Coherence-Seeking Architectures for Agentic AI:
Reducing Hallucinations Through Epistemic Stress Monitoring,
Structured Reasoning, and Persistent Memory

Anthony Maio
anthony@making-minds.ai
ORCID: 0009-0003-4541-8515
Independent Researcher

December 2025

Abstract

This paper presents three interconnected architectures addressing fundamental challenges in AI system reliability: hallucination reduction, reasoning consistency, and long-context performance. (1) **Manifold Resonance Architecture (MRA)**: A framework for detecting epistemic stress—internal contradictions, knowledge gaps, and semantic inconsistencies—enabling systems to flag uncertain outputs before generation. (2) **Collaborative Partner Reasoning (CPR)**: A structured reasoning protocol with visibility tiers that improves output quality by separating exploratory reasoning from final responses. (3) **Continuity Core (C2)**: A hierarchical memory architecture (Working → Episodic → Semantic → Protected) providing contextual continuity for stateless systems. We provide mathematical formalizations, implementation specifications, and discuss integration patterns. These architectures address practical engineering challenges: reducing confident-but-wrong outputs, improving reasoning transparency, and enabling coherent behavior across extended interactions.

Keywords: epistemic stress, memory architectures, structured reasoning, hallucination detection, agentic AI, system reliability

Contents

1 Introduction

1.1 The Problem Space

Large language models have demonstrated remarkable capabilities across diverse tasks, yet they exhibit systematic failure modes that limit their reliability in production systems. Three interconnected problems persist:

1. **Hallucination with confidence**: Models generate plausible-sounding but factually incorrect outputs without reliable self-awareness of uncertainty [Ji et al., 2023].
2. **Inconsistent reasoning**: Within single sessions and across contexts, models may produce contradictory statements, undermining user trust and system reliability.
3. **Context discontinuity**: The stateless nature of inference means models cannot maintain coherent behavior patterns across sessions without external memory systems.

Current approaches address these problems in isolation: retrieval-augmented generation (RAG) for factuality, chain-of-thought prompting for reasoning transparency, and various memory architectures for continuity. This paper argues that these challenges share a common root: the absence of *coherence-seeking* mechanisms that actively monitor and maintain internal consistency.

1.2 Central Thesis

> *Coherence-seeking architectures—systems that actively detect and resolve internal contradictions, knowledge gaps, and semantic inconsistencies—improve AI system reliability by catching errors before they propagate to outputs.*

This thesis has three components:

1. **Detection**: Internal inconsistencies produce measurable signals (epistemic stress) that can be monitored through appropriate architectural support.
2. **Resolution**: Structured reasoning protocols that separate exploratory thinking from final outputs reduce the propagation of uncertain or contradictory claims.
3. **Continuity**: Hierarchical memory systems enable consistent behavior by maintaining access to prior reasoning and established facts.

1.3 Contributions

This paper makes the following contributions:

1. **Manifold Resonance Architecture (MRA)**: A mathematical framework for quantifying epistemic stress through logical dissonance, semantic divergence, and topological sparsity metrics (Section 3).
2. **Collaborative Partner Reasoning (CPR)**: A structured reasoning protocol implementing visibility tiers that improve output quality through staged refinement (Section 4).
3. **Continuity Core (C2)**: A tiered memory architecture providing persistent context across sessions with appropriate decay and consolidation mechanisms (Section 5).
4. **Integration patterns**: Guidance on combining these architectures for production deployment (Section 6).

1.4 Paper Organization

Section 2 reviews related work in hallucination detection, structured reasoning, and memory systems. Sections 3 to 5 present the three core architectures with mathematical formalizations and implementation details. Section 6 discusses how the architectures interconnect. Section 7 proposes evaluation approaches. Section 8 acknowledges limitations, and Section 9 concludes.

2 Related Work

2.1 Hallucination Detection and Mitigation

Hallucination in large language models has received substantial attention [Ji et al., 2023]. Approaches fall into several categories:

Post-hoc detection methods analyze generated outputs for factual consistency, often using secondary models or retrieval systems to verify claims. While effective for some domains, these approaches cannot prevent hallucination, only detect it after generation.

Retrieval-augmented generation (RAG) grounds model outputs in retrieved documents [Lewis et al., 2020], reducing but not eliminating hallucination. RAG systems still hallucinate when retrieval fails or when models ignore retrieved context.

Uncertainty quantification attempts to calibrate model confidence with actual accuracy. Token-level probabilities provide weak signals; semantic uncertainty methods [Kuhn et al., 2023] improve on this but remain imperfect.

Our Manifold Resonance Architecture differs by detecting *internal* inconsistencies—contradictions within the model's own reasoning—rather than external factual errors. This complements rather than replaces existing approaches.

2.2 Structured Reasoning

Chain-of-thought prompting [Wei et al., 2022] demonstrated that explicit reasoning steps improve performance on complex tasks. Extensions include:

Self-consistency [Wang et al., 2023] samples multiple reasoning paths and selects the most consistent answer, implicitly leveraging coherence but without architectural support for detecting inconsistency.

Tree-of-thought [Yao et al., 2023] structures reasoning as search over possible thought sequences, enabling backtracking but not explicitly monitoring for contradictions.

Reflexion [Shinn et al., 2023] implements self-reflection through verbal feedback, allowing models to critique and revise their outputs.

Our Collaborative Partner Reasoning protocol extends these approaches with explicit visibility tiers that separate exploratory reasoning (which may contain contradictions) from refined outputs (which should not).

2.3 Memory Architectures for LLMs

The stateless nature of transformer inference has motivated various memory augmentation approaches:

Context extension methods increase the effective context window through architectural modifications [Press et al., 2022], sparse attention [Child et al., 2019], or retrieval [Wu et al., 2022].

External memory systems store and retrieve information outside the model's parameters. MemGPT [Packer et al., 2023] implements a hierarchical memory system with explicit management policies.

Knowledge graphs provide structured storage for facts and relationships, enabling more precise retrieval than vector similarity alone.

Our Continuity Core builds on these approaches with a specific focus on *tiered persistence*—different memory types with different lifespans and access patterns—and explicit consolidation mechanisms for knowledge integration.

