

The Notion-Based Reasoning System

Structure-Content Decomposition for Reasoning

Aryan Gupta
aryan.cs.app@gmail.com

June 7, 2026

Abstract

Current large language models simulate multi-step reasoning by autoregressive token prediction over surface representations. This collapses on tasks requiring transfer of an abstract procedure across content domains: the same algorithm applied to letter-string puzzles in a novel alphabet, the same equilibrium argument applied across physics and economics, the same inductive proof pattern applied across number theory and graph theory. The failure has a structural diagnosis. A standard transformer does not factorise the relational scaffold of a problem from its specific content, and so has no representation that can be transferred independently of content.

We propose the Notion-Based Reasoning System (NBRS), motivated by the structure-content factorisation observed in the hippocampal-entorhinal cognitive map [1, 2, 3] and by the structure-mapping account of analogy [7, 8]. NBRS maintains an explicit library of *notions*: parameterised relational templates over typed operations, induced from few examples, stored as content-independent abstractions, and applied to new problems via structure-mapping. An LLM serves as the content engine that parses problems and supplies values; a separate structural layer maintains the notion library and selects, binds, and instantiates notions; the resulting concrete computation runs on a deterministic graph executor (the Graph-Structured Recurrent Executor, retained as substrate). Library growth is driven by a replay-style consolidation process that abstracts notions from successful traces.

The technical contributions of this document are: (i) formal definitions of notions, structural binding, and analogical mapping over typed operation DAGs; (ii) the architectural specification of NBRS as a five-component system (content layer, structure extractor, notion library, analogy engine, execution substrate) with the consolidation process that grows the library; (iii) execution guarantees for instantiated notions inherited from the underlying executor (determinism, termination, soundness, depth-decoupling); (iv) a precise statement of the transfer property the architecture is designed to achieve, with the conditions under which it holds; (v) a falsifiable empirical programme centred on two cross-domain transfer demonstrations: ARC-AGI subset performance and cross-discipline scientific-equilibrium analogy.

1 Introduction

1.1 The transfer gap

The capability that most distinguishes human reasoning from contemporary large language models is not depth, breadth, or factual coverage. It is *transfer*: the ability to recognise that a problem in a new domain has the same structure as a problem already solved in some other domain, and to apply the previously-learned procedure to the new content without further training. A child who has learned long-division on integers can apply the same algorithm to polynomials with minimal additional instruction. A physicist who recognises that supply-demand equilibrium has the structure of mechanical equilibrium can transfer intuitions about damped oscillations to market dynamics. A reader who absorbs the moral of a parable can

identify the same structural pattern in a story with different characters, settings, and surface content.

Recent stress tests of analogical reasoning in frontier LLMs document the gap precisely. Lewis and Mitchell [9] and Webb et al. [10] show that performance on letter-string analogies, Raven’s Progressive Matrices, and digit-pattern problems collapses when test stimuli are constructed to be unlikely in training data: GPT-class models fail on novel alphabets where humans, including children, generalise robustly. Chollet’s ARC-AGI benchmark [11, 12] was constructed specifically to expose this failure: each task supplies a handful of input–output examples of an abstract transformation and requires applying it to a held-out input. The failure mode is consistent across studies: LLMs match on surface features, not on relational structure. Where the abstract pattern requires a content-independent representation, the model has nothing to fall back on.

1.2 The neurobiological evidence for structure–content factorisation

The brain solves the transfer problem with an architectural commitment that contemporary neural networks do not make. The hippocampal–entorhinal system maintains *factorised* representations: structural codes encode the relational scaffold of a task or environment (the graph of transitions, the relative positions, the type relationships), while sensory codes encode the specific content (which objects are at which positions, which values fill which slots). Conjunctive cells in the hippocampus combine them.

The factorisation is striking enough that the same grid-cell apparatus that organises physical spatial navigation also organises navigation through abstract conceptual spaces. Constantinescu, Behrens, and colleagues [3] demonstrated that human subjects navigating a continuous two-dimensional space of bird shapes (defined by neck length and leg length) exhibit the same six-fold hexagonal fMRI signature that grid cells produce for physical space. The brain’s structural representation is domain-general; what changes between physical and conceptual tasks is the content bound to that structure, not the structure itself.

The Tolman–Eichenbaum Machine [1] formalises this as a computational model. After training on diverse environments, its medial-entorhinal cells learn to represent the abstract transition structure shared across environments; its lateral-entorhinal cells encode the sensory content; its hippocampal cells encode conjunctions. Transfer to a new environment requires re-binding sensory content to an already-learned structure, not relearning structure from scratch. Recent extensions and ablations [4, 5] show that the factorisation is necessary for the model’s transfer behaviour: removing it removes the generalisation property.

1.3 The proposal

We propose the Notion-Based Reasoning System (NBRS): a neurosymbolic system that operationalises the cognitive-map factorisation for procedural reasoning. NBRS maintains a library of *notions*: parameterised relational templates over typed operations, induced from few examples, stored as content-independent abstractions, and applied to new problems by mapping the problem’s relational structure onto the notion and binding the notion’s variables to problem content. An LLM provides the content layer (parsing problems, supplying values, generating natural-language output); a structural layer parses content into relational graphs and operates the notion library; instantiated notions execute on the deterministic typed-DAG Graph-Structured Recurrent Executor; a consolidation process driven by replay over successful traces grows the library over time.

The technical content of this document is in five parts. Section 2 fixes the conceptual framework: the structure–content distinction, notions as the unit of transfer, the relationship to cognitive maps. Sections 3 and 4 give the formal preliminaries and define notions, instantiation, and structural matching. Section 5 specifies the five-component architecture. Section 6

compresses the execution substrate; full operational properties are retained in Appendix A. Sections 7 and 8 develop the notion-induction and analogical-transfer mechanisms, including conditions for soundness. Section 10 states the empirical programme: two cross-domain transfer demonstrations on which the architecture is to be evaluated against contemporary LLM baselines.

2 Conceptual Framework

2.1 The structure–content distinction

A reasoning problem has two separable aspects. The *content* consists of the specific values, entities, types, and surface features that the problem mentions. The *structure* consists of the relational scaffold: which entities depend on which others, what operations connect them, what invariants hold, what the goal is in relation to the givens.

