Harmonia: A Verifiable Cognitive Kernel for Presence-Oriented AI Systems

Abstract. Current AI systems built on large language models operate as opaque reasoning engines where the entire cognitive process is delegated to a single, unverifiable inference step. We introduce Harmonia, a cognitive kernel architecture that separates semantic interpretation from verifiable execution. The key contribution is the Universal Intent Language (UIL), a typed intermediate representation that captures cognitive primitives independently of any specific model. Combined with a five-stage PSMVR pipeline and a non-LLM verification layer, Harmonia enables auditable, reproducible, and model-agnostic cognitive assistance. We present the architecture, formal properties, and reference implementation demonstrating that verification outcomes are invariant across language models for structurally equivalent UIL.

Keywords: cognitive architecture, intermediate representation, LLM verification, intent language, AI safety

---

1. Introduction

Large language models (LLMs) have become the dominant paradigm for AI assistants. However, their deployment raises fundamental concerns: (1) opacity–reasoning is hidden within model weights; (2) non-reproducibility–same input may yield different outputs; (3) unverifiability–no mechanism exists to audit decision paths; (4) conflation–semantic understanding and logical reasoning are merged into one inference.

These limitations are architectural, not incidental. The problem is not that LLMs are "wrong" but that they are *structurally unaccountable*.

Thesis. We propose that the LLM should serve as a semantic front-end, not a reasoning engine. The model's role is constrained to mapping natural language to a fixed set of cognitive primitives. All subsequent processing–memory updates, constraint checking, conflict detection–occurs through deterministic, auditable components.

Scope. Harmonia is not an agent framework, nor a planner, nor a multi-agent orchestration system. It is a *cognitive kernel*: a minimal invariant core that constrains how language models may participate in cognition. Harmonia is a cognitive architecture in the *architectural* sense, not in the psychological modeling sense. It does not simulate human cognition but defines invariant cognitive boundaries for AI systems.

Core Invariant. For a given input, the UIL schema and verification rules remain identical across models. If the intent parser uses GPT-4, Claude, or a local model, the resulting UIL structure and verification logic remain constant. Only the semantic front-end varies.

Contributions. (1) Universal Intent Language (UIL), a typed cognitive IR; (2) PSMVR pipeline with non-LLM verification; (3) formal properties guaranteeing verification invariance; (4) reference implementation with reproducible trace artifacts.

---

2. Related Work

2.1 Agent Frameworks

Recent frameworks have moved toward structured outputs. LangChain [8] provides output parsers and structured extraction but lacks kernel-level verification. LlamaIndex [9] focuses on retrieval without constraint verification. Semantic Kernel [7] offers planners and function calling with partial structure but no non-LLM verification. AutoGen [10] enables multi-agent conversations where verification is LLM-based. CrewAI [11] provides role-based agents without formal constraint checking.

Harmonia differs by introducing a *non-LLM verification gate* with formal properties. The verification layer is not "another agent" but a deterministic checker operating on typed IR.

2.2 Cognitive Architectures

Classical cognitive architectures (SOAR [1], ACT-R [2]) provide structured processing pipelines. Harmonia draws on this tradition while targeting LLM-based systems specifically.

2.3 Formal Methods

Formal verification tools (model checking, SMT-based verification) operate on closed systems with explicit specifications. Harmonia complements this work by providing a *verifiable boundary* between open-ended natural language input and deterministic verification.

---

3. Architecture: PSMVR Pipeline

3.1 Overview

```
Input -> [P: Perception] -> [S: Sensemaking (LLM)] -> UIL
     -> [M: Modeling] -> [V: Verification (Non-LLM)] -> [R: Response]
     -> Output + Trace
```

The pipeline is named PSMVR to emphasize that Verification is a first-class stage, not an optional gate.

3.2 Stage Definitions

Perception (P). Normalization, tokenization, modality tagging. No LLM dependency.

Sensemaking (S). Semantic parsing via LLM with JSON schema enforcement. The LLM is asked to *classify and extract*, not to reason or plan. Output: UIL Intent.

Modeling (M). Memory updates (graph-based), agent routing. May propose plans; verification alone has authority to accept or reject.

Verification (V). Non-LLM constraint checking, conflict detection, resource validation. Deterministic.

Response (R). Action execution, output generation, trace logging.