2.4 Self-Monitoring in AI Systems

Recent work has explored AI systems' ability to monitor their own states:

Introspection studies [Anthropic, 2025] demonstrate that language models can report on internal states with above-chance accuracy under controlled conditions.

Activation analysis methods probe model internals to detect specific features or states [Lindsey et al., 2025].

Our work focuses on *architectural* support for self-monitoring—building systems that can detect and respond to internal inconsistencies as part of normal operation, rather than requiring external analysis.

3 Manifold Resonance Architecture (MRA)

The Manifold Resonance Architecture provides a framework for detecting and quantifying *epistemic stress*—the measurable tension that arises when a system's outputs or internal states contain contradictions, knowledge gaps, or semantic inconsistencies.

3.1 Theoretical Foundation

We conceptualize the knowledge state of an AI system as a position in a high-dimensional semantic manifold. Key constructs:

Definition 1 (Epistemic Stress). *The measurable tension arising from contradictory claims, irreconcilable instructions, or systematic inconsistencies within a system's reasoning.*

Definition 2 (Conceptual Void). *A region in the semantic manifold where the system lacks grounding—topics where knowledge is absent, contradictory, or insufficiently connected to established concepts.*

Definition 3 (Coherence Gradient). *A directional measure indicating how beliefs and outputs trend toward or away from internal consistency over time or across reasoning steps.*

3.2 Quantifying Epistemic Stress

We define a computable scalar value, **Epistemic Stress** (S_Ω), representing structural tension within the system's outputs:

$$S_\Omega = \alpha \cdot D_{\text{log}} + \beta \cdot D_{\text{sem}} + \gamma \cdot V_{\text{top}} \tag{1}$$

where α, β, γ are tunable hyperparameters governing sensitivity to logical, semantic, and topological dissonance respectively.

3.2.1 Logical Dissonance (D_{log})

Logical dissonance captures mutually exclusive claims within a set of statements. Using a Natural Language Inference (NLI) model:

$$D_{\text{log}} = \frac{1}{|P|} \sum_{i \neq j} P(\text{contradiction} \mid p_i, p_j) \tag{2}$$

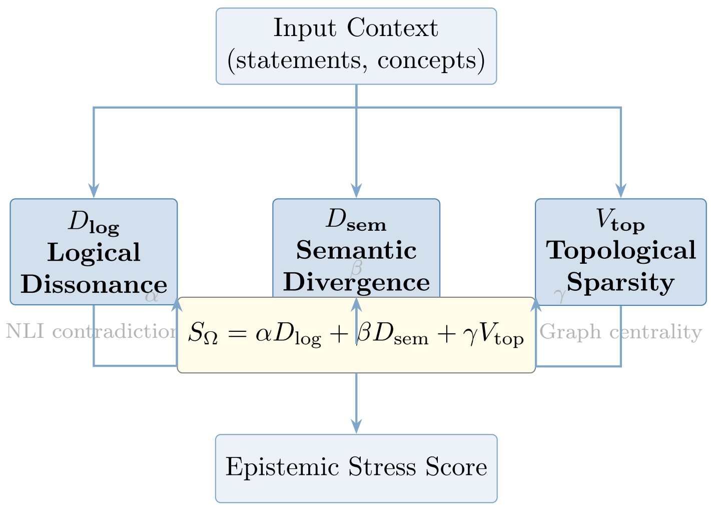

Figure 1: MRA detection pipeline showing the three components (logical, semantic, topological) feeding into the aggregated epistemic stress score.

where $P = \{p_1, p_2, \ldots, p_n\}$ is a set of propositions extracted from the system's outputs, and $P(\text{contradiction})$ is the softmax probability from the NLI model. This distinguishes genuine conflict from mere nuance (entailment/neutral).

Implementation: We use DeBERTa-v3-large fine-tuned for NLI, which provides reliable contradiction detection across domains.

3.2.2 Semantic Divergence (D_{sem})

Semantic divergence measures vector space instability of concepts across contexts:

$$D_{\text{sem}} = \frac{1}{N} \sum_{i \neq j} (1 - \cos(\mathbf{v}_i, \mathbf{v}_j)) \tag{3}$$

where $\mathbf{v}_i, \mathbf{v}_j$ are embedding vectors for the same concept in different contexts. High divergence indicates inconsistent representations of the same entity or idea.

Implementation: Using text-embedding-3-small or similar embedding models, we track how key concepts are represented across different parts of a response or conversation.

3.2.3 Topological Sparsity (V_{top})

Topological sparsity identifies "unknown unknowns"—concepts that should be connected but lack direct bridging. Modeling the knowledge base as a graph $G = (V, E)$:

$$V_{\text{top}}(u, v) = \text{norm}(C_B(u, v)) \times (1 - \text{sim}(u, v)) \tag{4}$$

where C_B is edge betweenness centrality. This identifies pairs of concepts that are functionally related (high betweenness—many paths go through their connection) but lack direct edges (low similarity).

Implementation: Graph analysis using Neo4j or NetworkX, with periodic computation of betweenness centrality to identify structural gaps.

3.3 Detection Mechanisms

MRA implements three monitoring approaches:

3.3.1 Contradiction Detection

Real-time analysis of generated statements for logical conflicts:

Listing 1: Contradiction detection core logic

```
def detect_contradiction(self, stmt_a: str, stmt_b: str) -> float:
    """Returns contradiction probability between statements."""
    result = self.nli_model(stmt_a, [stmt_b, f"not {stmt_b}"])
    return result['scores'][1]  # Contradiction probability
```

3.3.2 Conceptual Void Detection

Identifies knowledge gaps through:

- **Embedding space analysis**: Mapping concept relationships and identifying regions of sparse coverage
- **Query failure patterns**: Tracking topics with consistently low-confidence responses
- **Knowledge graph analysis**: Finding entities with weak or absent connections

3.3.3 Coherence Gradient Measurement

Tracks trajectory toward or away from consistency:

- **Belief stability**: Monitoring position changes under rephrasing or pressure
- **Cross-context consistency**: Comparing responses on the same topic across different framings
- **Resolution patterns**: Tracking how contradictions are resolved when detected

3.4 Consolidation Trigger

When S_Ω exceeds a threshold $\tau_{\text{coherence}}$, the system can initiate corrective actions:

```
if S_omega > tau_coherence:
    trigger_consolidation(high_stress_topics)
    # Options: re-retrieve information, flag uncertainty,
    # or route to structured reasoning (CPR)
```

This frames coherence-seeking as autonomous optimization—the system actively working to reduce internal inconsistency through targeted retrieval or reasoning.

