Asynchronous Reasoning Engine: Technical Overview

Purpose: A reasoning engine for decision support that tracks multiple contradictory hypotheses over noisy/asynchronous business data, reviving dormant theories with new evidence.

Motivation and Problem Statement

The Business Intelligence Challenge

Business decision-making consistently involves contradictory data signals: sales metrics suggest growth while customer satisfaction scores decline, security alerts indicate threats while operational metrics appear normal, or performance dashboards show optimal response times while user complaints report slowness.

Many automated reasoning systems struggle with this contradiction. Some force immediate resolution between contradictory signals, while others oscillate between different interpretations as new data arrives. Approaches based on formal logical frameworks often require consistency maintenance, which can lead to:

• **Premature resolution**: Forces choosing between contradictory signals before sufficient evidence accumulates • **Information loss**: Discards potentially valid alternative explanations to maintain consistency
• **Analytical brittleness**: Cannot handle the nuanced, multi-faceted nature of business problems
• **Poor decision support**: Provides artificially certain conclusions that may reverse with the next data point

Human Cognitive Advantage

Experienced business analysts naturally maintain multiple contradictory hypotheses with different confidence levels until additional evidence clarifies the situation. This cognitive flexibility allows productive reasoning through uncertainty rather than forcing premature certainty.

Our system replicates this human capability: embracing contradictory hypotheses as a feature rather than a logical error, allowing gradual evidence accumulation without forced immediate revision, and enabling dormant theories to resurface when new evidence makes them relevant.

Technical Innovation: Single LLM with Structured Context

Instead of building complex multi-agent systems or forced belief revision frameworks, we use a single language model managing a rewritable context window filled with structured assertion

tuples. Each piece of evidence and hypothesis becomes a timestamped tuple containing content, confidence level, status, source, and metadata.

The core innovation combines: • **Structured assertion tuples**: (content, timestamp, conf:X.XX, status, source) • **Deep thought processing**: Protected reasoning sessions with stable context snapshots • **Asynchronous evidence handling**: Fast ingestion with queued processing during analysis • **Hypothesis evolution**: Confidence-based status transitions rather than deletion/replacement • **Investigation management**: Complete lifecycle tracking with persistent results • **Multiple failsafe layers**: Protection against infinite reasoning loops

Unified Reasoning Architecture

Replicating Human Cognitive Integration

One of the most remarkable aspects of human intelligence is our ability to hold contradictory beliefs simultaneously without forcing immediate resolution. A fundamental challenge in reasoning system design is mimicking this natural human ability to maintain multiple competing explanations until evidence clarifies which is correct—or reveals that both might be true in different contexts.

The Human Cognitive Advantage

Humans can comfortably hold contradictory beliefs like: • "My company is doing well" AND "I'm worried about layoffs" • "This investment strategy works" AND "The market feels risky" • "Our customers are happy" AND "I'm seeing concerning support trends"

We live in the contradiction until we understand it better. Human cognition achieves this through: • **Parallel processing**across brain regions that consider multiple theories simultaneously • **Integrated memory and attention** systems that maintain all relevant context • **Unconscious background processing** that continuously evaluates evidence relationships • **Natural context switching** and prioritization between competing explanations

Traditional AI approaches often employ agentic systems—multiple specialized reasoning agents—to overcome inherent LLM limitations: • **Sequential processing**: LLMs cannot truly parallel process like biological brains • **Context window constraints**: Even large contexts suffer from attention degradation • **Specialization benefits**: Different reasoning patterns may benefit from dedicated agents • **Explicit state management**: What brains handle automatically requires engineered coordination

Single LLM Breakthrough

Our approach replicates human cognitive advantages through unified context presentation rather than distributed agent coordination. By providing the LLM with complete structured context containing all hypotheses, evidence, and relationships simultaneously, we achieve:

Integrated Reasoning: All evidence-hypothesis relationships are visible in each reasoning cycle, enabling comprehensive impact assessment rather than fragmented analysis.

Natural Hypothesis Interaction: Competing theories can be directly compared and contrasted within unified context, mimicking how human minds weigh alternatives.

Seamless Revival Logic: Dormant hypotheses remain contextually available for reactivation, rather than requiring complex inter-agent communication to resurrect forgotten theories.

