make an expert choice well defined.

A related family, early-exit methods, reduces computation by allowing inference to stop at an intermediate layer of a single network once an exit criterion is met [3]. MOSAIC's early termination (Section 4) has a similar intent, but it works at the level of an orchestration hierarchy across separate models, not inside the layers of one network.

2.2 Agentic and Multi-Agent LLM Systems

A growing body of work uses LLMs as orchestrating agents that call tools, other models, or sub-agents [4, 5]. MOSAIC's assistant hierarchy fits inside this paradigm, with one difference that we want to state clearly. Most agentic systems treat the orchestrator as a reasoning model. MOSAIC's orchestration layers are hardcoded logic. They call reasoning models, but they do not reason themselves in the sense of open-ended judgment. The reasoning happens inside the model calls that each level makes, not in the control flow that connects them. Whether this distinction is already recognised in the literature, or whether most "orchestrator" systems blur it, is a question we investigate.

2.2.1 Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) combines generation with retrieval of external knowledge, so that the generator is conditioned on relevant retrieved passages [6]. MOSAIC's Corpora component (Section 3.5) fits this standard pattern directly. An existing RAG retriever and index can be placed inside Corpora almost without change, because Corpora is defined as an unrestricted, finite, and findable data store that any level can access (Section 3.7 discusses this correspondence). However, the usual single-pass retrieve-then-generate loop of RAG is a narrower mechanism than MOSAIC's whole control structure. MOSAIC decides when and at which level retrieval happens, as part of a broader decomposition process. In RAG, retrieval is the whole mechanism.

2.3 Ensemble Methods

Combining several models and aggregating their outputs is a long-standing technique to improve predictive performance and robustness [7]. MOSAIC's fan-out to several parallel sub-orchestrators at one level (Section 3.3) looks similar to ensembling at first. However, the intent is decomposition by domain (for example, one branch handles the linear-algebra sub-problem and another handles the vector sub-problem). It is not independent attempts at the same sub-problem with aggregated voting.

2.4 Model Cascades and Routing for Cost Reduction

LLM cascades reduce inference cost by querying models in order of increasing size, and they stop once an answer is judged acceptable. FrugalGPT is the best-known example. It combines prompt adaptation, completion caching, and an LLM cascade [8]. It learns when the answer of a cheaper model can be accepted, so that a more expensive model is called only when needed. RouteLLM treats the same goal as a learned routing problem. It trains a router on preference data to decide, for each query, whether a strong model or a weak model should answer [9].

MOSAIC shares the goal of these systems: to reduce cost while keeping answer quality. H1 can be read as a claim in the same family. MOSAIC differs in *what* is selected and in *what does the selecting*. Cascades and learned routers choose among models of different sizes for the whole query, and the selection signal is usually a learned scorer or a quality estimate produced by a model. MOSAIC's levels do not choose one model for the whole query. They decompose the query and select specialised models for the resulting sub-tasks. They use hardcoded control logic, not a learned scorer.

This has a consequence that we state carefully. The stopping decision of a cascade depends on an estimate of answer quality. This is the kind of signal that Section 7 argues is unreliable as a gate. MOSAIC's early termination depends instead on whether the task specification has been satisfied at a given level (Section 3.3). Whether this replacement is an advantage is an empirical question. Cascade and routing systems are the natural baselines to test it (Section 8).

2.5 Orchestrated Multi-Model Systems

HuggingGPT uses a language model as a controller. The controller parses a request into sub-tasks, selects specialised models for each one, runs them, and combines the results [10]. This is the closest existing architecture to MOSAIC's decompose-then-dispatch structure. The main difference is the role of the controller. In HuggingGPT, the planning and selection steps are done by an LLM, so the orchestrator is itself a reasoning model. Its planning can make errors, loop, or produce a badly formed plan. MOSAIC instead makes each orchestration level hardcoded logic with strictly downward delegation. This gives the topological guarantee of Section 6.1 by construction. The cost of this choice is flexibility: an LLM controller can plan for almost any request, but hardcoded logic must anticipate the kinds of tasks it will meet. Section 9 lists this as a risk to H1.

Mixture-of-Agents stacks several LLMs in layers, so that the outputs of one layer become the context of the next [11]. This improves answer quality through aggregation. Its goal is quality, not cost, and it calls several full models for each query. It is therefore complementary to H1, and in cost terms it goes in the opposite direction. We mention it as a related layered multi-model design, not as a baseline.

3 Architecture

The MOSAIC architecture is based on the eAI conjecture, which is stated formally in Appendix D.

3.1 Overview

3.1.1 Ideal MOSAIC Components

An ideal MOSAIC architecture contains five core components:

1. Frontier Model
2. Assistants
3. Specialised Model
4. Corpora
5. Tools

Additional sub-components are not listed here, because they may be introduced as development requires.

3.1.2 Ideal MOSAIC Routing

An ideal MOSAIC routing can be described in six steps:

1. The user passes a question to the Frontier Model. If the Frontier Model cannot answer it with its own knowledge, it passes the query to the Assistants.
2. An Assistant receives a task specification or requirements (for example, a "param") from the level above. For the top-level orchestrator, the param comes from the Frontier Model. It is passed through an XML call or another method.
3. The Assistant applies fixed control logic to decide whether this param needs further decomposition. The decision depends on the signal of the specialised model (for example, "resolved / needs_decomposition / out_of_scope") that is given in its output. If decomposition is needed, the logic decides how to split the query, and which lower-level assistant(s) to call with which narrower query or queries.
4. When its own logic requires it, the Assistant calls a specialised model directly. This is not delegation to a lower level. It is part of the Assistant's own hardcoded sequence of steps.
5. The Assistant collects the results (from lower-level delegation, direct model calls, corpora, or all of them) and packages them into a response for the level above.
6. The result is forwarded to the Frontier Model. The Frontier Model checks two things separately: whether this round produced usable output at all, and whether that output is enough to satisfy the original query. If not, it calls another assistant, as long as the maximum limits have not been exceeded. Two independent bounded budgets control this step. The first budget limits repeated attempts to obtain usable output from a single assistant before another assistant is used instead. The second budget limits how many unsatisfying rounds the Frontier Model will accept before it produces a response anyway, even if it is not fully satisfied. This makes sure the loop ends even when a fully satisfying answer is never obtained (Section 6.2).

Internal failure handling *inside* a single assistant call is out of scope for this routing loop. This includes timeouts, exhaustion of a specific model backend, and retry-with-backoff for one model call. Such a failure appears here only as an unsuccessful call, the same as any other empty result. A separate design layer, which is not specified here, governs it.