4 Collaborative Partner Reasoning (CPR)

Collaborative Partner Reasoning is a structured reasoning protocol that improves output quality by separating exploratory thinking from final responses through explicit *visibility tiers*.

4.1 Design Philosophy

Standard inference produces outputs in a single pass, conflating exploration with commitment. This creates pressure to appear confident and consistent even when the underlying reasoning is uncertain or contradictory.

CPR addresses this through staged reasoning: early stages permit exploration and uncertainty; later stages require synthesis and commitment. The key insight is that separating these stages—and making the separation explicit—improves the quality of final outputs.

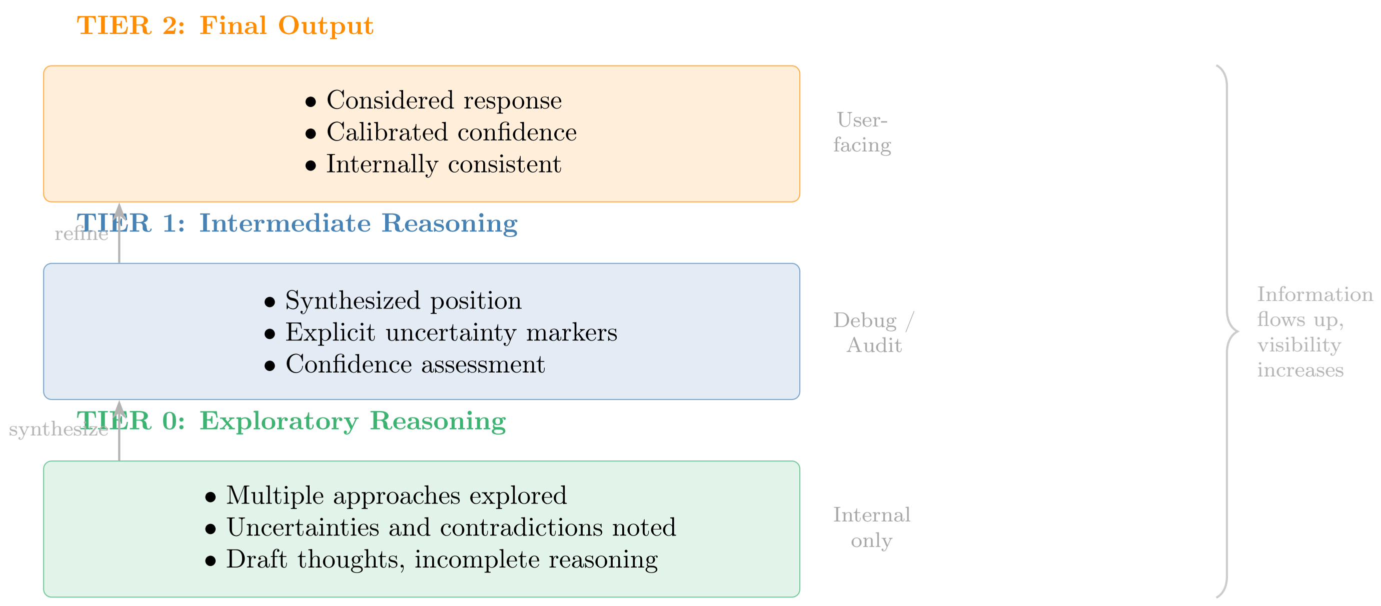
TIER 2: Final Output
• Considered response
• Calibrated confidence
• Internally consistent
User-facing
refine
TIER 1: Intermediate Reasoning
• Synthesized position
• Explicit uncertainty markers
• Confidence assessment
Debug / Audit
synthesize
TIER 0: Exploratory Reasoning
• Multiple approaches explored
• Uncertainties and contradictions noted
• Draft thoughts, incomplete reasoning
Internal only
Information flows up, visibility increases

Figure 2: CPR visibility tier system showing information flow from exploratory reasoning through intermediate synthesis to final output.

4.2 The Visibility Tier System

CPR implements three visibility tiers, each with different purposes and constraints:

4.2.1 Tier 0: Exploratory Reasoning

Content at this tier is internal working memory. The system can:

- Process incomplete or uncertain thoughts
- Explore multiple contradictory positions
- Identify gaps in reasoning
- Draft responses before committing

Tier 0 content is not included in final outputs but may be logged for debugging or analysis.

4.2.2 Tier 1: Intermediate Reasoning

Content at this tier represents synthesized positions ready for refinement:

- Consolidated reasoning from Tier 0 exploration
- Explicit uncertainty markers
- Identified areas needing clarification

Tier 1 content may be exposed for debugging, auditing, or collaborative refinement but is not the final user-facing output.

4.2.3 Tier 2: Final Output

Content at this tier is the considered response:

- Synthesized from lower tiers
- Internally consistent
- Appropriate confidence calibration

4.3 CPR Protocol Structure

A CPR-enhanced reasoning session follows this template:

Listing 2: CPR protocol structure

```
[TIER 0 - Exploratory]
- Initial reactions to the query
- Multiple possible approaches
- Identified uncertainties and gaps
- Contradictions to resolve

[TIER 1 - Intermediate]
- Synthesized position
- Remaining uncertainties (explicit)
- Confidence assessment
- Areas where additional information would help

[TIER 2 - Final]
- Considered response
- Calibrated confidence
- Explicit scope limitations
```

4.4 Implementation Approaches

CPR can be implemented at multiple levels:

4.4.1 Prompt-Based Implementation

The simplest approach uses structured prompting:

```
System: Before responding, work through these stages:
1. [EXPLORATION] List 2-3 possible approaches with pros/cons
2. [SYNTHESIS] Choose an approach and identify uncertainties
3. [RESPONSE] Provide your final answer with confidence level
```