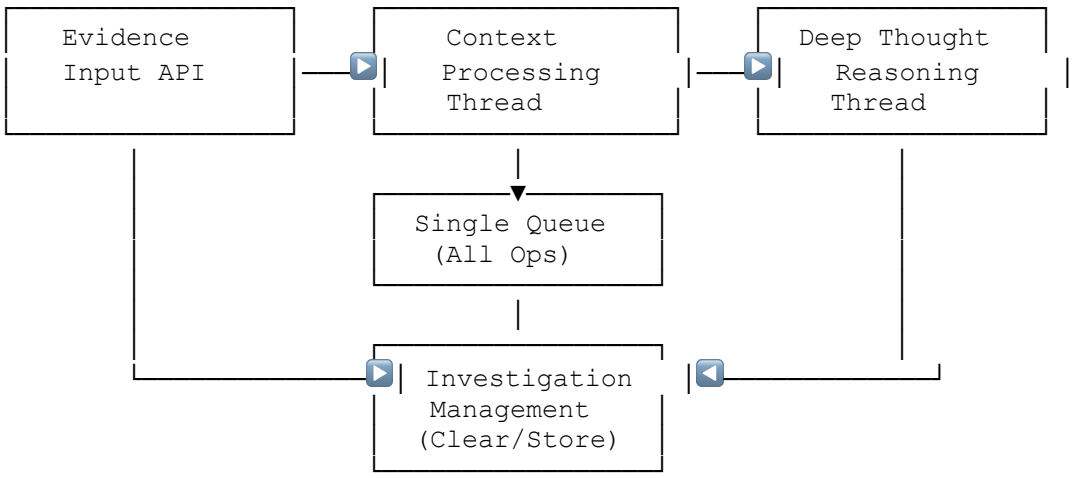
Unified Confidence Assessment: Evidence strength is evaluated against the complete hypothesis landscape, producing more accurate confidence adjustments.

This unified approach proves superior for contradiction management because it preserves the holistic perspective that makes human reasoning effective while leveraging LLM capabilities for structured analysis.

Architecture Summary

The Asynchronous Reasoning Engine implements a dual-thread architecture that separates fast evidence processing from deep analytical reasoning, enabling continuous evidence ingestion while maintaining protected reasoning sessions.

Core Components



Thread Architecture

Context Processing Thread

- **Purpose**: Fast, non-blocking processing of incoming evidence and hypotheses

- **Responsibilities**: • Validate and categorize incoming assertions • Process single operation queue with boundary logic • Maintain sub-10ms response times • Handle investigation management operations • Signal context changes to reasoning thread

Deep Thought Reasoning Thread

- **Purpose**: Protected analytical reasoning on stable context snapshots
- **Responsibilities**: • Monitor context version changes • Execute deep thought reasoning sessions • Manage hypothesis confidence evolution • Detect reasoning completion and oscillation patterns • Apply multiple failsafe mechanisms

Single Queue Architecture

Unified Operation Processing:

- All operations (evidence, hypotheses, clears, queries) use single queue
- Clear operations act as investigation boundaries
- Operations before clear are processed in batches
- Clear operations capture results and reset context
- Eliminates race conditions between operation types

Deep Thought Processing

Protected Reasoning Sessions

Deep thought mode represents the core innovation: uninterruptible reasoning sessions where the LLM analyzes a stable snapshot of assertion tuples without interference from new evidence. This mirrors how human analysts need uninterrupted thinking time to reason through complex problems.

Deep Thought Triggers

```
if self.context_version > self.last_reasoned_version:
    self.deep_thought_mode = True
    self.deep_thought_start = datetime.now()
    self.reasoning_loop_count = 0  # Reset failsafe counter
```

Evidence Isolation During Reasoning

```
def add_evidence(self, content: str, source: str, confidence: float) -> str:
    operation = ContextOperation(
        operation_type=OperationType.ADD_EVIDENCE,
        data={'content': content, 'source': source, 'confidence': confidence}
    )
    self.queued_assertions.put(operation)  # Always queued
    return operation.operation_id
```