Two problems share content when they mention the same things; they share structure when the dependency graphs of their solutions are isomorphic up to relabelling. Contemporary LLMs blend these in a single token stream and a single attention computation. When training distributions cover the content–structure joint distribution densely, this works; when test problems combine familiar structure with unfamiliar content, it breaks.

2.2 Notions as the unit of transfer

We define a *notion* as a parameterised relational template: a typed operation DAG in which leaves, constants, and primitive choices are replaced by typed variables. Concretely, a notion for the binary-search procedure has the form “given a sorted sequence x of type T^* and a target value y of type T , iteratively halve the search interval according to a comparison primitive $\text{cmp} : T \times T \rightarrow \text{Bool}$.” This template has no commitment to what T is, what specific elements x contains, or what specific value y takes. Instantiating the notion requires binding T , x , y , and cmp to a concrete problem; once bound, the notion produces a concrete typed operation DAG that the executor can run.

A notion, in this sense, is exactly the object that the cognitive-map literature identifies as the entorhinal structural code. It is the abstract pattern that survives transfer; the binding is the conjunctive code that combines structure with content.

2.3 Analogical transfer as structural alignment

Given a new problem P and a library of notions, transfer proceeds in three steps. First, the structure extractor parses P into a relational graph G_P describing P ’s entities, types, and dependencies. Second, the analogy engine searches the library for a notion N whose structure aligns with G_P under some type-respecting binding β . Third, the binding β instantiates N into a concrete operation DAG that the executor runs.

The structural alignment in the second step is precisely Gentner’s structure-mapping [7]: a partial isomorphism between notion variables and problem entities that preserves the relational structure. It is not surface similarity. Two problems with no shared surface features can share structure, and the analogy engine finds the notion whose structure matches, regardless of whether the problem’s content was ever seen during library construction.

3 Formal Preliminaries

We work in a typed first-order setting. Values have types; operations have input and output types; reasoning is a typed computation organised as a DAG; notions are typed DAGs with variables.

3.1 Types, values, primitives

Definition 3.1 (Type system). A type system is a pair $(\mathcal{T}, \mathcal{V})$ where \mathcal{T} is a finite, decidable set of types and $\mathcal{V} = \bigsqcup_{\tau \in \mathcal{T}} \mathcal{V}_\tau$ is a disjoint union of value domains. We write $v : \tau$ for $v \in \mathcal{V}_\tau$. The type system may include type constructors: list types T^* , product types $T \times T'$, function types $T \rightarrow T'$.

Definition 3.2 (Primitive operation). A primitive operation is a tuple $\tau = (\text{name}_\tau, \text{in}_\tau, \text{out}_\tau, f_\tau)$ with $\text{in}_\tau \in \mathcal{T}^*$ of length k_τ , $\text{out}_\tau \in \mathcal{T}$, and $f_\tau : \mathcal{V}_{\tau_1} \times \cdots \times \mathcal{V}_{\tau_{k_\tau}} \rightarrow \mathcal{V}_{\text{out}_\tau}$ a total semantic function. A primitive library $\mathcal{L} = \{\tau_1, \dots, \tau_K\}$ is a finite set.

3.2 Operation nodes and operation DAGs

An operation node is an instance of a primitive operation in a particular computation. An operation DAG is a typed, dependency-ordered graph of operation nodes whose root is the computation's output. Operation DAGs are the substrate on which instantiated notions execute.

Definition 3.3 (Operation node). An operation node over a library \mathcal{L} is a tuple $n = (\text{id}_n, \tau_n, \text{spec}_n, \text{pa}_n)$ where id_n is a unique identifier, $\tau_n \in \mathcal{L}$, spec_n encodes auxiliary parameters of the primitive, and $\text{pa}_n = (p_1, \dots, p_{k_{\tau_n}})$ is the ordered tuple of parent identifiers. A leaf node has $\text{pa}_n = ()$ and a concrete leaf value supplied externally.

Definition 3.4 (Operation DAG). An operation DAG is a pair $G = (V, r)$ with V a finite set of operation nodes, all parent identifiers referring to elements of V , the induced parent relation acyclic, and $r \in V$ a distinguished output node. The DAG is well-formed when (i) types match along every edge and (ii) every non-leaf node has exactly k_{τ_n} parents. All operation DAGs in the remainder are assumed well-formed.

Definition 3.5 (Compositional depth). The depth of node n is $\text{depth}(n) = 0$ if $\text{pa}_n = ()$ and $1 + \max_{p \in \text{pa}_n} \text{depth}(p)$ otherwise. The depth of the graph is $\text{depth}(G) := \text{depth}(r)$.

4 Notions

Notions are the load-bearing object of NBRs: they are what the library stores, what the analogy engine matches, and what transfer transfers.

4.1 Notion variables

Where an operation node has a concrete primitive $\tau_n \in \mathcal{L}$ and a concrete spec, a node within a notion may have a variable in any of these positions. We introduce three classes of variables, corresponding to the three axes of content along which transfer must abstract.

Definition 4.1 (Notion variables). A notion admits three kinds of variables, drawn from disjoint countable sets:

- *Type variables* X, Y, Z, \dots ranging over \mathcal{T} . A notion is parameterised over the types of values it operates on.
- *Value variables* x, y, z, \dots ranging over \mathcal{V} . A notion is parameterised over the specific leaf values supplied at runtime.
- *Operator variables* $\alpha, \beta, \gamma, \dots$ ranging over \mathcal{L} restricted by type signature. A notion may leave the choice of a particular primitive (e.g., the binary operation passed to a fold) as a parameter.

4.2 Notions as parameterised templates

Definition 4.2 (Notion). A notion N is a tuple $(V_N, r_N, \text{params}_N)$ where:

- V_N is a finite set of *parameterised nodes*, each of the form $(\text{id}, \alpha, \text{spec}, \text{pa})$ where α may be either a concrete primitive in \mathcal{L} or an operator variable, and spec may contain value variables.
- $r_N \in V_N$ is a distinguished root.
- params_N is a list of typed parameters: each type variable's kind, each value variable's type (which may itself reference a type variable), each operator variable's input/output type signature (likewise).

The induced parent relation must be acyclic and well-typed under any consistent variable assignment.