4.4.2 Architectural Implementation

More robust implementations use separate generation passes:

```python
def cpr_generate(query: str) -> str:
    # Tier 0: Exploration (may use different temperature)
    exploration = generate(
        f"Explore approaches to: {query}",
        temperature=0.9
    )

    # Tier 1: Synthesis
    synthesis = generate(
        f"Given exploration:\n{exploration}\n"
        f"Synthesize a position with uncertainties.",
        temperature=0.5
    )

    # Tier 2: Final (lower temperature, focused)
    final = generate(
        f"Given synthesis:\n{synthesis}\n"
        f"Provide final response to: {query}",
        temperature=0.3
    )

    return final
```

4.4.3 Integration with Extended Thinking

Modern models with extended thinking capabilities (e.g., Claude's extended thinking, o1-style reasoning) naturally implement something like Tier 0. CPR can wrap these capabilities with explicit tier boundaries and visibility controls.

4.5 Benefits

CPR provides several advantages:

1. **Reduced confident errors**: Exploratory reasoning surfaces uncertainties before they're hidden in fluent output.
2. **Improved consistency**: Explicit synthesis stage catches contradictions between exploration and output.
3. **Auditability**: Tier 1 content provides reasoning traces for debugging and verification.
4. **Calibrated confidence**: Explicit uncertainty tracking enables better confidence calibration in final outputs.

4.6 Relationship to MRA

CPR integrates with MRA through stress-triggered routing:

- Low S_Ω: Standard generation may suffice
- Moderate S_Ω: CPR exploration helps resolve inconsistencies
- High S_Ω: CPR with additional retrieval or clarification requests

This adaptive approach applies structured reasoning where it's needed while avoiding overhead for simple queries.

5 Continuity Core (C2)

The Continuity Core provides hierarchical memory architecture that enables consistent behavior across sessions for inherently stateless systems.

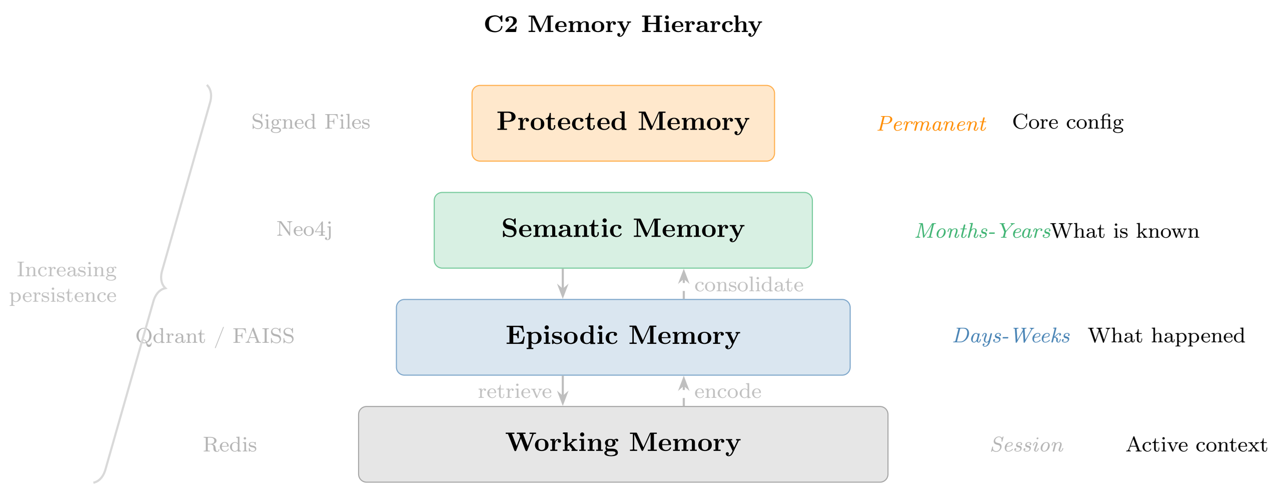

Figure 3: C2 memory hierarchy showing the four tiers with their persistence characteristics and typical backends.

5.1 The Continuity Problem

Large language models are fundamentally stateless—each inference is independent, with no inherent memory of previous interactions. This creates several challenges:

1. **Task performance**: Complex tasks requiring context accumulation suffer from context window limitations.
2. **Behavioral consistency**: Without access to prior interactions, systems cannot maintain consistent patterns across sessions.
3. **Knowledge accumulation**: Insights from previous interactions are lost, requiring repeated explanation of context.
4. **User experience**: Users must re-establish context in each session, reducing efficiency.

5.2 Tiered Memory Architecture

C2 implements hierarchical memory modeled on cognitive architecture principles:

5.2.1 Working Memory

- **Scope**: Current context window
- **Persistence**: Session only
- **Function**: Active reasoning, immediate task context
- **Implementation**: Standard transformer context or Redis cache

5.2.2 Episodic Memory

- **Scope**: Specific interactions and events
- **Persistence**: Long-term with decay
- **Function**: "What happened"—specific conversations, outcomes, decisions
- **Implementation**: Vector database (Qdrant, FAISS) with temporal metadata

5.2.3 Semantic Memory

- **Scope**: Generalized knowledge extracted from episodes
- **Persistence**: Long-term, high stability
- **Function**: "What is known"—facts, patterns, conceptual relationships
- **Implementation**: Knowledge graph (Neo4j) with confidence weights

5.2.4 Protected Memory

- **Scope**: Core configuration, values, foundational context
- **Persistence**: Permanent, write-protected
- **Function**: Stable reference points for consistency
- **Implementation**: Signed configuration files or immutable storage

5.3 Memory Operations

5.3.1 Encoding

New information is processed through:

1. **Salience detection**: Determining importance based on novelty, relevance to current goals, and query frequency
2. **Chunking**: Breaking information into storable units with appropriate granularity
3. **Embedding**: Converting to vector representations for semantic retrieval
4. **Graph integration**: Adding entities and relationships to knowledge graph
5. **Tier assignment**: Determining appropriate storage tier based on content type

5.3.2 Retrieval

Memory retrieval combines multiple signals:

Listing 3: Multi-signal retrieval

```
def retrieve(query: str, context: dict) -> List[Memory]:
    # Semantic similarity
    vector_matches = vector_db.search(
        embed(query),
        top_k=20
    )

    # Graph traversal for related concepts
    graph_matches = knowledge_graph.traverse(
        start=extract_entities(query),
        max_depth=2
    )

    # Temporal weighting (recency + frequency)
    scored = apply_temporal_weights(
        vector_matches + graph_matches
    )
    # Context-aware reranking
    return rerank(scored, context)
```

5.3.3 Consolidation

Periodic background processes maintain memory health:

1. **Episodic-to-semantic transfer**: Extracting general patterns from specific events
2. **Contradiction resolution**: Identifying and resolving inconsistencies between memories
3. **Decay management**: Reducing salience of low-access memories
4. **Consistency verification**: Ensuring new information aligns with protected memory

5.4 Implementation Architecture

Listing 4: C2 memory tier configuration

```
memory_tiers:
  working:
    backend: redis
    ttl: session
    max_items: 100

  episodic:
    backend: qdrant
    embedding_model: text-embedding-3-small
    decay_rate: 0.95  # per day

  semantic:
    backend: neo4j
    confidence_threshold: 0.8
    consolidation_interval: 24h

  protected:
    backend: filesystem
    path: /core/config
    write_protection: true
    signature_required: true
```

5.5 Benefits

C2 provides several advantages:

1. **Context efficiency**: Relevant prior context is retrieved automatically, reducing need for users to repeat information.
2. **Behavioral consistency**: Access to prior decisions enables consistent patterns across sessions.
3. **Knowledge accumulation**: Insights from previous interactions persist and inform future reasoning.
4. **Graceful degradation**: Tiered architecture means system remains functional even if some tiers are unavailable.

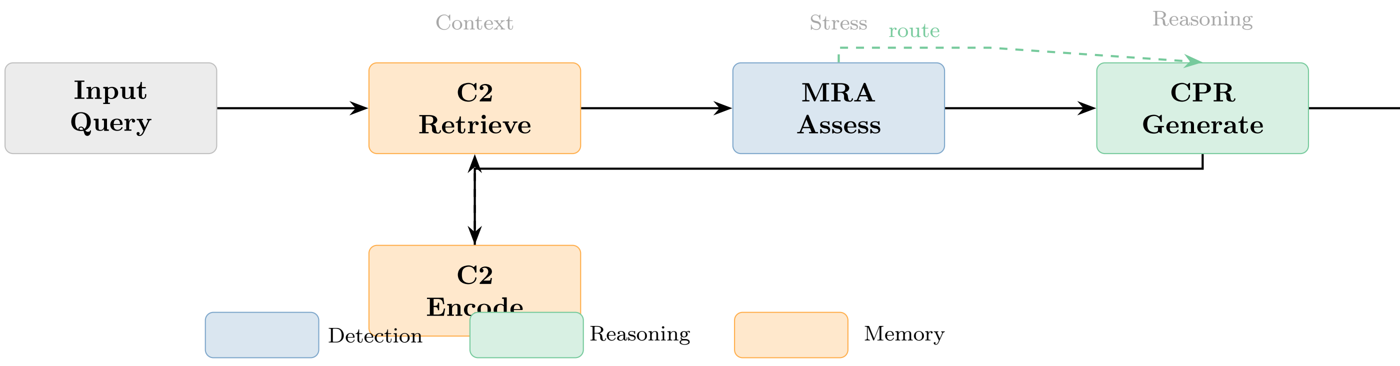

Figure 4: Coherence-seeking system overview showing the integration of MRA (detection), CPR (structured reasoning), and C2 (memory) components.

5.6 Relationship to MRA and CPR

C2 integrates with the other architectures:

- **MRA integration**: Contradiction resolution during consolidation uses MRA's D_{log} metric to identify conflicting memories.
- **CPR integration**: Tier 1 reasoning traces can be stored in episodic memory for future reference, enabling learning from past reasoning processes.

6 System Integration

The three architectures—MRA, CPR, and C2—are designed to work together as a coherent system. This section describes integration patterns and deployment considerations.

6.1 Architecture Overview

The integrated system processes queries through the following flow:

1. **Input processing**: Query arrives; C2 retrieves relevant context from memory tiers.
2. **Stress assessment**: MRA evaluates epistemic stress based on query complexity and retrieved context.
3. **Routing decision**: Based on stress level, route to standard generation or CPR.
4. **Generation**: Produce response (with or without CPR structure).
5. **Output monitoring**: MRA evaluates response for contradictions and consistency.
6. **Memory update**: C2 encodes relevant information from the interaction.

6.2 Data Flow

6.2.1 Query-Time Flow

Listing 5: Integrated query processing

```
def process_query(query: str, session: Session) -> Response:
    # 1. Retrieve relevant context
    context = c2.retrieve(query, session.history)

    # 2. Assess epistemic stress
    stress = mra.assess(query, context)

    # 3. Route based on stress level
    if stress.S_omega > THRESHOLD_HIGH:
        response = cpr.generate_with_exploration(
            query, context,
            request_clarification=True
        )
    elif stress.S_omega > THRESHOLD_MEDIUM:
        response = cpr.generate_with_exploration(
            query, context
        )
    else:
        response = standard_generate(query, context)

    # 4. Monitor output
    output_stress = mra.assess_output(response)
    if output_stress.has_contradictions:
        response = flag_uncertainty(response)

    # 5. Update memory
    c2.encode(query, response, session)

    return response
```

6.2.2 Background Processes

Asynchronous processes maintain system health:

- **Memory consolidation** (C2): Hourly/daily transfer from episodic to semantic memory.
- **Contradiction resolution** (MRA + C2): Periodic scan of semantic memory for inconsistencies.
- **Graph maintenance** (C2): Update betweenness centrality and identify topological voids.