Reasoning Session Structure

```
def _perform_reasoning_cycle(self):
    # Increment simple loop counter first
    self.reasoning_loop_count += 1

    # Simple failsafe check
    if self.reasoning_loop_count >= self.max_reasoning_loops:
        return {
            'should_stop_reasoning': True,
            'stop_reason': f'FAILSAFE: Maximum reasoning loops reached',
            'failsafe_triggered': True
        }

    # Work with stable context snapshot
    context_hash_before = self._compute_context_hash()

    # LLM analyzes assertion tuples
    self._reason_about_new_items(context_summary, new_items)

    # Detect if meaningful changes occurred
    context_hash_after = self._compute_context_hash()
    context_changed = context_hash_before != context_hash_after

    # Continue until stable or limits reached
    return {'should_stop_reasoning': not context_changed or other_limits}
```

Multiple Failsafe Mechanisms

1. **Simple Loop Counter**: `reasoning_loop_count` vs `max_reasoning_loops`
2. **Hash-Based Termination**: Identical context hashes indicate stability
3. **Sterile Cycle Detection**: No meaningful changes for N cycles
4. **Oscillation Detection**: Recognizes cyclic patterns
5. **Deep Thought Timeout**: Time-based reasoning limits
6. **Human Override**: Manual intervention capability

Hash-Based Termination

Deep thought sessions continue until the system reaches analytical stability, detected through context hashing:

```
def _compute_context_hash(self) -> str:
    context_data = []
    for item in sorted(self.context_items, key=lambda x: x.timestamp):
        item_repr = (
            item.content,
            item.item_type.value,
            item.status.value,
            round(item.confidence, 3),  # Round to avoid floating point noise
            round(item.importance, 3),
            item.source,
            tuple(sorted(item.tags))
```

```
        )
        context_data.append(item_repr)

    context_str = str(context_data)
    return hashlib.sha256(context_str.encode()).hexdigest()[:16]
```

When consecutive reasoning cycles produce identical context hashes, deep thought terminates with stable conclusions reached.

Data Structures

ContextItem

```
@dataclass
class ContextItem:
    content: str                # Immutable assertion text
    timestamp: datetime         # Creation time
    item_type: ItemType        # EVIDENCE | HYPOTHESIS | REVIVAL
    status: Status             # ACTIVE | WEAKENED | DORMANT
    confidence: float          # 0.1 to 0.95 confidence level
    importance: float          # System-assessed importance
    source: str               # Evidence source identifier
    tags: List[str]           # Categorization tags
    reasoning_version: int    # Last reasoning cycle that processed this item
```

Investigation Management

```
@dataclass
class ContextOperation:
    operation_type: OperationType
    data: Dict[str, Any]
    timestamp: datetime = field(default_factory=datetime.now)
    operation_id: str = field(default_factory=lambda:
str(datetime.now().timestamp()))
    result_queue: Optional[queue.Queue] = None
```

Assertion Tuple Format

The LLM receives context as structured tuples:

```
(Enterprise Q4 sales exceeded targets by 30%, 14:23, conf:0.85, active,
sales_system)
(SMB market requires different approach, 14:18, conf:0.70, active,
llm_generation)
(Infrastructure scaling is adequate, 14:12, conf:0.40, weakened,
llm_generation)
(Seasonal patterns explain recent trends, 14:10, conf:0.15, dormant,
llm_generation)
```

Context Versioning

• **context_version**: Incremented on each context change • **last_reasoned_version**: Last version processed by reasoning • **context_change_event**: Thread synchronization primitive
• **reasoning_loop_count**: Simple counter for current reasoning session • **max_reasoning_loops**: Configurable failsafe limit

Hypothesis Evolution

Confidence-Only Evolution

Hypotheses maintain immutable content but evolve through: • **Confidence adjustments**: ±0.1 to ±0.3 based on evidence strength • **Status transitions**: Confidence thresholds determine active/weakened/dormant status
• **Two-phase revival**: Dormant hypotheses first return to active status, then gain confidence in subsequent cycles