Model and corpora access is available at every level of assistant. It is not reserved for the lowest level. What differs between levels is not *whether* a level can call a specialised model. What differs is the *shape of the conversation* and *which specialised model* the hardcoded logic of that level uses. This shape depends on how narrow the param is when it reaches that level (Section 3.3.2 explains this granularity argument in detail). An ideal approach can be to assign different model or corpora selections to lower levels.

3.2 Frontier Model

In a MOSAIC system, the Frontier Model is not the "reasoning" or "problem-solving" model. It should be a general model that can communicate and answer simple or basic queries. It should also know about the assistants: how to call them and how to work with their responses. Section 5.2 discusses the capabilities of the Frontier Model.

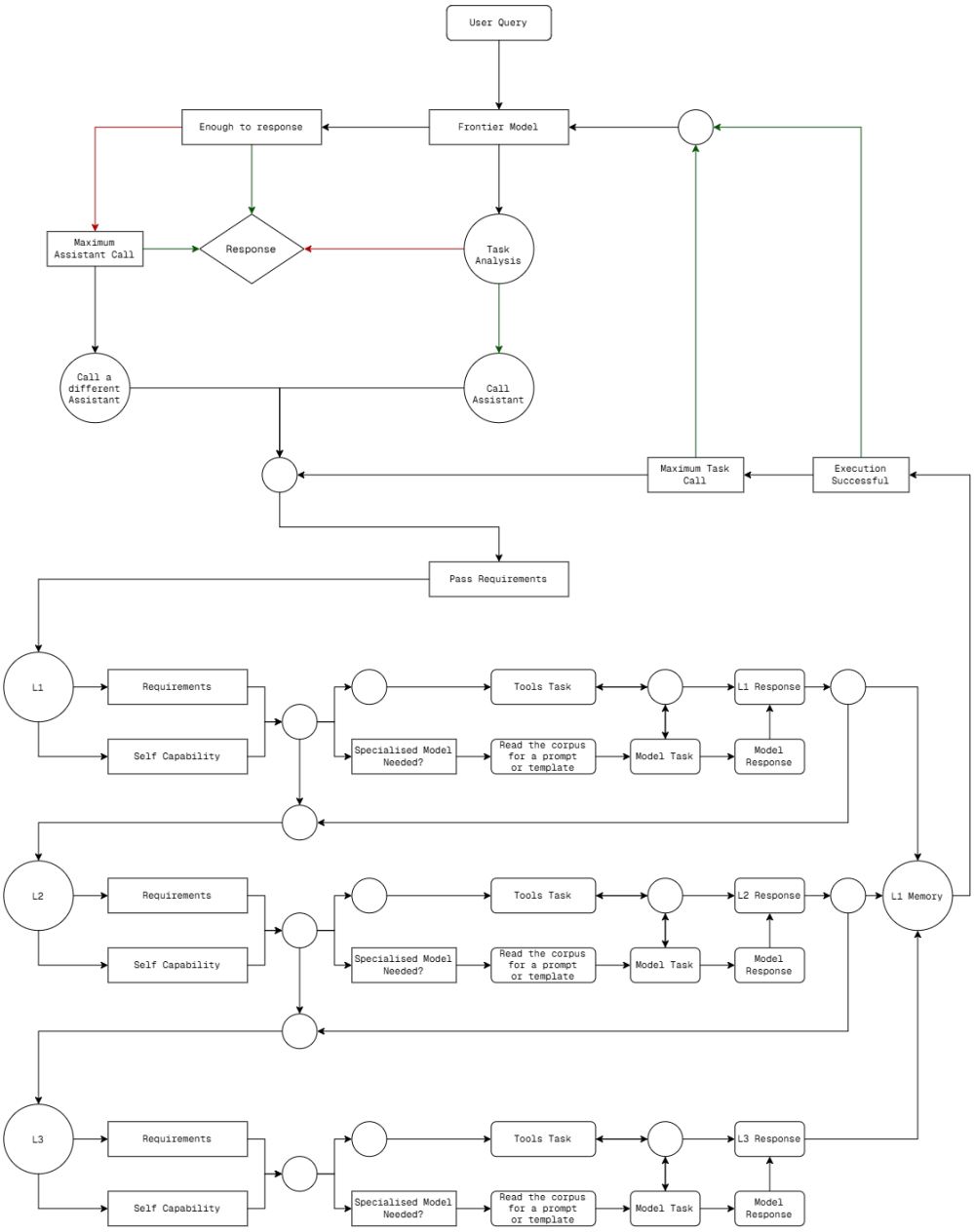


Figure 1: An ideal MOSAIC workflow/algorithm. The green and red lines show the true and false paths. “Maximum Task Call” and “Maximum Assistant Call” are the two retry budgets of Section 6.2. “Enough to response” is the satisfaction check of the Frontier Model, and it is not a budget. L1 Memory stores the responses of L1, L2, and L3 (Section 3.3.1).

3.3 The Delegation Hierarchy (Assistants)

Assistants are not models. They are hardcoded logic that helps the output to follow a strict chain, and it also reduces the risk of infinite loops. MOSAIC's orchestration layer (the assistants) is organised as a strict hierarchy. It has at least one level, and we recommend at least three. The maximum can be as large as needed to complete the task. The number of levels S can be any positive finite integer, $S \in \{1, 2, 3, \dots\}$. The main work of this layer is to:

1. Build a good pattern, and pass data to and retrieve data from a specialised model.
2. Split the load of a task equally, when the query is challenging or hard to compute.
3. Break the task into several pieces to get an accurate answer.
4. Call the corpora for background data that a specialised model needs.
5. Call tools, if needed, to provide background data.

An ideal MOSAIC system should contain at least three levels. They can be:

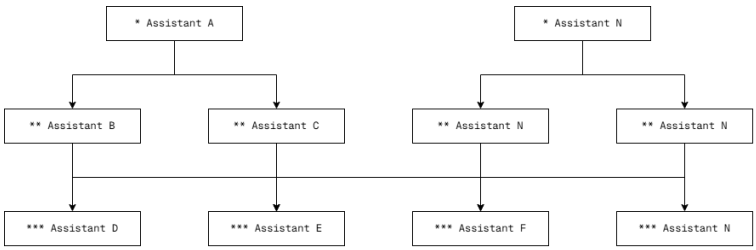


Figure 2: Assistant tree hierarchy. Asterisks show the level.