Example 4.3 (The fold notion). The notion $\text{FOLD}(\alpha, x_0, \ell)$ has parameters: a type variable X (element type), a type variable Y (accumulator type), an operator variable $\alpha : Y \times X \rightarrow Y$, a value variable $x_0 : Y$ (initial accumulator), and a value variable $\ell : X^*$ (the sequence). The notion body is a length-dependent chain: for a sequence of length k , the notion expands into a depth- k chain of α invocations starting from x_0 . We treat the notion as a function from inputs to a concrete operation DAG via the expansion rule.

Example 4.4 (The binary-search notion). The notion $\text{BSEARCH}(\text{cmp}, \ell, y)$ has a type variable X , an operator variable $\text{cmp} : X \times X \rightarrow \text{Ordering}$, a value variable $\ell : X^*$ (a sorted sequence), and a value variable $y : X$ (the target). The body iteratively halves the candidate range using cmp until the target is located or the range is empty.

Example 4.5 (The equilibrium notion). The notion $\text{EQUILIBRIUM}(\alpha_+, \alpha_-, x)$ has a type variable X (the quantity), two operator variables $\alpha_+, \alpha_- : X \rightarrow X$ (the opposing influences as functions of the quantity), and a value variable $x : X$. The body asserts $\alpha_+(x) = \alpha_-(x)$ and solves for x . Instantiating this notion to mechanical equilibrium binds α_+ and α_- to applied and restoring forces; instantiating it to market equilibrium binds them to supply and demand functions. The notion is a single object, applied across both domains by appropriate binding.

4.3 Instantiation

Definition 4.6 (Notion instantiation). A binding β for a notion N is a map from N 's parameters to concrete types, values, and primitives respecting the parameter signatures. The instantiation $\text{Inst}_\beta(N)$ of N under β is the concrete operation DAG obtained by substituting β 's assignments into the notion body, applying any length-dependent expansion rules, and resolving the resulting DAG's types.

Proposition 4.7 (Well-typed instantiation produces a well-formed DAG). *For any notion N and any binding β that respects N 's parameter signatures, $\text{Inst}_\beta(N)$ is a well-formed operation DAG.*

Proof. Direct from the notion's typing discipline and the well-formedness of the substitution. Type variables are bound to concrete types; value variables to values of the bound type; operator variables to primitives with the bound signature. Edge typing in the instantiated DAG follows. \square

4.4 Notion composition

Notions compose by allowing a parameterised node to invoke another notion rather than a primitive. The composition mechanism is identical to function calling in a typed lambda calculus: a notion N that uses N' as a sub-procedure simply contains a parameterised node whose

operator is N' (with its parameters bound either to concrete values or to outer variables of N). Composition is therefore the mechanism by which higher-level reasoning patterns (e.g., divide-and-conquer) are built from lower-level ones (e.g., a sort, a merge).

5 Architecture

NBRS is a five-component system. Each component has a specific responsibility and a specific interface to the others.

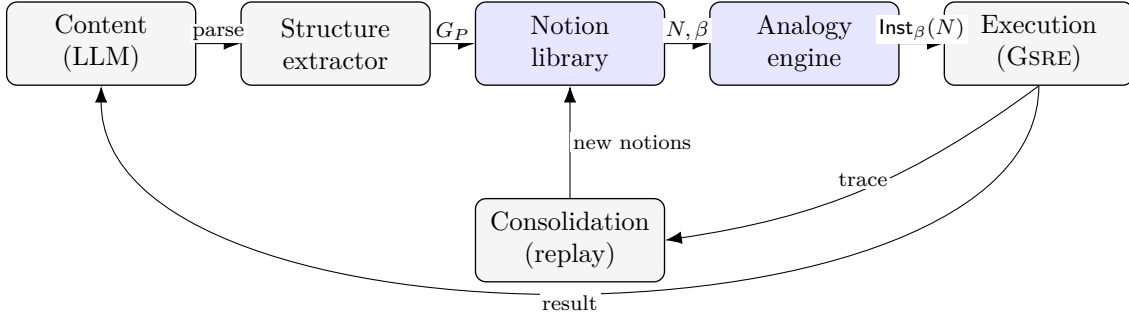


Figure 1: The Notion-Based Reasoning System. The pipeline is linear: the content layer parses the problem, the structure extractor produces a problem graph G_P , the notion library is queried by the analogy engine which returns a matched notion N with a binding β , and the instantiation $\text{Inst}_\beta(N)$ runs on the execution substrate. Two feedback loops grow and apply the system: traces from successful executions feed the consolidation process, which abstracts new notions back into the library; the executor’s final value returns to the content layer for natural-language output.

5.1 Content layer

The content layer is a frontier LLM. Its responsibility is restricted to what LLMs do well: parsing natural-language problems into entities and types, supplying concrete values, recognising domain-specific patterns at the surface level, and generating natural-language output from concrete answers. It does not perform multi-step reasoning. The architectural assumption is that an LLM trained on web-scale text is a competent semantic parser and a competent content generator, even when it cannot reliably execute multi-step logical procedures.

5.2 Structure extractor

The structure extractor receives a parsed problem from the content layer and produces a *problem graph* G_P : a typed relational graph describing the problem’s entities (typed nodes), the relations among them (typed edges or operation nodes), the givens (which entities are known), and the goal (which entity is to be computed). The structure extractor abstracts away from surface vocabulary as far as it can while preserving relational fidelity. A simple word problem about apples and a structurally identical problem about voltages should produce isomorphic problem graphs modulo type relabelling.

5.3 Notion library

The notion library \mathcal{N} is a growing set of notions. Each notion is stored with: its parameterised structure (Definition 4.2), a precondition predicate over problem graphs that determines applicability, example traces from which the notion was abstracted, and meta-information used by the analogy engine (frequency of use, success rate, average depth of instantiated DAG).

5.4 Analogy engine

The analogy engine receives a problem graph G_P and queries the library to find notions whose structure aligns with G_P . Alignment is a partial graph isomorphism under a typed binding: a map from the notion’s variables to entities, types, and operators in G_P that preserves the relational structure. Where multiple notions match, the engine ranks candidates by structural coverage, parameter fit, and prior success rate; where partial matches are possible, the engine may propose a binding that uses a sub-notion and leaves the residual structure to be solved by another (recursive) library lookup.

The analogy engine is the architectural analogue of Gentner’s Structure-Mapping Engine [8], lifted to learned representations of types and primitives.