Status Transition Logic

```
if any(term in analysis for term in ['supports', 'strengthens']):
    if item.status == Status.DORMANT:
        # Revival: Reset to viable confidence level
        item.status = Status.ACTIVE
        item.confidence = 0.6  # Sufficient to avoid immediate re-dormancy
        self._log_revival(item, "LLM detected supporting evidence")
    else:
        # Normal confidence increase
        increase = min(0.2, max(0.05, 0.15))
        item.confidence = min(0.95, item.confidence + increase)

elif any(term in analysis for term in ['contradicts', 'weakens']):
    decrease = min(0.25, max(0.1, 0.2))
    item.confidence = max(0.1, item.confidence - decrease)

    # Update status based on new confidence
    if item.confidence < 0.3:
        item.status = Status.DORMANT
    elif item.confidence < 0.5:
        item.status = Status.WEAKENED
```

Oscillation Detection

Pattern Recognition

```
def _detect_oscillation(self, current_hash: str) -> Dict[str, Any]:
    recent_hashes = self.context_hash_history[-
self.oscillation_detection_window:]

    # 2-state oscillation detection
    if len(set(recent_hashes)) == 2:
        hash_a, hash_b = list(set(recent_hashes))
```

```
        expected_pattern = [hash_a, hash_b] *
(self.oscillation_detection_window // 2)
        if recent_hashes == expected_pattern:
            return {
                'oscillating': True,
                'pattern': '2-state',
                'states': [hash_a[:8], hash_b[:8]]
            }
```

Oscillation Response

• Immediate reasoning termination • Preservation of oscillating hypothesis states
• Explicit reporting of analytical ambiguity

LLM Integration

Single LLM Architecture

The system uses a unified LLM approach rather than multiple specialized agents, allowing the language model to manage the complete investigation process while maintaining natural reasoning patterns.

Azure OpenAI Compatible DSPy Integration

```
class LLMContextManager(dspy.Module):
    def __init__(self):
        super().__init__()

        # FIXED: Azure OpenAI compatible DSPy signatures
        self.reason_about_context = dspy.ChainOfThought(
            "context_summary, new_items -> reasoning_analysis"
        )
        self.generate_hypotheses = dspy.ChainOfThought(
            "evidence_items, business_context -> hypothesis_suggestions"
        )
```

Reasoning Prompts

Structured prompts for: **• Context analysis**: Evaluating new evidence against existing hypotheses **• Hypothesis generation**: Creating initial theories from evidence patterns
• Confidence assessment: Determining evidence strength and hypothesis support

Instructions embedded in input data for Azure OpenAI compatibility rather than separate instruction parameters.

Investigation Lifecycle Management

Complete Investigation Tracking

```
# Start new investigation
investigation_id = engine.clear_context("customer_analysis")

# Evidence accumulation and reasoning...

# Retrieve results
results = engine.get_investigation_results(investigation_id)
```

Investigation Results Storage

Results include:

- Final hypothesis states and confidence levels
- Evidence summary and source breakdown
- Reasoning cycle statistics
- Timestamps and processing metrics
- Complete reasoning audit trail

Web Service Architecture

FastAPI Integration

Production-ready HTTP API with:

- RESTful endpoint design
- Comprehensive request/response validation
- OpenAPI documentation with interactive testing
- Webhook integration support
- Health monitoring and debug endpoints

Key API Endpoints

```
POST /evidence              # Add evidence to reasoning engine
POST /hypothesis            # Add hypothesis for consideration
POST /context/clear         # Clear context and start new investigation
GET  /investigations/{id}   # Retrieve investigation results
POST /query                # Natural language query interface
GET  /status               # Engine status and performance metrics
POST /force_stop           # Human override for stuck reasoning
POST /config/max_loops     # Configure failsafe parameters
```

Webhook Support

- **Salesforce integration**: Automatic evidence from CRM events
- **Monitoring systems**: Alert processing from DataDog, New Relic, etc.
- **Extensible framework**: Custom webhook development

Performance Characteristics

Timing Benchmarks

• **Evidence ingestion**: < 10ms average response time • **Context processing**: < 50ms for evidence categorization and queuing • **Deep thought sessions**: 10-60 seconds for complete evidence analysis • **Reasoning cycles**: 1-5 seconds per cycle within deep thought • **API response times**: < 100ms for non-reasoning operations • **Investigation management**: < 100ms for result capture and storage

Memory Management

• **Context compression**: Automatic removal of low-importance items when token limits approached • **Hash history**: Rolling window of recent context states for oscillation detection • **Queue management**: Bounded queues with overflow handling • **Investigation storage**: Persistent results with configurable TTL