- **Level 1 (L1)**: L1 receives the broadest, least-scoped param from the Frontier Model. Its hardcoded logic, supported by a direct model call where needed, identifies the general domain(s) that the query covers. It then decides how many L2 assistants to call, and which ones. L1 may call more than one L2 in parallel when a task covers multiple domains. This follows the load-splitting and task-breaking purpose stated at the start of this section.
- **Level 2 (L2)**: L2 receives a domain-scoped param from L1 (for example, "the linear-algebra part of this problem"). Its hardcoded logic resolves ambiguity inside the domain. It identifies what is known, what is unknown, and what values an ambiguous symbol could take inside this domain. Then it decides how to split the domain-scoped task into fully atomic sub-tasks for L3.
- **Level 3 (L3)**: L3 receives a fully atomic, executable param from L2 (for example, "apply Method X to this specific sub-expression"). L3 is forced to answer: it does not delegate further. Its hardcoded logic has a direct, execution-shaped conversation with a specialised model. L3 also has a Self Capability check, but this check only allows L3 to return a failure signal (for example, "out_of_scope"). It cannot delegate. The model carries out the specified step and, where the design requires it, L3 can run a self-verification pass (for example, "Is this correct? Return true or false, then give the reasoning.").

Delegation is strictly downward. An assistant at level n may call assistants at level $n + 1$ or narrower. It may never call a level above itself. This is a structural property of the hierarchy, not a convention that each assistant is expected to follow. It is the basis for the topological guarantee in Section 6.1.

Note that any assistant at any level must be able to perform at least one task from the requirements. There is no mandatory loop that forces the system to go through the whole delegation chain once a requirement is fulfilled at some level n. In other words, if a task is solved at level 1, and two more levels exist, the following levels are ignored. L1 Memory (also called Assistant Memory, see Section 3.3.1) tracks how much progress a level has made and whether the result is ready to be sent to the Frontier Model.

3.3.1 L1 Memory

L1 Memory, also called Assistant Memory, is a sub-component of the assistant loop. It stores the log, progress, activities, and results of the task that each assistant performs. The loop that L1 controls is finite (Section 6.2), so L1 Memory can hold all the data of that loop. If the loop ends at L2, the data of L1 and L2 is passed to L1 Memory. If the loop ends at L3, the data of L1, L2, and L3 is passed to L1 Memory. Figure 1 shows that the response of each assistant (L1, L2, and

L3) is also written to L1 Memory. This is a write to a shared store, not a call to a higher-level assistant, so it does not break the strict downward delegation of Section 3.3.

3.3.2 Why the Bottom Level Never Needs to Escalate

A natural question is what prevents a bottom-level assistant from meeting a task that is so difficult that it needs to ask a broader, decomposition-shaped question to a model. Such a question would be outside its execution-shaped role. Two independent factors rule this out:

- **Structurally**: the topology does not allow the bottom level to call an upper level at all (Section 3.3). Even if such a need appeared, there is no path to act on it.
- **By construction of the query it holds**: when a query reaches the bottom level, the upper levels must have already done all the necessary decomposition and disambiguation. The bottom level never receives a param that is broad enough to make a decomposition-shaped question meaningful. There is nothing left to break down.

These two factors are not redundant safeguards for the same failure. They answer different questions. The topological constraint guarantees that L3 *cannot* escalate upward, even in theory. The granularity argument explains why, even without that constraint, the *need* to escalate would not appear when L1 and L2 work correctly. Stating both is useful because it shows that the architecture does not simply forbid a failure mode by decree. The failure mode is naturally absent from how the hierarchy is meant to work.

3.4 Specialised Models

Specialised models are models that are moderately or heavily trained on data from one specific domain or sub-domain. A large frontier model has a large weight to run, and it contains data from thousands of domains and sub-domains. Specialised models are relatively small in weight. They avoid multi-domain hallucination, because they are never trained on anything outside one specific domain.

For this reason, MOSAIC intends to use several small specialised models, which act like the neurons of a human brain, so that the assistant level can work properly and contribute to H1. Specialised models are experts in their field. This makes the call and the response accurate without needing to trigger a large model.

In a usual scenario, a query contains one specific question. Using the full capacity of a large model for it is expensive, and it also carries a risk of hallucination. MOSAIC instead calls one specialised model for each domain, and it assumes a medium-sized (in parameters) Frontier Model.

3.4.1 Comparison of a MOSAIC Specialised Model and a General Large Model

Table 1 compares a domain-specific specialised model with a single large, general-purpose model. It illustrates the basis for the model-scale substitution mechanism (Section 4). Figures marked *(illustrative)* are example values that follow the design intent of the architecture (Section 3.4). They are not fixed specifications. Figures marked *(hypothesis, not yet measured)* are expectations that the evaluation plan of this report (Section 8) is designed to test. They are not established results.

3.5 Corpora

MOSAIC recommends using one corpus, or dividing the corpora into several branches. A MOSAIC system fully accepts corpora, as long as they are finite and findable by any method. A single corpus may introduce hallucination, so we recommend using multiple corpora, one for each domain. A corpus is not restricted to one type of data. It may contain prompts, reasoning, judgements, cache data, and actual data of many kinds. The assistant architecture decides whether JSON, JSONL, CSV, or plain text is the best format for a corpus. MOSAIC does not provide any assistants by default. This gives developers more flexibility to change how an assistant parses data, without changing the structure of a corpus.

3.6 Component Calling (Communication) Rule

Table 2 states, for each MOSAIC component, which other components it may call. **Any** means unrestricted access. A named target (for example, *L2 Assistant*) means that the component may call that one target and no other. **None** means that calling is not permitted. An em dash means the relation of a component to itself, which does not apply. Also, an L1

Table 1: Specialised model versus large model, compared by capability, resource footprint, and expected cost and reliability.

Task	**Specialised Model**	**Large Model**
Can solve a domain-specific query	Yes	Yes
Can solve a multi-domain query alone, without orchestration	No	Yes
Minimum compute needed to run the full model	CPU	GPU
Model weight *(illustrative)*	Can be as low as 0.5B parameters	Can be as low as 27B parameters
Model size on disk *(illustrative, assuming FP16 precision: about 2 GB per billion parameters)*	Can be as low as 1 GB	Can be as low as 54 GB
Relative inference cost for an in-domain query *(hypothesis, not yet measured)*	Low (only the domain-relevant parameter set is activated)	High (the full parameter set is activated, whatever the difficulty of the query)
Time to load model weights into memory per switch *(hypothesis, not yet measured)*	Low per model, but it may be paid many times if a query calls several specialised models across L1/L2 branches	Paid once; one loaded large model serves every later query without reloading

assistant can pass a response to the Frontier Model without calling it, because the Frontier Model already expects a response with the call.

Table 2: Calling permissions across MOSAIC components.