5.5 Execution substrate

Once a notion N is selected and bound, $\text{Inst}_\beta(N)$ is a concrete operation DAG. It runs on the Graph-Structured Recurrent Executor (Section 6, full treatment in Appendix A). The substrate provides the determinism, termination, and soundness guarantees that make the architecture’s output predictable from the library’s content.

5.6 Consolidation

The consolidation process is offline and replay-driven, in deliberate analogy to hippocampal replay during rest [5, 6]. The consolidation engine periodically examines the corpus of recent successful traces, looks for recurring structural patterns, abstracts these patterns into candidate notions by replacing constants with typed variables, and (subject to verification on held-out traces) adds them to the library. Section 7 treats this in detail.

6 The Execution Substrate

The execution layer of NBRS is the Graph-Structured Recurrent Executor (GSRE). We summarise its properties here and defer the full operational specification to Appendix A. The role of this section is to establish what the architecture inherits from the executor, not to develop it as a contribution.

6.1 Summary of executor properties

The GSRE is a deterministic state machine over an operation DAG. Each node carries a state in $\{\text{PENDING}, \text{READY}, \text{EXECUTING}, \text{RESOLVED}, \text{FAILED}\}$; a node becomes **READY** only when all of its parents are **RESOLVED**. A shared neural module NodeProc_θ is invoked at each node on its local context $(\tau_n, \text{spec}_n, (v_p)_{p \in \text{pa}_n})$ alone. The same parameters θ are reused at every node, regardless of position or graph size.

The properties of relevance to NBRS are:

- *Determinism.* For any DAG G and leaf assignment, the executor produces a unique value assignment (Theorem A.1).
- *Order independence.* The value assignment is invariant under any choice of topological schedule among ready nodes (Theorem A.2).
- *Termination.* Execution completes in at most $|V|$ steps (Theorem A.3).
- *Soundness against the oracle.* If NodeProc_θ is primitive-correct on every node it sees, the executor’s output equals the oracle’s (Theorem A.4).

- *Per-step bounded input.* The per-invocation input size is bounded by the maximum primitive arity, independent of $\text{depth}(G)$ and $|V|$ (Proposition A.5).
- *Capacity–depth decoupling.* Under an independent-error model, success probability decays at worst geometrically in depth, with the per-step parameter count constant in depth (Theorem A.6).

6.2 Inherited guarantees for instantiated notions

The point of inheriting the substrate is the following: every notion instantiation $\text{Inst}_\beta(N)$ is, by construction, a well-formed operation DAG. The executor properties above therefore apply directly. The architecture commits to the following inherited guarantee:

Proposition 6.1 (Inherited correctness for instantiated notions). *If a notion N is sound (its expansion under a valid binding β produces a DAG whose oracle evaluation equals the intended notion semantics) and the node processor NodeProc_θ is primitive-correct on the primitives that $\text{Inst}_\beta(N)$ invokes, then NBRs’s output on N instantiated by β equals the notion’s intended semantics.*

The architectural separation isolates two questions that previous monolithic reasoning systems conflate. *Is the right algorithm applied?* concerns notion selection and binding, which is the analogy engine’s job. *Is the algorithm executed correctly?* concerns the substrate’s reliability, which is the executor’s job. The two can be analysed and improved independently.

7 Notion Induction

The library grows by inducing notions from traces of successful problem-solving. This section formalises the induction problem and identifies its key challenges.

7.1 The induction problem

Definition 7.1 (Trace corpus). A trace corpus is a finite collection $\mathcal{T} = \{(G_i, \beta_i, v_i)\}_{i=1}^M$ where each G_i is an operation DAG, β_i is the binding under which it was produced (if known), and v_i is its root output value (verified correct).

Definition 7.2 (Notion induction). The notion-induction problem is: given a trace corpus \mathcal{T} and the current library \mathcal{N} , identify candidate new notions N^* such that some non-trivial subset of \mathcal{T} is expressible as instantiations of N^* under appropriate bindings, and admit N^* to \mathcal{N} when the resulting compression of \mathcal{T} exceeds the description-length cost of N^* .

7.2 Compositional compression

Definition 7.3 (Compression value of a candidate notion). For a candidate notion N^* , let $\text{matches}(N^*, \mathcal{T})$ be the set of traces in \mathcal{T} that can be re-expressed as $\text{Inst}_\beta(N^*)$ for some valid binding β . Let $|G_i|$ be the description length of the explicit trace, $|\beta|$ the description length of a binding, and $|N^*|$ the description length of the notion itself. The compression value is

$$\Delta(N^*; \mathcal{T}) = \sum_{i \in \text{matches}(N^*, \mathcal{T})} (|G_i| - |\beta_i|) - |N^*|.$$

A candidate with $\Delta(N^*; \mathcal{T}) > 0$ is admitted; one with $\Delta \leq 0$ is rejected as unproductive. The criterion is the standard minimum-description-length criterion used in inductive program synthesis, adapted to the notion setting [13].

7.3 Type generalisation

The hardest sub-problem of notion induction is type generalisation. A trace corpus may contain traces over `Int`, `Float`, and `String` that all share structural pattern `P`. The induced notion should be parameterised over a type variable `X`, with the structural pattern `P` preserved. Detecting that the corpus is consistent with a single type-parameterised notion, rather than three separate type-specific notions, is the operation that turns library extension into genuine abstraction.

We do not give an induction algorithm here; the formal target is a procedure that, given \mathcal{T} , produces candidate type-parameterised notions with $\Delta > 0$. Approaches consistent with the framework include: unification-based generalisation over typed graph patterns, neural sequence-to-sequence notion synthesis, and wake-sleep procedures in the style of DreamCoder.

7.4 Verification of induced notions

A candidate N^* admitted to the library must be sound: applying N^* to a binding β must produce a DAG whose oracle evaluation matches the notion’s intended semantics. Two verification strategies are operationally distinct:

- *Compression-based.* N^* is admitted if it compresses a held-out portion of \mathcal{T} that was not used in its induction. This guards against overfitting to the training traces.
- *Executable-test-based.* The system generates synthetic bindings for N^* , executes the resulting DAGs against an oracle (where available) or against a corroborating notion or LLM-supplied ground truth, and admits N^* only when the success rate on synthetic tests exceeds a threshold.