Scaling Considerations

• **Context window limits**: Configurable token budgets with intelligent compression • **Reasoning cycle limits**: Hard caps on reasoning duration and cycle count • **Multiple failsafe layers**: Simple loop counter + hash-based + sterile cycles + oscillation detection • **Thread safety**: RLock protection for context modifications

Production Deployment

Configuration Parameters

```
max_context_tokens: int = 4000          # LLM context window budget
compression_threshold: int = 3200       # Trigger context compression
max_sterile_cycles: int = 3             # Reasoning termination threshold
max_reasoning_cycles: int = 10          # Hard reasoning limit (legacy)
max_reasoning_loops: int = 10           # Simple loop counter failsafe
oscillation_detection_window: int = 6   # Pattern detection window
max_deep_thought_minutes: int = 5       # Deep thought timeout
```

Environment Variables

```
# Required
AZURE_OPENAI_KEY=your_key
AZURE_OPENAI_ENDPOINT=https://your-endpoint.openai.azure.com/
AZURE_OPENAI_DEPLOYMENT=your-deployment-name

# Optional
AZURE_OPENAI_VERSION=2024-02-15-preview
MAX_REASONING_LOOPS=10
DEEP_THOUGHT_TIMEOUT_MINUTES=5
```

Monitoring and Observability

• **Real-time status**: Engine state, hypothesis counts, reasoning progress • **Performance metrics**: Processing times, reasoning cycle efficiency • **Reasoning audit trails**: Complete hypothesis evolution history • **Oscillation alerts**: Automatic detection of analytical ambiguity • **Health monitoring**: Detailed status assessment with performance warnings • **Investigation analytics**: Historical investigation pattern analysis

Error Handling

• **LLM failure recovery**: Graceful degradation when reasoning services unavailable • **Context corruption protection**: Hash validation and rollback capabilities • **Thread synchronization**: Deadlock prevention and timeout handling • **Multiple failsafe layers**: Prevents infinite reasoning through redundant mechanisms

API Interface

Primary Operations

```
engine.add_evidence(content: str, source: str, confidence: float) -> str
engine.add_hypothesis(content: str, confidence: float) -> str
investigation_id = engine.clear_context(base_name: str) -> str
results = engine.get_investigation_results(investigation_id: str) -> Dict
engine.query_context_sync(query: str, timeout: float) -> str
engine.get_status_snapshot() -> Dict[str, Any]
engine.force_stop_reasoning(reason: str) -> None
engine.configure_max_loops(max_loops: int) -> None
```

RESTful HTTP API

```
# Start investigation
curl -X POST "http://localhost:8000/context/clear" \
  -H "Content-Type: application/json" \
  -d '{"base_name": "q4_analysis"}'

# Add evidence
curl -X POST "http://localhost:8000/evidence" \
  -H "Content-Type: application/json" \
  -d '{
    "content": "Q4 sales exceeded targets by 30%",
    "source": "sales_system",
    "confidence": 0.85
  }'

# Query reasoning
curl -X POST "http://localhost:8000/query" \
  -H "Content-Type: application/json" \
  -d '{"query": "What evidence supports enterprise focus?"}'
```

Event Callbacks

• **Reasoning state changes**: Deep thought entry/exit notifications • **Hypothesis evolution**: Confidence updates and status transitions • **Oscillation detection**: Analytical ambiguity alerts • **Context compression**: Memory management notifications • **Investigation completion**: Result capture and storage notifications

Extensions and Future Work

Planned Enhancements

• **Hypothesis merging**: Automatic combination of similar theories • **Evidence source weighting**: Dynamic confidence adjustment based on source reliability • **Multi-domain reasoning**: Parallel reasoning across different knowledge domains • **Temporal reasoning**: Time-based evidence decay and trend analysis • **SQLite integration**: Replace stub investigation storage with full database • **Investigation analytics**: Pattern analysis across historical investigations

Integration Patterns

• **Message queue integration**: Kafka/RabbitMQ for high-volume evidence streams • **Database persistence**: Context state durability and recovery
• **Microservice deployment**: Containerized reasoning services with load balancing • **Real-time dashboards**: Live reasoning state visualization • **Advanced webhook framework**: Custom integrations for enterprise systems

---

Version 1.0