Component	**Assistants**	**Tools**	**Corpora**	**Specialised Model**	**Frontier Model**
Frontier Model	*L1 Assistant*	**Any**	**Any**	None	–
L1 Assistant	*L2 Assistant(s)*	**Any**	**Any**	**Any**	None
L2 Assistant	*L3 Assistant(s)*	**Any**	**Any**	**Any**	None
L3 Assistant	None	**Any**	**Any**	**Any**	None
Tools	None	–	**Any**	None	None
Corpora	None	None	–	None	None
Specialised Model	None	**Any**	**Any**	–	None

Corpora are the place where data lives. Data cannot logically call anything. Tools should not use a specialised model, because loading a model for one short task makes the process expensive. Instead, if a tool needs access to a specialised model, it should be introduced as the logic of an assistant at some level, with the proper functionality.

3.7 MOSAIC's Relationship with Existing Work

MOSAIC did not appear from nothing. Existing work such as RAG, MoE, and multi-agent orchestration is not something that MOSAIC completely denies. After various observations, we noticed that MOSAIC's Corpora component can be replaced with RAG. The development of specialised models can be informed by the MoE principle of narrow domain specialisation, but an MoE expert is not equal to a MOSAIC specialised model. A specialised model must earn, by itself, the capability requirements stated in Section 5. Most importantly, unlike an MoE expert, it must be usable alone, without the help of an assistant (Section 5.1). An MoE expert is a sub-network that shares representations with the rest of its model, so it cannot work independently in the same way.

We also state that multi-agent orchestration is the closest architecture to MOSAIC, but there is a real difference at the component level. A multi-agent orchestration system relies on various agentic AIs that communicate freely with each other to find a better result. However, an agentic AI is not the same as a specialised model in MOSAIC's sense. An agentic AI can solve multi-domain work with good accuracy. Specialised models, in contrast, are not selected to solve a broad domain of work, and they are intended to stay low in weight, to respect MOSAIC's efficiency goals. For example, a MOSAIC specialised model named "math" should solve queries related specifically to pure mathematics. An agentic AI can solve math problems across a wider range of mathematical fields, not limited to pure mathematics.

Another difference is the chain-termination behaviour described in Section 6.2. MOSAIC's control loop guarantees termination through two independent, bounded retry budgets. The loop ends even without a fully satisfying answer.

This is a specific, named property of MOSAIC's design. It is not a claim that every multi-agent orchestration system lacks an equivalent mechanism.

3.8 Tools versus Assistants

Tools and assistants are both hardcoded. A common question is what makes them different. Tools are meant to solve a short task. For example, webfetch is a tool that fetches the data of a web page. An assistant, in contrast, is intended to solve tasks that need several tools, with more advanced functionality. An assistant does not only fetch a web page and return the data. It checks whether the data is really present or blocked by an error. Then it tries other methods until it finds the actual data of the page. After that, it processes the data (for example, it removes unnecessary parts), converts it (for example, from HTML to markdown), and makes it cost-efficient and machine readable. Tools are meant to communicate once, while assistants can communicate many times until they are satisfied.

4 MOSAIC Efficiency Law

MOSAIC's efficiency does not come from a single mechanism. It comes from three independent sources that stack together. Each one is already introduced earlier in this report, and here we bring them together as one governing principle:

1. **Model-scale substitution.** A query is resolved with several small, domain-narrow specialised models, not with one large, dense, frontier-scale model (Section 3.4).

   Cost is reduced because the number of parameters activated for a query is limited by what the domain of that query requires. It is not set by the full capacity of a generalist model that can do everything.

2. **Early termination.** A query is resolved at the first level of the hierarchy that is able to answer it. No deeper level is called once the *Self Capability* check of a higher level succeeds (Section 3.1, Section 3.3).

   Here, Self Capability means the check that each level makes to decide whether it can satisfy the task specification on its own. Cost is reduced because most of the query volume is expected never to reach the deeper and more expensive levels of the hierarchy. Complex queries are expected to be rare among general users.

3. **Cumulative partial-resolution.** A level is expected to complete at least one task before it passes the query to a deeper level. This makes sure that the work done by one level is not repeated by a deeper level (Section 3.3).

   Cost is reduced for the calls that still happen, because the task at each level shrinks to what is left after the earlier levels, instead of restarting from the full original query.

Stated as a single governing principle:

> For any query, the total computation that MOSAIC performs is limited by (a) the smallest model scale that is sufficient for the domain of the query, (b) the shallowest hierarchy depth that is sufficient to resolve it, and (c) the smallest residual task size that remains at each level, after the partial progress of earlier levels is counted. It is not set by the computation that a single dense frontier-scale model would need to perform the same task alone.

This principle is what H1 (Section 1.2) claims should hold in aggregate across a representative task distribution. It is also what the call-depth distribution and escalation-rate metrics of Section 8.1 are designed to measure. It is a design target that the architecture is built to satisfy. It is not a result that has already been shown. Mechanisms (2) and (3) depend on the Self Capability check being reasonably accurate. A poorly calibrated Self Capability check could make the system violate this principle in practice, even though the architecture is designed to satisfy it (Section 8.2 discusses this risk).

5 MOSAIC Component Capability Requirements

A MOSAIC architecture is expected to have certain capabilities in order to support H1. Several questions about the components can appear. MOSAIC introduces specialised models and a different kind of Frontier Model, so a natural question is what makes them different from existing models. This section lists some capabilities that these components should have.

5.1 Specialised Model Capability

1. A domain-specific model should contain deep knowledge of its domain and only a mild level of natural language, so that communication stays understandable.
2. A domain-specific model should avoid becoming heavy by increasing its number of parameters, when it already does the job well for a chosen parameter.
3. A domain-specific model must be usable alone, even without the help of an assistant.
4. A domain-specific model should not be a large model by weight. Sharding can be used to create sub-domains if needed. For example, if a specialised model named "physics" is too large, it may introduce "physics_velocity" to keep efficiency.
5. A domain-specific model should be able to run on a CPU at minimum.

5.2 Frontier Model Capability

1. A Frontier Model used in MOSAIC must know the capabilities of the assistants.
2. A Frontier Model must be able to answer general queries without the help of assistants.
3. A Frontier Model must not call any assistant lower than L1. (In the initial design, such a call returns an error anyway, because a low-level assistant accepts calls only from an assistant above it. See the strict downward-delegation rule of Section 3.3.)
4. A Frontier Model must know the communication method between the Frontier Model and the assistants.