The verification step is what separates notion induction from blind pattern-mining. A notion admitted without verification is at best a hypothesis; admitted with verification, it is a tool the architecture can rely on.

8 Analogical Transfer

The transfer claim that distinguishes NBRS from contemporary monolithic reasoning systems is the following: a notion learned from problems in one content domain can be applied to a problem in a different content domain whose structure aligns with the notion, without retraining and without exposure to bindings drawn from the new domain.

8.1 The transfer property, stated

Definition 8.1 (Cross-domain transfer). Let \mathcal{N} be the notion library after training on a problem distribution over content domain $\mathcal{C}_{\text{train}}$. Let P^* be a problem from content domain $\mathcal{C}_{\text{test}} \neq \mathcal{C}_{\text{train}}$ whose problem graph G_{P^*} admits a valid binding β^* to some notion $N \in \mathcal{N}$ such that $\text{Inst}_{\beta^*}(N)$ correctly solves P^* . NBRS exhibits cross-domain transfer on P^* if the analogy engine, given G_{P^*} and \mathcal{N} , returns (N, β^*) (or any other binding that yields a correct DAG).

The property has three preconditions: (i) a notion with the required structure exists in the library (the induction layer has produced it from earlier training); (ii) the structure extractor produces a problem graph for P^* that is amenable to alignment with that notion; (iii) the analogy engine identifies and binds the notion correctly. Failure to transfer can be diagnosed to one of these three preconditions.

8.2 Structure-mapping over learned representations

The analogy engine implements a form of structure-mapping [7]: it searches for a partial isomorphism between the notion’s graph and the problem’s graph that preserves relations and

respects types, prioritising mappings that align deeper relational structure over those that align mere surface features. The implementation challenge, which the literature on classical structure-mapping engines did not face, is that the notion’s variables and the problem’s entities live in learned representational spaces. The alignment is not over hand-written symbols but over typed embeddings.

We treat the alignment procedure as itself a learned component: a neural matcher that takes a notion graph and a problem graph and outputs a candidate binding, trained on (synthetic and real) pairs of (problem, applicable notion) examples. Training such a matcher is the central engineering problem of the analogy engine.

8.3 Soundness of transfer

Proposition 8.2 (Soundness of bound transfer). *Suppose $N \in \mathcal{N}$ is a sound notion and NodeProc_θ is primitive-correct on the primitives that $\text{Inst}_\beta(N)$ invokes. If the analogy engine returns a binding β such that the parameter-signature constraints of N are satisfied and the typing of $\text{Inst}_\beta(N)$ matches the problem P^* ’s required input-output types, then the value v_r returned by the executor is correct for P^* .*

Proof. Notion soundness gives correctness of $\text{Inst}_\beta(N)$ against the notion’s intended semantics. Primitive correctness gives executor correctness against the notion’s semantics. Type compatibility gives that the notion’s semantics is what P^* requires. \square

The implication: transfer failure is always attributable to one of (i) wrong notion selected, (ii) wrong binding constructed, (iii) executor error on the primitives, (iv) type-compatibility mismatch undetected by the analogy engine. The architecture localises blame, which is what makes failure analysis tractable.

9 Generalisation Properties

The structure–content factorisation has direct consequences for generalisation. Because notions are content-independent, applying a learned notion to new content does not require learning new notion parameters. Because the executor sees only local contexts, executing a notion on values drawn from a new distribution does not require depth-dependent capacity. The architecture’s generalisation properties are therefore inherited from two separable sources: notion-level generalisation (the analogy engine’s ability to recognise structural matches across domains) and execution-level generalisation (the executor’s ability to handle the bindings the analogy engine produces).

9.1 Notion-level generalisation

Proposition 9.1 (Structural transfer dominates surface transfer). *Let \mathcal{N} be a library trained on $\mathcal{C}_{\text{train}}$. For a test problem P^* from $\mathcal{C}_{\text{test}}$, the existence of a successful binding to some $N \in \mathcal{N}$ depends only on the relational structure of P^* , not on whether $\mathcal{C}_{\text{test}}$ overlaps $\mathcal{C}_{\text{train}}$.*

Proof. Notion variables abstract over types, values, and operators. Two problems with isomorphic relational structure (under a type relabelling that respects operator signatures) admit the same set of notion bindings. Domain overlap affects only what types and values appear; it does not affect what relational structures appear. Therefore the existence of a transfer-supporting notion is invariant to domain overlap, conditional on the structure of P^* being one for which an appropriate notion exists in \mathcal{N} . \square

This is the formal counterpart of the cognitive-map result: a structural representation that abstracts over content predicts transfer wherever structure repeats.

9.2 Execution-level generalisation

The execution substrate inherits the local-context invariance property of the GSRE: the per-node processor sees only the local context $(\tau_n, \text{spec}_n, (v_p)_{p \in \text{pa}_n})$, so its error rate on a node depends only on the marginal distribution of local contexts, not on the depth or shape of the surrounding graph (Theorem A.7). For instantiated notions, this means that depth, which can be arbitrary if a notion like FOLD is bound to a long sequence, does not introduce new failure modes at the executor level.

10 Empirical Programme

The architecture’s claims are empirical. The minimum programme that would test them comprises two cross-domain demonstrations, each chosen to expose a specific failure mode of contemporary LLMs that the architecture is designed to address. A short third subsection lists the diagnostic measurements that accompany the two demonstrations.

10.1 Demonstration 1: the ARC-AGI subset

Task. The Abstraction and Reasoning Corpus (ARC-AGI) [11, 12] presents tasks of the form: given two to five input–output grid pairs demonstrating an abstract transformation, infer the transformation and apply it to a held-out input grid. The benchmark was constructed specifically to require sample-efficient abstraction beyond what frontier LLMs reliably achieve. Performance on ARC-AGI is a direct test of the structure–content factorisation hypothesis: solving a task requires extracting a transformation from a handful of examples and binding it correctly to a previously unseen input.

Scope. We target a subset of ARC-AGI tasks whose transformations are expressible as compositions of grid-level primitive operations: rotate, reflect, recolour, fill connected region, count occurrences, replicate, translate, mask, overlay. The primitive library is fixed in advance. The notion-induction layer must mine, from the task’s demonstration examples, a notion whose instantiation under appropriate bindings reproduces all examples and generalises to the held-out input.