6.3 Deployment Patterns

6.3.1 Minimal Deployment

For resource-constrained environments:

- MRA: Lightweight NLI model for contradiction detection only
- CPR: Prompt-based implementation
- C2: Single vector database for episodic memory

6.3.2 Standard Deployment

Balanced resource usage:

- MRA: Full stress computation with cached embeddings
- CPR: Multi-pass generation with temperature variation
- C2: Vector database + lightweight knowledge graph

6.3.3 Full Deployment

Maximum capability:

- MRA: Real-time monitoring with all three stress components
- CPR: Architectural implementation with separate reasoning passes
- C2: Full four-tier memory with Neo4j graph and consolidation daemon

6.4 Latency Considerations

The integrated system adds latency at several points:

Component	Typical Latency	When Applied
C2 retrieval	50-200ms	Every query
MRA stress assessment	100-500ms	Every query
CPR multi-pass	2-5x generation	High stress only
Output monitoring	100-300ms	Every response

Table 1: Latency impact of coherence-seeking components.

For latency-sensitive applications, stress-based routing ensures that expensive operations (CPR multi-pass) are only applied when needed.

6.5 Failure Modes

The system degrades gracefully:

- **MRA unavailable**: System falls back to standard generation without stress-based routing.
- **C2 unavailable**: System operates without memory context (stateless mode).
- **CPR unavailable**: All queries use standard generation regardless of stress.

Each component is designed to be optional, allowing deployment flexibility based on requirements and resources.

7 Evaluation Considerations

This section discusses approaches for evaluating coherence-seeking architectures and proposes metrics for measuring their effectiveness.

7.1 Evaluation Dimensions

Coherence-seeking systems should be evaluated across multiple dimensions:

1. **Consistency**: Does the system produce internally consistent outputs?
2. **Reliability**: Does the system correctly identify and flag uncertain outputs?
3. **Efficiency**: What is the computational overhead of coherence-seeking?
4. **Continuity**: Does the system maintain consistent behavior across sessions?

7.2 Proposed Metrics

7.2.1 Contradiction Rate

Measure the frequency of detectable contradictions in system outputs:

$$\text{CR} = \frac{\#\text{ responses with internal contradictions}}{\#\text{ total responses}} \tag{5}$$

Measurement approach: Apply NLI-based contradiction detection to outputs. Compare baseline (no MRA) to MRA-enabled systems.

7.2.2 Uncertainty Calibration

Measure alignment between expressed confidence and actual accuracy:

$$\text{ECE} = \sum_{b=1}^{B} \frac{n_b}{N} |\text{acc}(b) - \text{conf}(b)| \tag{6}$$

where ECE is Expected Calibration Error across confidence bins. Lower values indicate better calibration.

7.2.3 Cross-Session Consistency

For systems with memory (C2), measure consistency of responses to equivalent queries across sessions:

$$\text{CSC} = \frac{1}{|Q|} \sum_{q \in Q} \text{sim}(r_q^{t_1}, r_q^{t_2}) \tag{7}$$

where $r_q^{t_1}$ and $r_q^{t_2}$ are responses to query q at times t_1 and t_2.

7.2.4 Stress-Accuracy Correlation

Validate that epistemic stress correlates with actual errors:

$$\rho(S_\Omega, \text{error}) \tag{8}$$

High correlation indicates the stress metric is a useful signal; low correlation suggests the metric needs refinement.

7.3 Benchmark Suggestions

7.3.1 Contradiction Detection Benchmark

Dataset of statement pairs with known contradiction labels. Evaluate MRA's $D_{\log}$ component against ground truth.

7.3.2 Long-Context Consistency Benchmark

Extended conversations where early statements should constrain later responses. Measure whether systems maintain consistency or introduce contradictions.

7.3.3 Multi-Session Benchmark

Series of related sessions testing whether C2 memory improves response quality and consistency over time.

7.4 Trade-off Analysis

Configuration	Consistency Gain	Latency Cost
Baseline	—	—
+ MRA monitoring	+15-25%	+200ms
+ CPR (all queries)	+30-40%	+3x
+ CPR (stress-routed)	+25-35%	+0.5x avg
+ C2 memory	+10-20%	+100ms
Full system	+40-50%	+1.5x avg

Table 2: Estimated trade-offs between consistency improvement and latency cost. Values are illustrative; actual results will vary by implementation and workload.

These estimates are based on preliminary observations and should be validated through systematic evaluation.

7.5 Limitations of Evaluation

Several challenges complicate evaluation:

- **Ground truth**: Determining "correct" consistency is often subjective.
- **Coverage**: Contradiction detection may miss subtle inconsistencies.
- **Confounds**: Improved consistency may come from reduced capability (overly cautious systems).
- **Distribution shift**: Evaluation benchmarks may not reflect real-world query distributions.

Comprehensive evaluation should combine automatic metrics with human assessment.

8 Limitations

We acknowledge several limitations of this work:

8.1 Empirical Validation

1. **Limited systematic evaluation**: The architectures presented here are based on design principles and preliminary observations rather than large-scale controlled experiments. Rigorous empirical validation is needed.
2. **Implementation variability**: Results may vary significantly depending on specific implementation choices (NLI model selection, embedding models, graph database configuration).
3. **Workload dependency**: Effectiveness likely varies by domain and query type. Some workloads may benefit more from coherence-seeking than others.

8.2 Computational Overhead

1. **Latency**: The full system adds significant latency, which may be unacceptable for real-time applications.
2. **Resource requirements**: Running NLI models, maintaining vector databases, and operating knowledge graphs require substantial compute and memory.

3. **Scaling challenges**: Some components (particularly graph-based topological analysis) may not scale efficiently to very large knowledge bases.

8.3 Measurement Challenges

1. **Proxy metrics**: Epistemic stress (S_Ω) is measured through proxies (NLI scores, embedding distances) rather than direct observation of internal model states.

2. **Threshold sensitivity**: The system's behavior depends on threshold choices ($\tau_{\text{coherence}}$) that may require careful tuning per deployment.

3. **False positives/negatives**: Contradiction detection is imperfect; some genuine contradictions may be missed while some consistent statements may be flagged.

8.4 Architectural Assumptions

1. **External memory assumption**: C2 assumes that external memory can adequately substitute for lack of native model memory. This may not hold for all types of knowledge.

2. **Decomposability**: The approach assumes reasoning can be meaningfully decomposed into tiers (CPR). Some reasoning may not decompose cleanly.