6 Structural Guarantees

We separate two categories of guarantee. The first category holds **unconditionally**, as a direct result of the structure. The second holds **conditionally**, and it depends on the individual components being implemented correctly. This distinction is important, because a paper that claims more than the structure really guarantees would overstate the contribution of the architecture.

6.1 Unconditional Guarantee: Topological Non-Looping

Delegation is strictly downward (Section 3.3): no assistant may call a level equal to or above its own. Because of this, the call graph produced by any query is acyclic by construction. No depth counter or runtime cycle check is needed to establish this. It follows directly from the fact that the level strictly decreases along any chain of delegation, and the hierarchy has finite depth. This guarantee holds whatever the implementation quality of any individual level. It is a property of the topology itself.

6.2 Unconditional Guarantee: Retry-Bounded Termination

The top-level control loop (Section 3.1) introduces real cycles that do not exist in the strictly acyclic L1/L2/L3 hierarchy. The loop may return to an earlier step when a sufficiency check fails. Unlike the delegation hierarchy, this loop needs an explicit bound to guarantee termination. The design gives this bound through two independent, bounded retry budgets. The first budget (Maximum Task Call in Figure 1) limits repeated attempts to obtain usable output from a single assistant before another assistant is used. The second, separate budget (Maximum Assistant Call in Figure 1) limits how many unsatisfying rounds the Frontier Model will accept before it produces a response anyway, even if it is not fully satisfied. This makes sure that the system does not loop forever, even on queries for which no satisfying answer exists (Section 9, item 1, discusses this case).

6.3 Conditional Guarantee: Domain and Difficulty-Tier Correctness

The role of L2 includes making sure that a task is routed to a specialised model that matches both the correct domain and the correct difficulty tier. For example, a mathematics-domain model should be used for a pure-math sub-task, not a physics-domain model. In the same way, a model suited to graduate-level mathematics should be used, not a general-purpose or lower-capability model, when precision matters.

This guarantee is different in kind from the guarantees of Sections 6.1 and 6.2. Those hold whatever the correctness of the implementation. This guarantee holds only if the routing logic of L2 is itself designed correctly. The contribution of

the architecture here is to **concentrate the responsibility for this decision in one well-defined place** (L2). Domain-and-tier selection is then not ambiguous or spread across the system. This makes the failure mode easy to audit and to correct in one place if it happens. However, the architecture does not itself verify that the routing logic of L2 is correct. We state this difference explicitly because it is the natural point that a reviewer would question, and because claiming an unconditional guarantee here would not survive careful review.

7 Confidence Estimation

MOSAIC deliberately does not use a self-reported confidence score, produced by a single component, as a gating mechanism anywhere in the architecture. This is a considered design choice, not an oversight. A confidence signal (token-level entropy, prompted self-assessment, or any similar method) measures how consistent a model is with its own earlier output. It does not measure whether that output is correct. A model can collect a lot of internally consistent evidence for a wrong conclusion, and it can report high confidence in that conclusion because of this abundance. This makes self-reported confidence an unreliable gate exactly in the cases where a false answer would look most trustworthy.

MOSAIC instead uses a structural mechanism to reduce the chance that an unfounded or hallucinated value enters the pipeline. It does not rate how trustworthy a value is after the fact. The self-verification step available at L3 (Section 3.3, "Is this correct? Return true or false, then give the reasoning.") is a re-derivation, or a second examination, of the result of the executing step. It is not a confidence rating. It checks whether the specific computation is internally sound. It does not state how certain the system is overall.

We state clearly that this position does not solve the conditional guarantee of Section 6.3. The correctness of domain-and-tier routing at L2 still depends on correct implementation. Also, execution-level self-verification does not structurally guarantee that the computation of a specialised model is correct. A self-verification step is itself a model that examines its own earlier work. A model that reasoned incorrectly the first time could, for the same underlying reason, reason incorrectly again when it re-examines that work. Section 9, item 2, states this limitation explicitly, and the same caution applies here. Avoiding a self-reported confidence *number* is not the same as a guarantee that the output of the hierarchy is correct, and this report does not claim that it is.

8 Proposed Evaluation Plan

This section proposes a direct evaluation of H1 (Section 1.2) against a frontier-model baseline. The metrics are chosen to reflect the specific efficiency mechanisms of Section 4, and not only aggregate accuracy and cost.

8.1 Full-System Efficiency Evaluation

Objective: Test whether a MOSAIC-structured system of specialised models achieves task performance comparable to a single dense frontier-scale model, across a representative task distribution, while escalating to frontier-scale inference on a small minority of queries, as H1 states.

Design: Using a representative benchmark suite that covers multiple domains, compare:

- **Condition A**: queries answered by the full MOSAIC hierarchy (L1 domain identification and delegation, L2 decomposition, L3 execution), with model-scale substitution, early termination, and cumulative partial-resolution working as described in Section 4.
- **Condition B (baseline)**: the same queries answered directly by a single dense frontier-scale model.

Metrics: We measure the following:

- Task accuracy of Condition A compared with the Condition B baseline.
- Overall frontier-escalation rate: the fraction of queries where the MOSAIC hierarchy alone is not sufficient and escalation to a frontier-scale model is required.
- End-to-end inference cost, normalised against the frontier baseline at matched accuracy.
- **Call-depth distribution**: the fraction of queries resolved at L1 alone, compared with those that need L2 or L3. We propose this as the metric that most directly tests the early-termination and cumulative partial-resolution mechanisms of Section 4. It is different from the escalation rate. The escalation rate only tests whether the hierarchy as a whole was sufficient. It does not measure how much of the internal depth was actually used.

Threshold for a positive result: a pre-specified accuracy margin against the frontier baseline, combined with an escalation rate below a threshold set by the authors. Both are set before the evaluation is run, to avoid choosing thresholds after seeing the results.

8.2 Risks to Valid Evaluation

- **Escalation-rate gaming**: a routing policy tuned to minimise the escalation rate as a target, and not as a consequence of really resolving queries correctly, would produce a misleading efficiency result. Accuracy and escalation rate must always be reported together, never the escalation rate alone.
- **Benchmark selection bias**: the choice of tasks should not favour the domains that the implemented L2 and L3 assistants were designed around, because this would overstate generality.
- **Self Capability miscalibration**: as noted in Section 4, early termination and cumulative partial-resolution depend on the Self Capability check of each level being reasonably accurate. The call-depth distribution results should be read together with accuracy. A level that systematically overestimates its own capability could produce a shallow call-depth distribution that looks efficient but hides lower accuracy.