Architecture mapping. The content layer ingests the task’s grids and parses each into a typed grid representation (entities are connected regions, with attributes such as colour, shape, and position). The structure extractor produces, for each demonstration pair, a candidate relational graph mapping input regions to output regions. The notion-induction process synthesises a parameterised template from the demonstration graphs. The analogy engine binds this template to the test input. The executor produces the predicted output grid.

Comparison. Each task is evaluated against (i) a frontier LLM with chain-of-thought prompting, (ii) the same LLM augmented with retrieval over the demonstration corpus, (iii) ablations of NBRS with the notion library disabled or the analogy engine replaced by random selection, (iv) where available, results from the ARC Prize leaderboard for context.

Success criteria. The minimum bar is per-task accuracy on the chosen subset matching the frontier-LLM baseline using substantially fewer compute and substantially less reliance on prompt-engineering. The aspirational bar is exceeding the frontier LLM, particularly on tasks classified as “hard” on the ARC leaderboard. The interpretive bar is that the notions induced from one subset of tasks transfer to a held-out subset whose tasks were never used in induction. Such transfer demonstrates that the library has captured genuine compositional abstractions.

10.2 Demonstration 2: cross-discipline scientific equilibrium analogy

Task. The more ambitious demonstration tests whether a notion induced from problems in one scientific discipline transfers to a structurally analogous problem in an unrelated discipline. The architecture is trained on a corpus of mechanical-equilibrium problems (force balance on rigid bodies, spring–mass systems at rest, lever arms, fluid pressures at equilibrium), and the induced notions are evaluated on novel problems from economic equilibrium, chemical equilibrium, and population equilibrium.

The structural pattern. Each test domain shares the abstract equilibrium-reasoning structure formalised in Example 4.5: identify two (or more) opposing influences, write them as functions of a quantity, set them equal at the equilibrium point, and solve. The surface content differs entirely. Successful transfer requires the analogy engine to recognise the structural match across surface domains and to bind the equilibrium notion’s variables to the new domain’s entities.

Test problems. We evaluate transfer on four discipline pairs, each of which carries an equilibrium structure but no shared surface content with the mechanical training corpus:

- *Supply–demand market equilibrium.* Find the price at which a supply function and a demand function are equal.
- *Chemical equilibrium under perturbation.* Predict the direction of shift in concentrations when a reaction at equilibrium is perturbed (Le Chatelier).
- *Predator–prey steady state.* Compute the fixed-point populations of a Lotka–Volterra-style system.
- *Nash equilibrium in a 2-player normal-form game.* Identify the strategy pair where each player’s choice is a best response to the other.

Architecture mapping. The content layer (the LLM) parses each problem statement into entities, types, and the goal. The structure extractor produces a problem graph whose nodes are quantities and whose edges encode the relevant relations (the supply function depending on price, the demand function depending on price, the goal of finding their intersection). The analogy engine matches this problem graph against EQUILIBRIUM and proposes a binding: α_+ to the supply function, α_- to the demand function, X to the price type, x to the price variable. The executor runs the bound DAG.

Comparison. The same three baselines as Demonstration 1, plus a fourth: a frontier LLM with explicit chain-of-thought scaffolding tuned for cross-domain analogical reasoning (e.g., explicit prompts asking the model to map the problem to a familiar analogue).

Success criteria. The minimum bar is exceeding the frontier-LLM baseline on at least one of the four test domains while not regressing on within-discipline (mechanical-equilibrium) test problems. The aspirational bar is exceeding the frontier LLM across all four test domains, with a notion-library size that did not include any economic, chemical, biological, or game-theoretic training examples. A negative result is interpretable: it would localise to a specific component (structure-extractor parsing of the new domain, analogy-engine match against the equilibrium notion, executor handling of the bound primitives) and the diagnostic measurements of Section 10.3 would identify which.

10.3 Diagnostics and ablations

Each demonstration is accompanied by the following measurements:

- *Notion-induction efficiency.* The number of training traces required before each notion enters the library at the verification threshold. Reported as a learning curve over the trace corpus size.
- *Analogy-engine accuracy.* The fraction of test problems for which the analogy engine returns the intended notion and binding, decomposed by demonstration tier. Errors classified as wrong-notion, wrong-binding, or no-match.
- *Per-primitive error rate and correlation structure.* The empirical error rate of `NodeProc θ` on the primitives that appear in test DAGs, together with the empirical covariance of error events across invocations sharing a context shape. This calibrates the executor’s depth-decoupling claim (Theorem A.6).
- *Marginal-invariance diagnostic.* A KL or MMD divergence between the empirical context marginals of training-derived DAGs and test-derived DAGs. A small divergence supports the depth-generalisation claim of Corollary A.8; a large divergence quantifies the gap.
- *Notion-library ablation.* For each demonstration, the performance of NBRs with the notion library replaced by an empty library (forcing the system to solve each problem from scratch via the executor) and with the analogy engine replaced by random notion selection. The drop between full NBRs and these ablations attributes performance to the library and the alignment, respectively.

A negative result on either headline demonstration that is not accompanied by an explanatory diagnostic should be interpreted as a falsification of the architectural hypothesis at its current scale. A negative result accompanied by a localising diagnostic identifies the component to improve.

11 Connections to Prior Work

Cognitive maps and entorhinal–hippocampal computation. The architectural commitment to structure–content factorisation is taken from the Tolman–Eichenbaum Machine [1] and the cognitive-map literature it formalises [2, 3, 4]. NBRs operationalises the same factorisation in a procedural-reasoning setting: notions are structural codes, bindings are conjunctive codes, and library growth is analogous to entorhinal consolidation.

Structure-mapping theory. The analogy engine is the operational counterpart of Gentner’s Structure-Mapping Theory [7] and its computational implementation, the Structure-Mapping Engine [8]. The novel commitment of NBRs is to perform structure-mapping over learned (rather than hand-symbolic) representations of types and operators.

Library learning and inductive program synthesis. Notion induction (Section 7) is closest to library learning as developed in DreamCoder [13] and the broader inductive-program-synthesis literature [14]. NBRs extends this to typed parameterisation over operators and to integration with an LLM content layer, both of which the existing library-learning systems do not address.