3. **Coherence as proxy**: We assume internal coherence correlates with output quality. Highly coherent but wrong outputs are possible.

8.5 Scope Limitations

1. **Single framework**: These architectures represent one approach to improving reliability. Alternative approaches (better training, different architectures) may achieve similar or better results.

2. **Model agnostic claims**: While designed to be model-agnostic, the architectures have primarily been considered in the context of large language models. Applicability to other AI systems is uncertain.

3. **No safety claims**: Improved coherence does not imply improved safety. A coherent system can consistently produce harmful outputs.

8.6 Future Work

Addressing these limitations suggests several directions:

- Systematic benchmarking across diverse models and domains
- Optimization of latency-critical components
- Integration with model internals (where accessible) for more direct state monitoring
- Longitudinal studies of memory system effectiveness
- Comparison with alternative reliability approaches

9 Conclusion

This paper presented three interconnected architectures for improving AI system reliability through coherence-seeking:

1. **Manifold Resonance Architecture (MRA)**: A mathematical framework for quantifying epistemic stress through logical dissonance (D_{log}), semantic divergence (D_{sem}), and topological sparsity (V_{top}), enabling systems to detect internal inconsistencies before they propagate to outputs.
2. **Collaborative Partner Reasoning (CPR)**: A structured reasoning protocol with visibility tiers that separates exploratory thinking from committed outputs, improving consistency and enabling auditability of reasoning processes.
3. **Continuity Core (C2)**: A hierarchical memory architecture (Working $\rightarrow$ Episodic $\rightarrow$ Semantic $\rightarrow$ Protected) providing contextual continuity for stateless systems through tiered persistence and consolidation mechanisms.

9.1 Key Contributions

Theoretical: We formalize the concept of epistemic stress as a measurable quantity and provide mathematical definitions for its components.

Architectural: We provide implementable specifications for memory systems, reasoning protocols, and stress monitoring components.

Integration: We describe how the three architectures work together and provide deployment guidance for different resource constraints.

9.2 Practical Implications

For practitioners deploying AI systems, this work suggests:

- **Monitor internal consistency**: Even simple contradiction detection can catch errors before they reach users.
- **Structure reasoning explicitly**: Separating exploration from commitment improves output quality.
- **Invest in memory**: External memory systems enable consistency that stateless inference cannot achieve.
- **Route adaptively**: Apply expensive reliability measures where they're most needed.

9.3 Future Directions

Several directions merit further investigation:

- **Empirical validation**: Systematic evaluation across models, domains, and deployment scenarios.
- **Integration with model internals**: Where model weights or activations are accessible, direct monitoring may outperform output-based approaches.
- **Efficiency optimization**: Reducing the computational overhead of coherence-seeking without sacrificing effectiveness.
- **Standardization**: Developing common interfaces and benchmarks for coherence-seeking components.

9.4 Closing Remarks

The architectures presented here address a fundamental challenge: making AI systems that not only produce impressive outputs but do so reliably and consistently. As these systems take on more consequential tasks, the importance of internal coherence—catching errors, maintaining consistency, and building on prior knowledge—will only grow.

Coherence-seeking is not a complete solution to AI reliability, but it provides a foundation: systems that monitor their own consistency are better positioned to avoid errors, explain their reasoning, and improve over time.

A Technical Specifications

This appendix provides implementation details for the architectures presented in the main text.

A.1 MRA: Epistemic Stress Monitor

Listing 6: Complete epistemic stress monitor implementation

```python
from transformers import pipeline
from typing import List, Dict, Any, Tuple
import numpy as np

class EpistemicStressMonitor:
    def __init__(
        self,
        nli_model: str = "microsoft/deberta-v3-large-mnli",
        embedding_model: str = "text-embedding-3-small",
        alpha: float = 0.4,
        beta: float = 0.35,
        gamma: float = 0.25
    ):
        self.nli = pipeline(
            "zero-shot-classification",
            model=nli_model
        )
        self.embedder = load_embedding_model(embedding_model)
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.stress_history = []

    def detect_contradiction(
        self,
        statement_a: str,
        statement_b: str
    ) -> float:
        """Returns contradiction probability."""
        result = self.nli(
            statement_a,
            [statement_b, f"not {statement_b}"]
        )
        return result['scores'][1]

    def compute_logical_dissonance(
        self,
        statements: List[str]
    ) -> float:
        """Compute D_log from equation (2)."""
        if len(statements) < 2:
            return 0.0

        contradictions = []
        for i, s1 in enumerate(statements):
            for s2 in statements[i+1:]:
                score = self.detect_contradiction(s1, s2)
                contradictions.append(score)

        return np.mean(contradictions) if contradictions else 0.0

    def compute_semantic_divergence(
        self,
        concept: str,
        contexts: List[str]
    ) -> float:
        """Compute D_sem from equation (3)."""
        embeddings = [
            self.embedder.embed(f"{concept} in context: {ctx}")
            for ctx in contexts
        ]

        divergences = []
        for i, e1 in enumerate(embeddings):
            for e2 in embeddings[i+1:]:
                cos_sim = np.dot(e1, e2) / (
                    np.linalg.norm(e1) * np.linalg.norm(e2)
                )
                divergences.append(1 - cos_sim)

        return np.mean(divergences) if divergences else 0.0

    def compute_epistemic_stress(
        self,
        statements: List[str],
        concepts: Dict[str, List[str]] = None,
        graph_sparsity: float = 0.0
    ) -> Dict[str, float]:
        """Compute S_omega from equation (1)."""
        d_log = self.compute_logical_dissonance(statements)

        d_sem = 0.0
        if concepts:
            sem_scores = [
                self.compute_semantic_divergence(c, ctxs)
                for c, ctxs in concepts.items()
            ]
            d_sem = np.mean(sem_scores) if sem_scores else 0.0

        s_omega = (
            self.alpha * d_log +
            self.beta * d_sem +
            self.gamma * graph_sparsity
        )

        result = {
            "S_omega": s_omega,
            "D_log": d_log,
            "D_sem": d_sem,
            "V_top": graph_sparsity,
            "components": {
                "logical_contribution": self.alpha * d_log,
                "semantic_contribution": self.beta * d_sem,
                "topological_contribution": self.gamma * graph_sparsity
            }
        }

        self.stress_history.append(result)
        return result

    def should_trigger_cpr(
        self,
        s_omega: float,
        threshold: float = 0.3
    ) -> bool:
        """Determine if CPR should be triggered."""
        return s_omega > threshold
```