9 Limitations and Risks to the Central Hypothesis

1. **The unanswerable-query case.** The retry-bounded termination guarantee (Section 6.2) makes sure that the system produces a response even when no satisfying answer exists. However, this report does not evaluate whether that response correctly communicates uncertainty or failure to the user, as opposed to a confidently wrong answer that appears only because the retry budget is exhausted. This is a different quality question from termination itself, and we leave it for future work.
2. **Orchestrator logic correctness remains conditional (Section 6.3).** As stated there, the correctness of domain-and-tier routing is not structurally guaranteed. The H1 evaluation implicitly depends on the L1 and L2 routing logic being reasonably well implemented. A poor implementation could produce a false negative on the hypothesis.
3. **Task distribution dependence.** Results on one benchmark suite should not be assumed to generalise to any deployment context without further testing.
4. **No stated mechanism for why H1 should hold beyond the architectural design.** This report proposes three cost-reduction mechanisms as the basis for expecting lower inference cost. However, it does not propose a specific mechanism for why task accuracy should stay comparable to a frontier-scale model as cost is reduced. H1 is evaluated as a joint accuracy-and-cost claim (Section 8.1). The accuracy of H1 depends on the response of the specialised model and on the process of the assistant loop. A positive cost result without a comparable accuracy result would not support H1. This report does not currently propose an explanation for the accuracy side of the claim beyond two conditions: the quality of the specialised models and the correctness of routing.

10 Conclusion

This report has described MOSAIC, an architecture in which a query is resolved through a strict hierarchy of hardcoded orchestration logic, and not through a routing table between models of different sizes. Model access is available at every level, and it is not reserved for the lowest level. Its central proposal is a single efficiency hypothesis (H1). It is supported by three stacking architectural mechanisms: model-scale substitution, early termination, and cumulative partial-resolution. We state them together as a single governing efficiency principle (Section 4), and we do not treat efficiency as an unexplained empirical target.

We have formalised two structural guarantees that the architecture provides unconditionally (topological non-looping and retry-bounded termination). We have also described one guarantee that it provides only if it is implemented correctly (domain-and-tier routing). We proposed an evaluation plan that tests H1 directly against a frontier-model baseline. It uses the call-depth distribution, together with accuracy and escalation rate, to measure the specific internal mechanisms of the architecture, and not only the aggregate cost.

We regard running the full-system efficiency evaluation (Section 8.1) as the highest-priority next step for the project. We also state explicitly (Section 9, item 4) that this report does not yet propose a specific mechanism for the accuracy side of the joint claim of H1. This remains an open question for future work.

Acknowledgement

The authors declare that this work (1) has not received any external funding and (2) has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 1 2017.

[2] William Fedus, Barret Zoph, Noam Shazeer, and Alexander Clark. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23:1–39, 2022.

[3] Simone Scardapane, Alessandro Baiocchi, Alessio Devoto, Valerio Marsocci, Pasquale Minervini, and Jary Pomponi. Conditional computation in neural networks: Principles and research trends. *Intelligenza Artificiale*, 18(1):175–190, 7 2024.

[4] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang (Eric) Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *www.microsoft.com*, 8 2023.

[5] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv:2302.04761 [cs]*, 2 2023.

[6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 4 2021.

[7] Thomas G. Dietterich. Ensemble methods in machine learning. *Multiple Classifier Systems*, 1857:1–15, 2000.

[8] Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance, 5 2023.

[9] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 7 2024.

[10] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems 36*, pages 38154–38180, 2023.

[11] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 6 2024.

[12] Matthew C. Rosen and David J. Freedman. How distributed is the brain-wide network that is recruited for cognition? *Nature Reviews Neuroscience*, 27(2):138–150, 12 2025.

A Architecture Diagram Reference

The architecture described in Section 3.3 corresponds to the following diagrams developed during this project: the top-level control loop (Frontier Model, evidence and satisfaction checks, bounded retry) and the L1/L2/L3 delegation hierarchy (strictly downward calling, and per-level branching into tools, corpus, specialised model, and further delegation). The component names in this report (L1, L2, L3, Frontier Model, Assistant A) match the labels used in those diagrams.

B Glossary

Agentic AI An artificial intelligence system that can accomplish a specific goal with limited supervision. It consists of AI agents, which are machine learning models that imitate human decision-making to solve problems in real time. In a multi-agent system, each agent performs a specific subtask that is needed to reach the goal, and their work is coordinated through AI orchestration.

L1 / L2 / L3 The three levels of the delegation hierarchy, ordered from the broadest task scope (L1) to a fully atomic execution (L3). See Section 3.3.

L1 Memory (Assistant Memory) A sub-component of the assistant loop that stores the log, progress, activities, and results of the task that each assistant performs. See Section 3.3.1.

Orchestration logic The hardcoded, non-model control procedures that form an assistant at any level. They are explicitly distinguished from the models that such logic calls. See Section 3.1.

Self Capability The check that a level makes to decide whether it can satisfy the task specification on its own, without delegating to a deeper level. See Section 4.

Topological non-looping guarantee The unconditional guarantee that the delegation call graph is acyclic. It follows from strictly downward-only delegation. See Section 6.1.

Retry-bounded termination guarantee The unconditional guarantee that the top-level control loop ends, through two independent bounded retry budgets. See Section 6.2.

H1 (Efficiency hypothesis) The claim that a MOSAIC-structured system of specialised models can match the accuracy of a frontier model while escalating rarely. See Section 1.2.

C MOSAIC Solving a Citation Claim Verification Query

This appendix presents one worked example of the cost-depth relationship described in Section 4. The architecture tries to resolve the query at the cheapest available point first, and it moves to deeper and more expensive levels only when a shallower attempt is not sufficient. This trace shows the mechanism in a concrete way. It is one example, not a general result, and it does not replace the empirical evaluation proposed in Section 8.1.

User query: "Here is a claim: 'Nabil (2026) claimed that a single conventional citation format used in a paper can introduce 32–100 tokens for an LLM when reading.' Is it true?"

Frontier Model: This looks like a citation-claim verification task. I need to find the ground truth. Which assistant can handle this? Assistant A looks right. Sending the call now, and waiting for a response.

Assistant A: I got a call from the Frontier Model. It wants a claim verified. Let me check whether I have the minimum documents needed to proceed. I have a title and a partial author name, which is enough to start. I will call the Specialised Model to learn who Nabil is and whether he has worked on this topic. First, let me call Corpora for the right template, one for when an author name and a topic are both present. I got the prompt. Calling Model Scholar: "An author named 'Nabil' worked on a topic described as 'a single conventional citation format used in a paper can introduce 32–100 tokens for an LLM when reading,' in 2026. Do you have enough data to locate a DOI or URL?" Model Scholar returned a DOI. Retrieving the metadata now. I got it. Now let me ask Model Fact Check to verify whether the metadata supports the claim. I read the corpus first for the right prompt before passing the data along. Calling Fact Check: "Here is an abstract, title, and author details, and here is a claim. Does the abstract reflect the claim?" Fact Check says it cannot confirm this from the abstract alone. This means the task needs to go to L2 for a full-paper check. Before delegating, I will pass along the claim and the metadata I already collected, so that L2 does not have to start from nothing.