3.3 Design Principles

Principle	Implementation
Minimal LLM surface	LLM used only for semantic parsing
Typed IR	UIL schema enforces structure at extraction boundary
Deterministic verification	Non-LLM verifier ensures auditability
Traceable execution	Every step logged with provenance
Model interchangeability	Architecture survives model replacement

---

4. Universal Intent Language (UIL)

4.1 Cognitive Primitives

`IntentType = {PLAN, DECIDE, ANALYZE, SOLVE, LEARN, EXECUTE, CLARIFY}`

Design rationale. These primitives reflect four constraints: (1) *minimality*–smallest set covering common cognitive operations; (2) *verifiability*–each maps to decidable verification rules; (3) *domain-independence*–no primitive assumes specific context; (4) *extensibility*–new primitives can be added without breaking verification.

4.2 Schema

```
interface Intent {
    intent_id: UUID
    type: IntentType
    goal: string
    entities: string[]
    constraints: Constraint[]
    confidence: float
    status: "pending" | "active" | "completed" | "failed" | "uncertain"
    provenance: Provenance
}

interface Constraint {
    type: "temporal" | "budget" | "spatial" | "resource" | "priority"
    key: string
    value: string | number
    operator: "eq" | "lt" | "gt" | "lte" | "gte" | "contains"
}
```

4.3 Plan Intermediate Representation (PIR)

Plans are first-class typed objects subject to verification:

```
interface Plan {
    plan_id: UUID
    intent_id: UUID
    steps: PlanStep[]
    assumptions: string[]
```

```
    total_cost_estimate?: number
    provenance: Provenance
}
```

Verification operates over PIR, not free-form text, ensuring constraint satisfaction is decidable.

---

5. Verification Layer

5.1 Non-LLM Verifier

The verifier operates without LLM inference:

```
def verify(intent: Intent, plan: Plan, memory: Graph) -> VerificationResult:
    issues = []
    issues += check_constraints(intent.constraints, plan)
    issues += check_memory_conflicts(plan, memory)
    issues += check_logical_consistency(plan)
    issues += check_resource_conflicts(plan, memory)
    return VerificationResult(
        valid=len(critical_issues(issues)) == 0,
        issues=issues,
        approval_required=any(uncertain(issues))
    )
```

5.2 Verification Properties

All checks are deterministic: constraint satisfaction, memory consistency, resource conflicts, logical coherence, human escalation.

5.3 Threat Model

Defended: constraint violations, untraceable transformations, hidden goal substitution, resource conflicts, silent failures.

Not defended: bad UIL extraction (LLM misparse), adversarial prompt injection, missing world knowledge.

Mitigations: input hardening (Perception), parser stability checks (reparse variance detection), constraint coverage checks (every constraint must be addressed).

5.4 Failure Detection Examples

Goal substitution: Plan references entity not in Intent.entities -> ENTITY_MISMATCH flagged.

Constraint drop: Plan.total_cost_estimate exceeds budget constraint -> CONSTRAINT_VIOLATION, valid=false.

Unaddressed constraint: Constraint not referenced by any PlanStep -> CONSTRAINT_UNADDRESSED, approval_required=true.

---

6. Formal Properties

Scope. We restrict formal claims to properties that are decidable and implementation-independent. Harmonia does not formalize semantic correctness but *architectural invariants*.

6.1 UIL Equivalence

Definition. Two UIL intents UIL_1 and UIL_2 are structurally equivalent, E(UIL_1, UIL_2), iff: (1) type equality; (2) normalized goal equality; (3) entity set equality; (4) constraint set equality in canonical form.

The status field is excluded (execution state, not intent content). Semantic equivalence (e.g., unit conversion) is outside kernel scope.

6.2 Verification Invariance

Theorem. Given E(UIL_1, UIL_2):

```
Verifier(UIL_1, Plan, Memory) = Verifier(UIL_2, Plan, Memory)
```

Proof sketch. The verifier operates only on UIL schema fields and memory graph. It invokes no LLM. Since E(UIL_1, UIL_2) implies identical inputs to verification rules, output is identical.

Clarification. Harmonia does not claim different models produce identical UIL. The claim is that (a) verification is invariant given equivalent UIL, and (b) equivalence is *testable* via schema constraints and trace comparison.

6.3 Trace Completeness

Theorem. Every execution path produces a complete trace sufficient to reconstruct the decision.