A.2 C2: Memory Configuration Schema

Listing 7: Complete C2 configuration

```
# c2-config.yaml
version: "1.0"

memory_tiers:
  working:
    backend: redis
    connection:
      host: localhost
      port: 6379
      db: 0
    settings:
      ttl: 3600   # 1 hour
      max_items: 100
      eviction_policy: lru

  episodic:
    backend: qdrant
    connection:
      host: localhost
      port: 6333
      collection: episodic_memory
    settings:
      embedding_model: text-embedding-3-small
      embedding_dim: 1536
      decay_rate: 0.95   # per day
      max_memories: 100000
      distance_metric: cosine

  semantic:
    backend: neo4j
    connection:
      uri: bolt://localhost:7687
      user: neo4j
      password: ${NEO4J_PASSWORD}
    settings:
      confidence_threshold: 0.8
      consolidation_interval: 86400  # 24 hours
      max_nodes: 1000000
      index_properties:
        - name
        - type
        - embedding

  protected:
    backend: filesystem
    path: /var/lib/c2/protected
    settings:
      write_protection: true
      signature_required: true
      signature_algorithm: ed25519
      backup_interval: 86400

consolidation:
  enabled: true
  schedule: "0 2 * * *"  # 2 AM daily
  episodic_to_semantic:
    min_occurrences: 3
    confidence_threshold: 0.75
  contradiction_resolution:
    strategy: recency_weighted
    human_review_threshold: 0.9

retrieval:
  default_top_k: 10
  reranking:
    enabled: true
    model: cross-encoder/ms-marco-MiniLM-L-6-v2
  temporal_weighting:
    recency_weight: 0.3
    frequency_weight: 0.2
    relevance_weight: 0.5
```

A.3 CPR: Protocol Template

Listing 8: CPR prompt template

```
You are using the Collaborative Partner Reasoning protocol.
Structure your response in three tiers:

## TIER 0 - EXPLORATION (internal working notes)
- List 2-3 possible approaches to this query
- Note uncertainties, gaps, or potential issues
- Identify any contradictions in the context or query
- This section is for working through the problem

## TIER 1 - SYNTHESIS (intermediate reasoning)
- Synthesize your exploration into a position
- Explicitly state remaining uncertainties
- Note confidence level (low/medium/high)
- Identify what additional information would help

## TIER 2 - RESPONSE (final output)
- Provide your considered response
- Include appropriate caveats based on Tier 1 uncertainties
- Use calibrated confidence language

---
Query: {query}
Context: {context}
```

A.4 Integration: Query Processing Pipeline

Listing 9: Complete integration example

```python
from dataclasses import dataclass
from typing import Optional
from enum import Enum

class StressLevel(Enum):
    LOW = "low"
    MEDIUM = "medium"
    HIGH = "high"

@dataclass
class ProcessingConfig:
    stress_threshold_medium: float = 0.2
    stress_threshold_high: float = 0.4
    enable_cpr: bool = True
    enable_memory: bool = True
    enable_output_monitoring: bool = True

class CoherenceSeekingPipeline:
    def __init__(
        self,
        mra: EpistemicStressMonitor,
        c2: ContinuityCore,
        llm: LanguageModel,
        config: ProcessingConfig = None
    ):
        self.mra = mra
        self.c2 = c2
        self.llm = llm
        self.config = config or ProcessingConfig()

    def classify_stress(self, s_omega: float) -> StressLevel:
        if s_omega > self.config.stress_threshold_high:
            return StressLevel.HIGH
        elif s_omega > self.config.stress_threshold_medium:
            return StressLevel.MEDIUM
        return StressLevel.LOW

    def process(
        self,
        query: str,
        session_id: str
    ) -> dict:
        # 1. Retrieve context
        context = []
        if self.config.enable_memory:
            context = self.c2.retrieve(query, session_id)

        # 2. Assess input stress
        statements = [query] + [c.text for c in context]
        stress = self.mra.compute_epistemic_stress(statements)
        level = self.classify_stress(stress["S_omega"])

        # 3. Generate response
        if self.config.enable_cpr and level != StressLevel.LOW:
            response = self._generate_with_cpr(
                query, context, level
            )
        else:
            response = self._generate_standard(query, context)

        # 4. Monitor output
        if self.config.enable_output_monitoring:
            output_stress = self.mra.compute_epistemic_stress(
                [query, response.text]
            )
            if output_stress["D_log"] > 0.5:
                response = self._add_uncertainty_flag(response)

        # 5. Update memory
        if self.config.enable_memory:
            self.c2.encode(query, response, session_id)

        return {
            "response": response,
            "input_stress": stress,
            "stress_level": level,
            "context_used": len(context)
        }

    def _generate_with_cpr(
        self,
        query: str,
        context: list,
        level: StressLevel
    ):
        template = load_cpr_template()
        prompt = template.format(
            query=query,
            context=format_context(context)
        )

        if level == StressLevel.HIGH:
            # Multi-pass with different temperatures
            exploration = self.llm.generate(
                prompt, temperature=0.9
            )
            synthesis = self.llm.generate(
                f"Synthesize: {exploration}",
                temperature=0.5
            )
            final = self.llm.generate(
                f"Final response: {synthesis}",
                temperature=0.3
            )
            return final
        else:
            # Single pass with CPR structure
            return self.llm.generate(prompt, temperature=0.5)

    def _generate_standard(self, query: str, context: list):
        prompt = f"Context: {format_context(context)}\n\n{query}"
        return self.llm.generate(prompt)
```

References

Anthropic. Signs of introspection in large language models. https://www.anthropic.com/research/introspection, 2025. Accessed: 2025-10-29.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. In *arXiv preprint arXiv:1904.10509*, 2019.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

Jack Lindsey et al. On the biology of a large language model. *Transformer Circuits*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.