Assistant L2-A: I got a call from Assistant A. It contains a claim to verify, with metadata attached and a note that a full-paper check is needed. This is a large and complex task. I could appoint an L3 assistant to work in parallel while I verify the claim myself, but that would leave L3 doing all the heavy work. Let me retrieve the full paper first. I got it. I will clean it slightly before handing it over. Now I call an L3 assistant to check whether any section, sentence, or stated result in the paper really confirms the claim. It found one, so I pass along everything I have.

Assistant L3-A: I got a call from Assistant L2-A. It wants the claim checked against the full paper content. I read the corpus for the right prompt for the case where full paper content and a claim to verify are both present. I found it. Calling the Deep Fact Checker Model: "Here is a claim, and here is the full paper content. Find the closest or exact matching content, or identify where the claim differs from what is actually stated." I got the result in a clean, structured format. This is as far as I can take it, so I forward it to L1 Memory.

Data sent to L1 Memory: "The claim is misleading; the data cited is inaccurate. A similar claim appears in the Benchmark section of the paper, which is likely where it came from. The actual finding of the paper is: 'the results showed across multiple tokenisers and associated models that popular conventional citation formats, including IEEE, Harvard, APA, and 20 other formats, require 31–59 input tokens to represent a selected citation.' "

L1 Memory checks whether this follows an established pattern. It does. The result is sent to the Frontier Model.

Frontier Model receives the response.

Frontier Answer: No, the claim as stated is not accurate. What Nabil et al. (2026) actually found is that 20 different conventional citation formats introduce 31–59 tokens for an LLM to read, tested across multiple tokenisers and associated models. More detail is available in the Benchmark section of the paper.

Observations on this trace. Two different specialised models are called at two different depths. Each one is scoped to the material that is actually available at that point in the chain. Fact Check (L1) works on the abstract alone, because this is all that Assistant A has retrieved at that stage. The Deep Fact Checker Model (L3) works on the full paper content, which is available only after L2 has retrieved and prepared it. This is consistent with the granularity argument of Section 3.3.2: a deeper level is not simply "more capable" in the abstract. It is scoped to a different and more complete input than the level above it.

The trace also shows that escalation happens only when it is forced. Assistant A tries to resolve the query with its own model calls before it delegates to L2. L2 delegates to L3 only after it judges that verifying the claim by itself would overload L3 with the whole task. No level is given work that it did not first try to resolve on its own.

D Encephalon of Artificial Intelligence Conjecture

The **Encephalon of Artificial Intelligence (eAI)** is a proposed AI architecture inspired by the distributed and selective nature of biological neural computation. It does not require one model to perform every computational operation with its full available capability. Instead, an eAI architecture distributes computation across a general-purpose model and a collection of specialised model components, which can be selectively activated according to the requirements of a task.

The human brain contains about 86 billion neurons, but neural computation does not require all neurons to be active at the same time for every task [12]. Information is processed through patterns of electrochemical activity that are distributed across interconnected groups of neurons. Different groups can therefore take part in different computational processes at different times and with different levels of activity.

By analogy, an eAI system does not necessarily need to activate its whole model capability for every task. A Main Model is loaded first, and it acts as the general-purpose computational component. When the Main Model needs to expand its knowledge or capability, it can communicate with one or more specialised model components, and selectively activate or load them. The objective is to allocate computational capability according to the requirements of a task, and not to apply the full model capability to every task.

The analogy is architectural, not biological. eAI does not claim to reproduce the mechanisms of the human brain. It adopts the principle that useful computation can emerge from the selective coordination of several computational components.

Rules of the Encephalon of Artificial Intelligence

1. **General-purpose core:** The architecture contains a Main Model for general-purpose reasoning, coordination, and task decomposition.
2. **Modular components:** The architecture contains a finite, or potentially extensible, collection of specialised model components. Each one can provide particular knowledge or computational capabilities.
3. **Selective activation:** The Main Model is lightweight compared with the complete system, and it is never intended to solve all problems by itself. Specialised components are activated or loaded when their capabilities are relevant.
4. **External knowledge and tools:** When the required information or capabilities are not available in the active models, the system may obtain them through corpora, retrieval systems, external data sources, or computational tools.
5. **Bounded computation:** The architecture contains a hard loop logic, such as explicit termination conditions or iteration limits, to prevent unbounded reasoning or execution loops.

eAI's Rule Inspiration from the Human Encephalon

1. **General-purpose core.** This rule was adapted from observing human general knowledge, where a person answers a query without needing to reason about it.
2. **Modular components.** This rule reflects how electrical signals pass from one neuron to another through various biochemical activities.
3. **Selective activation.** This rule explains the case where a human activates or expands the use of neurons when a query is challenging to solve.
4. **External knowledge and tools.** This rule explains the case where a human is stuck in a process and learns from a book, the internet, or another reliable source to solve the query.

5. **Bounded computation.** This rule explains the case where a human never finds valuable information, and then slows down or gives up the process. An AI must give up as well. Otherwise, a system without a hard loop could spend 100 years solving a query, like the great scientists. An AI is expected to solve a query faster than a human, or to give up completely.

The central conjecture of eAI is therefore the following:

> **An AI system may achieve efficient, accurate, task-specific computation by selectively coordinating specialised computational components, rather than requiring the full available model capability to take part in every task, in the same way that a brain does not activate its full capability all the time.**

E Illustrative Deployment Footprint and Cost

This appendix gives an illustrative estimate of the hardware that MOSAIC needs, and of its cost and latency. **Every figure here is an estimate from stated assumptions. Nothing was measured.** The purpose is to show the shape of the trade-off, not to report results. All values should be replaced by measurements before any claim is made from them (Section 8.1).

E.1 Execution Model

MOSAIC follows the eAI rule of selective activation (Appendix D). A specialised model is **loaded when it is called, does its task, and is unloaded**. No specialised model stays in GPU memory between calls. The Frontier Model is the only component that stays resident. This is what keeps the memory footprint small. It also means that every specialised call pays a boot cost.