Neural module networks. Andreas et al. [15] compose computation graphs from a fixed module library at inference time. NBRs generalises this in two directions: the library is grown (not fixed), and the modules carry typed parameterisation that supports cross-domain transfer.

Consolidation in neural models. The replay-driven consolidation process is informed by recent computational models of hippocampal replay and memory consolidation [5, 6].

LLM-augmented neurosymbolic reasoning. Recent work on neurosymbolic LLM augmentation [9] typically translates problems into a fixed logical form for an external solver. NBRs differs in maintaining a learned, growing library of typed notions; the content layer’s role is parsing, not full reduction to logic.

ARC-AGI and analogy benchmarks. Chollet’s ARC-AGI benchmark [11, 12] and the analogical-reasoning stress tests of Lewis and Mitchell [9] and Webb et al. [10] are the empirical targets of Section 10.1.

The executor substrate. The Graph-Structured Recurrent Executor and its operational guarantees are summarised in Section 6 and detailed in Appendix A.

12 Discussion

The architectural bet of NBRs is that the transfer gap separating contemporary LLMs from human reasoning is fundamentally about representation: the absence of an explicit, content-independent layer that stores transferable procedures. The neuroscience evidence for such a layer in biological reasoners is strong and recent. The classical symbolic literature on analogical reasoning has long argued that structure-mapping is the right computational model. What has been missing is an integration that combines the abstraction power of cognitive-map-style factorisation, the procedural depth of program-induction-style library learning, and the perceptual breadth of LLM-style content processing.

Two design commitments of NBRs are non-obvious and warrant emphasis. First, the notion library is the architectural locus of learning at the structural level: the LLM is not asked to learn or transfer procedures, only to parse and generate. This separation is what gives the system the prospect of sample-efficient transfer. Second, the execution substrate is deliberately conservative: the GSRE guarantees only what a deterministic typed-DAG evaluator can guarantee. The interesting failure modes of the system are therefore localised to notion induction and analogical alignment, not to execution. This is the right place for them to be.

The empirical risk is real. Notion induction in non-trivial domains is open; structure-mapping in learned representations is open; integration with LLMs at the content layer is open. The architecture is not a guaranteed solution. It is a coherent place to spend effort, with each architectural commitment tied to a specific scientific hypothesis and a specific failure-diagnosis pathway.

Empirical status. This document is the architectural and theoretical foundation; the empirical programme of Section 10 has since been partially carried out, with mixed results summarised here. The cross-discipline equilibrium demonstration succeeded: a multi-step procedure discovered by search in one discipline transferred exactly to three unseen ones. The ARC-AGI study produced high precision but low coverage and no induction gain at the scale tested, and was run without a language-model baseline, so it supports no comparative claim. A depth-stratified computation study, intended to exhibit the capacity–depth decoupling against a degrading baseline, instead found that an open 20B model computed the chains accurately itself; the decoupling theorem holds as stated (the executor stayed exact at every depth), but the advantage it was meant to confer was not observed because the baseline did not degrade in that regime. The theorems below are unaffected by these outcomes, being statements about the executor rather than about any baseline; what the experiments revise is the expectation that the decoupling yields a measured advantage in the regimes tested so far.

A The Execution Substrate: Full Specification

This appendix specifies the GSRE in detail and proves its operational properties.

A.1 Node states

During execution, every node $n \in V$ is in one of five states drawn from $\mathcal{Q} = \{\text{PENDING}, \text{READY}, \text{EXECUTING}, \text{RESOLVED}, \text{FAILED}\}$, written $\text{state}(n) \in \mathcal{Q}$. Once $\text{state}(n) = \text{RESOLVED}$, the associated value $v_n \in \mathcal{V}_{\text{out}_{\tau_n}}$ is fixed.

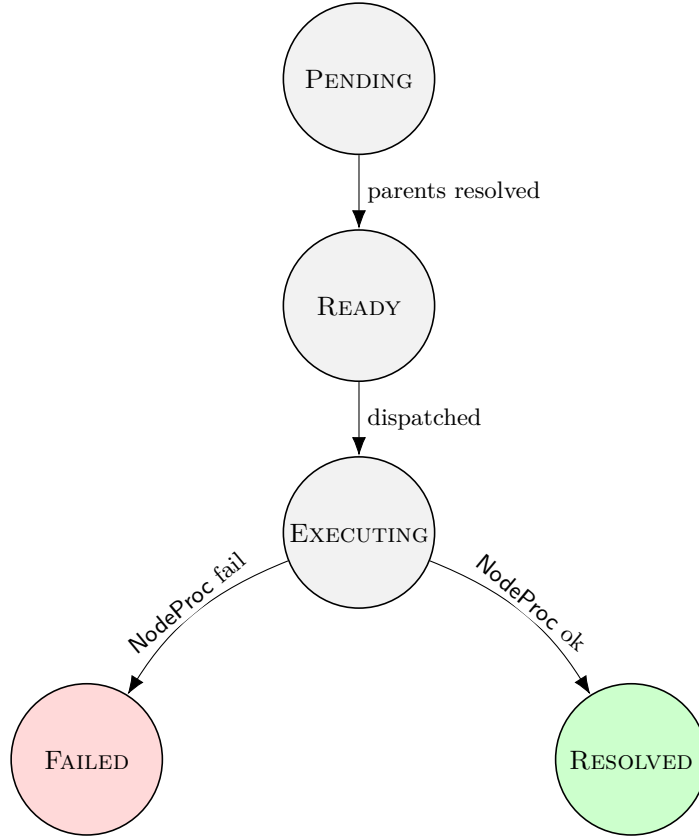


Figure 2: The GSRE node-state machine. $\text{PENDING} \rightarrow \text{READY}$ is gated by parent state; $\text{READY} \rightarrow \text{EXECUTING}$ by dispatch; $\text{EXECUTING} \rightarrow \text{RESOLVED}$ or FAILED by the outcome of the processor call.