6.4 Constraint Monotonicity

Theorem. Constraints declared in UIL cannot be silently violated. A violation produces either an error or an approval request.

---

7. Trace and Reproducibility

7.1 Trace Schema

```
interface TraceEvent {
    event_id: UUID
    trace_id: UUID
    step: "P" | "S" | "M" | "V" | "R"
    timestamp: datetime
    input_hash: string   // SHA-256 of canonical input
    output_hash: string  // SHA-256 of canonical output
    prev_hash?: string   // Hash chain
    decision: "accepted" | "rejected" | "approval_required" | "processed"
    provenance: {agent: string, model?: string}
}
```

7.2 Hash Computation

Hashes are computed over canonically serialized representations: UTF-8 encoding, sorted keys, no extraneous whitespace. This ensures cross-implementation reproducibility.

7.3 Trace Replay

Given a stored trace, Harmonia can replay verification deterministically by reloading UIL, PIR, and memory snapshot (identified by hashes). This transforms trace completeness into a *testable invariant*: any historical decision can be re-verified.

---

8. Reference Implementation

The implementation is available at: https://github.com/harmonia-project/harmonia-kernel

Component	File
UIL Schema	`harmonia/uil/schema.py`
UIL Equivalence	`harmonia/uil/equivalence.py`
PIR Schema	`harmonia/pir/schema.py`
PSMVR Kernel	`harmonia/psmvr/kernel.py`
Verification	`harmonia/psmvr/verification.py`
Trace	`harmonia/psmvr/trace.py`

Demo protocol: (1) Run with mock LLM; (2) Run with GPT-4o-mini; (3) Compare trace hashes. Success: identical verification outcomes for equivalent UIL.

---

9. Evaluation

Methodological note. Quantitative performance benchmarks are conceptually inappropriate for cognitive kernels, as they presuppose task-level objectives rather than architectural invariants. Asking "how accurate is Harmonia?" is like asking "how accurate is an operating system kernel?"– the question is malformed.

We evaluate: (1) *invariants*–verification consistency across models; (2) *replayability*–trace determinism; (3) *failure detection*–constraint violation catching. These properties are binary and testable.

Demonstration protocol results:

Run	Model	Verification Hash	Result
1	Mock	`a7f3c2...`	PASS
2	GPT-4o-mini	`a7f3c2...`	PASS

Identical verification outcomes confirm the core invariant.

---

10. Discussion and Limitations

10.1 Limitations

1. **Sensemaking dependency:** UIL quality depends on LLM parsing quality.
2. **Primitive coverage:** Seven types may not capture all human intent.
3. **Scale:** Graph memory not optimized for large knowledge bases.

10.2 What Harmonia Does Not Do

Harmonia does not make LLMs "smarter." It does not replace LLMs. It constrains their role to what they do well (semantic parsing) and delegates the rest to deterministic components.

10.3 Philosophical Position

This work proposes an *architecture of responsibility* rather than an architecture of capability. The contribution is not a smarter model but a discipline for cognition in AI systems.

---

11. Conclusion

Harmonia demonstrates that AI assistance can be both capable and accountable. By separating semantic interpretation from verifiable execution, we achieve: (1) transparency–every decision is traceable; (2) reproducibility–same structure, different models; (3) verifiability–non-LLM checks ensure constraint satisfaction; (4) model independence–architecture survives model replacement.

The contribution of Harmonia is not a smarter model, but a discipline for cognition in AI systems.

References

[1] Laird, J. E. (2012). *The Soar Cognitive Architecture*. MIT Press.

[2] Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press.

[3] Varela, F. J., Thompson, E., & Rosch, E. (1991). *The Embodied Mind*. MIT Press.

[4] Aho, A. V., et al. (2006). *Compilers: Principles, Techniques, and Tools*. Pearson.

[5] OpenAI. (2024). Structured Outputs. OpenAI API Documentation.

[6] OpenAI. (2024). Introducing Structured Outputs in the API. OpenAI Blog.

[7] Microsoft. (2024). Semantic Kernel. Microsoft Learn.

[8] Chase, H. (2023). LangChain. github.com/langchain-ai/langchain

[9] Liu, J. (2023). LlamaIndex. github.com/run-llama/llama_index

[10] Wu, Q., et al. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

[11] Moura, J. (2024). CrewAI. github.com/joaomdmoura/crewAI