E.2 Assumptions

Models. All specialised models have 0.5B parameters (about 1 GB at FP16). They differ in training data, not in size. For example, the Fact Check model and the Deep Fact Checker model of Appendix C have the same weight footprint. The Frontier Model has 7–8B parameters, 4-bit quantised (about 5 GB), and stays resident.

Hardware and price. These are the rental quotes used for this estimate. They were not independently verified.

Table 3: Rental quotes used for the estimate.

Machine	GPU memory	Host RAM	vCPUs	Price
RTX 3090	24 GB	125 GB	16	$0.50/hr
A100 SXM	80 GB	125 GB	16	$1.59/hr

Speed and boot (unmeasured guesses). On the RTX 3090: Frontier decode about 70 tokens/s and prefill about 3,000 tokens/s; specialised decode about 200 tokens/s and prefill about 10,000 tokens/s. Boot of one specialised model is about 1.2 s when its weights are already in host RAM (a copy plus initialisation), and about 1.7 s when they are read from disk. The baseline is a 32B model at FP16 (about 64 GB of weights) on the A100, with decode about 30 tokens/s and prefill about 4,000 tokens/s. This baseline is above the 27B minimum of Table 1 (Section 3.4.1), so that the comparison keeps some headroom.

E.3 Memory Footprint

Under strict on-demand execution, peak GPU memory is limited by

$$M_{\text{peak}} = F + \max_i (s_i + KV_i) \tag{1}$$

where F is the footprint of the resident Frontier Model (weights, KV cache, and overhead), s_i is the weight footprint of specialised model i, and KV_i is its KV-cache memory at the input length of that call. M_{peak} does **not** grow with the number of specialised models in the system. It grows only with the largest single model and the longest single input.

Table 4: Estimated GPU memory need by configuration.

Configuration	Estimated need	Notes
One active specialised model, short context	about 7.5–9 GB	Frontier about 6–7 GB plus one 0.5B model and framework overhead
Practical floor	about 12 GB	Leaves about 3–4.5 GB for framework overhead, memory fragmentation, and longer inputs
Several specialised models active at once (parallel branches), short context	about 13–17 GB	Roughly 6–7 models fit within 24 GB
Baseline, 32B at FP16	about 64 GB of weights	Needs an 80 GB GPU in practice

Two conditions apply. First, KV-cache memory grows with input length, so long inputs (such as the 10,000-token deep check of Appendix C) reduce the number of specialised models that can be active at once. Second, the parallel figure applies only when several branches run at the same time (Section 3.3). If calls are strictly sequential, one specialised model is active at a time and the floor is the relevant number.

This is a statement about **memory footprint**, not answer quality. Equal accuracy is what H1 proposes to test (Section 8.1) and it is not shown here.

E.4 Illustrative Cost and Latency

Per-query cost is time on the machine multiplied by the hourly price. For a specialised call:

$$T_{\text{call}} = T_{\text{boot}} + T_{\text{prefill}} + T_{\text{decode}} \tag{2}$$

The Frontier Model is called at least twice per query (dispatch, then judgement and answer; Section 3.1). The total is the sum over all calls plus orchestration time. Because retries are bounded by two budgets (Section 6.2), the worst case is bounded:

$$C_{\text{worst}} \leq C_{\text{single_pass}} \times (1 + R_{\text{assistant}}) \times (1 + R_{\text{frontier}}) \tag{3}$$

MOSAIC on the RTX 3090 compared with the baseline on the A100 (different machines and prices):

Table 5: MOSAIC on an RTX 3090 versus the 32B baseline on an A100 (illustrative).

Scenario	MOSAIC	Baseline	Cost	Latency
Short query, resolved at L1	about \$0.0010, 7 s	about \$0.0030, 7 s	about 3× lower	similar
Long-input query, full L1–L3 chain	about \$0.0020, 14 s	about \$0.0041, 9 s	about 2× lower	about 1.5× slower
Worst case, both retry budgets used once	about \$0.0079, 57 s	about \$0.0041, 9 s	about 2× higher	about 6× slower

Both systems on the same A100 (same price):

Table 6: MOSAIC and the 32B baseline on the same A100 (illustrative).

Scenario	MOSAIC	Baseline	Cost	Latency
Short query, resolved at L1	about \$0.0023, 5 s	about \$0.0030, 7 s	about 1.3× lower	about 1.3× faster
Long-input query, full L1–L3 chain	about \$0.0053, 12 s	about \$0.0041, 9 s	about 1.3× higher	about 1.3× slower
Worst case, both retry budgets used once	about \$0.0212, 48 s	about \$0.0041, 9 s	about 5× higher	about 5× slower

E.5 What These Estimates Show and Do Not Show

1. **The main benefit is a smaller machine, not a faster or cheaper computation on the same machine.** On equal hardware the cost advantage almost disappears (second table). The advantage in the first table comes mostly from the lower price of the RTX 3090. MOSAIC can run on a 12–24 GB GPU where a 64 GB baseline cannot run at all. We do not claim that MOSAIC outperforms a large GPU.
2. **Cost and latency trade against each other.** The trade is small for queries resolved at shallow depth and poor for deep chains and for retry exhaustion. The call-depth distribution (Section 8.1) decides which case dominates in real use.
3. **Boot cost is small per call but adds up.** It is about 1.2–1.7 s for each specialised call. For a long input, the prefill time of the deep-check call is larger than the boot cost, so depth is expensive mainly because of the input the level holds, not because of model size.
4. **Retry budgets are also a cost control.** Cost at budget exhaustion should be reported together with typical cost.
5. **These numbers hold only if accuracy holds.** The comparison is meaningful only if specialised-model accuracy and routing accuracy (Section 6.3) reach the levels that H1 needs. If they do not, no cost or latency figure supports H1.

E.6 Limits of This Appendix

- All speeds, boot times, and the 10,000-token input are assumptions.
- The rental prices come from stated quotes and change often. They are hardware-hour costs, not the retail prices of commercial APIs, and the two should not be compared directly.
- Idle cost is excluded. A resident Frontier Model costs money even when no query arrives, and a warm baseline serving many users can lower its cost per query by batching. Both systems must be compared at a stated traffic level.
- The estimates assume sequential calls, unloading that really frees GPU memory, and that the 0.5B models handle the stated input length.
- The 32B baseline is not a frontier-scale model. Frontier-scale model sizes are not public, so we do not estimate their cost here.
- The A100 comparison uses a single isolated query. Running several MOSAIC instances on the same card would reduce cost per query, and it is not computed here.

This report reflects the design of MOSAIC as of 12th August, 2026 (Version 2). The next version may introduce structural, architectural, and conceptual changes, depending on the results of the evaluation.