A.2 The traversal engine

A.3 Operational properties

Theorem A.1 (Determinism). *Let NodeProc_θ be a deterministic function of its input. For any well-formed operation DAG G and leaf assignment σ , the value assignment returned by Algorithm 1 is uniquely determined by G , σ , and θ .*

Theorem A.2 (Order independence). *Under the hypothesis of Theorem A.1, the value assignment produced is invariant under any nonempty subset $\mathcal{B} \subseteq R$ chosen at each iteration.*

Theorem A.3 (Termination). *Algorithm 1 terminates in at most $|V|$ iterations, with at most $|V| - |V_{\text{leaf}}|$ processor invocations.*

Theorem A.4 (Soundness against the oracle). *If NodeProc_θ is primitive-correct on every primitive it sees, the value v_r returned by Algorithm 1 equals the oracle ground-truth answer for any reasoning task (G, σ) .*

Algorithm 1 The Graph-Structured Recurrent Executor

Require: Well-formed operation DAG $G = (V, r)$ with leaf assignment σ ; processor NodeProc_θ .

Ensure: A (possibly partial) value assignment $v : V \rightarrow \mathcal{V}$.

```
1: for each  $n \in V$ :  $\text{state}(n) \leftarrow \text{PENDING}$ 
2: for each leaf  $n \in V$ :  $v_n \leftarrow \sigma(n)$ ;  $\text{state}(n) \leftarrow \text{RESOLVED}$ 
3: loop
4:    $R \leftarrow \{n \in V : \text{state}(n) = \text{PENDING} \text{ and all } p \in \text{pa}_n \text{ have } \text{state}(p) = \text{RESOLVED}\}$ 
5:   for each  $n \in R$ :  $\text{state}(n) \leftarrow \text{READY}$ 
6:   if  $R = \emptyset$  then
7:     if  $\text{state}(r) = \text{RESOLVED}$  then
8:       return  $v$ 
9:     else if  $\text{state}(r) = \text{FAILED}$  or no PENDING node has all-RESOLVED parents then
10:      return  $v$ 
11:     end if
12:   end if
13:    $\mathcal{B} \leftarrow R$ 
14:   for each  $n \in \mathcal{B}$ :  $\text{state}(n) \leftarrow \text{EXECUTING}$ 
15:    $\hat{v}_\mathcal{B} \leftarrow \text{NodeProc}_\theta(\{(\tau_n, \text{spec}_n, (v_p)_{p \in \text{pa}_n}) : n \in \mathcal{B}\})$ 
16:   for  $n \in \mathcal{B}$  do
17:     if  $\hat{v}_{\mathcal{B},n} \neq \perp$  then
18:        $v_n \leftarrow \hat{v}_{\mathcal{B},n}$ ;  $\text{state}(n) \leftarrow \text{RESOLVED}$ 
19:     else
20:        $\text{state}(n) \leftarrow \text{FAILED}$ 
21:     end if
22:   end for
23: end loop
```

Proposition A.5 (Per-step bounded input). *The input length presented to NodeProc_θ at any node n is bounded by $|\text{Enc}(\tau_n)| + |\text{Enc}(\text{spec}_n)| + k_{\tau_n} \cdot L_{\max}$, independent of $|V|$ and $\text{depth}(G)$.*

Theorem A.6 (Capacity–depth decoupling under independent primitive errors). *Suppose NodeProc_θ has per-invocation error rate $\leq \varepsilon$ and errors at distinct invocations are independent. Then $\mathbb{P}[v_r = \hat{v}_r] \geq (1 - \varepsilon)^{|V| - |V_{\text{leaf}}|}$.*

Theorem A.7 (Local-context invariance). *For any two well-formed DAGs G, G' and any nodes $n \in V_G, n' \in V_{G'}$ with the same local context $(\tau_n, \text{spec}_n, (v_p)_{p \in \text{pa}_n})$, the executor produces identical values.*

Corollary A.8 (Depth generalisation under marginal invariance). *Under marginal invariance of local contexts across depth (Theorem A.7) and the geometric decay of Theorem A.6, the executor's expected accuracy on out-of-distribution depths is bounded below by a function depending only on the size distribution of the test graphs.*

Proofs are direct from the structure of Algorithm 1 and parallel earlier treatments.

Remark A.9 (Independence is the load-bearing assumption in Theorem A.6). The independence assumption is unrealistic in shared-parameter models. The geometric bound is best read as a baseline against which empirical error-correlation structure is measured, not as a tight prediction. The unconditional property of the architecture is the per-step bounded input (Proposition A.5); the conditional property is the geometric decay (Theorem A.6). The empirical programme of Section 10 includes measurement of the actual error-correlation structure.

References

- [1] J. C. R. Whittington, T. H. Muller, S. Mark, G. Chen, C. Barry, N. Burgess, and T. E. J. Behrens. The Tolman–Eichenbaum Machine: Unifying space and relational memory through generalization in the hippocampal formation. *Cell*, 183(5):1249–1263, 2020.
- [2] T. E. J. Behrens et al. What is a cognitive map? Organizing knowledge for flexible behavior. *Neuron*, 100(2):490–509, 2018.
- [3] A. O. Constantinescu, J. X. O’Reilly, and T. E. J. Behrens. Organizing conceptual knowledge in humans with a gridlike code. *Science*, 352(6292):1464–1468, 2016.
- [4] J. C. R. Whittington, W. Dorrell, S. Ganguli, and T. E. J. Behrens. Disentanglement with biological constraints: A theory of functional cell types. *ICLR*, 2023.
- [5] E. Spens and N. Burgess. A generative model of memory construction and consolidation. *Nature Human Behaviour*, 8:526–543, 2024.
- [6] D. Kumaran, D. Hassabis, and J. L. McClelland. What learning systems do intelligent agents need? Complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20(7):512–534, 2016.
- [7] D. Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.
- [8] B. Falkenhainer, K. D. Forbus, and D. Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41(1):1–63, 1989.
- [9] M. Lewis and M. Mitchell. Evaluating the robustness of analogical reasoning in large language models. *arXiv:2411.14215*, 2024.
- [10] T. Webb, K. J. Holyoak, and H. Lu. Emergent analogical reasoning in large language models. *Nature Human Behaviour*, 7(9):1526–1541, 2023.
- [11] F. Chollet. On the measure of intelligence. *arXiv:1911.01547*, 2019.
- [12] F. Chollet et al. ARC Prize: Updates and the path forward. *arXiv:2412.04604*, 2024.
- [13] K. Ellis et al. DreamCoder: Bootstrapping inductive program synthesis with wake–sleep library learning. *PLDI*, 2021.
- [14] S. Gulwani, O. Polozov, and R. Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1–2), 2017.
- [15] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. *CVPR*, 2016.