Mach33 Candidate Exercise: Multi-Agent Research System

Examiner : J. Brant Arseneau

Introduction

This exercise challenges you to build a small multi-agent research application that demonstrates key concepts from the Mach33Agents system. You will implement a series of components that build upon each other, creating a functional research system with multiple specialized AI agents working together.The exercise is divided into six progressive parts, each focusing on a critical aspect of AI application development. By completing all parts, you will create a small but functional research tool that showcases your understanding of hierarchical contexts, agent configuration, conversation management, memory systems, knowledge representation, and automated report generation.

Time Allocation

This exercise is designed to be completed in approximately 6-8 hours. You may take more time if needed, but please focus on meeting the core requirements before adding extra features.

Technology Requirements

- JavaScript/TypeScript for implementation
- React (preferred) or Vue for UI components
- MongoDB for context, agent, conversation, and memory storage
- Neo4j for knowledge graph and entity relationship management
- Node.js backend for database connectivity
- Integration with multiple LLM services:
  - OpenAI API (GPT-4, GPT-3.5)
  - Anthropic API (Claude 3 models)
  - Local models via Ollama (optional)
  - Open source models via Hugging Face (optional)
- Docker Compose for local development environment setup

LLM Integration Requirements

- The system must support connecting to multiple LLM providers
- Each agent should be configurable to use different models based on their role
- The system must implement model fallback mechanisms for resilience
- API keys and model configurations must be stored securely
- Prompt templates must be adaptable to different model capabilities
- The system should track token usage and optimize for efficiency

Database Setup Requirements

MongoDB Configuration

- Create collections for: Contexts, Agents, Conversations, Messages, and Memories
- Implement proper indexing for efficient queries
- Use MongoDB aggregation pipelines for context inheritance resolution
- Structure documents to support hierarchical relationships

Neo4j Configuration

- Create node types for: Entities, Topics, and Concepts
- Implement relationship types for various entity connections
- Use Cypher queries for knowledge graph traversal and visualization
- Structure graph to support confidence scoring and relationship weights

Exercise Overview

You will build a research application with the following components:

1. Hierarchical Context System: Create the foundation for the multi-agent system
1. Agent Configuration: Configure specialized research agents with different roles
1. Conversation Engine: Implement a conversation system between agents and user
1. Memory Generation: Extract and store information from conversations
1. Research Workflow: Connect everything into a functional research tool
1. Report Generation: Create an authoring system that generates comprehensive reports from research memories

Part 1: Hierarchical Context System

Value to Users: This system enables researchers to organize complex information in intuitive nested structures, similar to how human experts mentally organize domain knowledge. Users can quickly navigate between different levels of abstraction without losing context, dramatically improving research efficiency and insight discovery.Scientific Context: This component draws from cognitive science theories of hierarchical knowledge organization, particularly schema theory and mental models. It implements computational approaches to context management found in artificial intelligence, specifically hierarchical knowledge representation and inheritance-based reasoning systems used in knowledge-based AI architectures.

Software Requirements Specification

1.1 Purpose

Implement the core hierarchical context system that forms the foundation of the Mach33Agents architecture. This system enables properties to be inherited from parent contexts to child contexts, with overrides possible at any level.

1.2 Functional Requirements

1.2.1 Context Hierarchy

- The system must support a four-level hierarchy: Domain → Project → Room → Agent
- Each context level must store its level type, ID, name, and properties
- Each context (except Domain) must reference its parent context
- The system must support creating, retrieving, and modifying contexts at any level

1.2.2 Property Inheritance

- Properties defined at parent levels must be automatically inherited by child levels
- Child levels must be able to override properties defined in parent contexts
- The system must track which level defined each property (origin tracking)
- The system must support getting all effective properties for a context (local + inherited)

1.2.3 Property Types

- The system must support the following property types:
  - String properties (e.g., name, description)
  - Numeric properties (e.g., importance, confidence)
  - Boolean properties (e.g., isActive, isVisible)
  - Array properties (e.g., knowledgeDomains, allowedTools)
  - Object properties (e.g., traits, parameters)

1.2.4 MongoDB Integration

- Contexts must be stored in a MongoDB collection with appropriate schema

- The system must implement efficient queries for context hierarchy traversal
- Property inheritance must be resolved using MongoDB aggregation pipelines
- The system must support transactions for context modifications
- Queries must be optimized with proper indexing

1.2.5 LLM Model Properties

- The context system must include properties for LLM model configuration
- Each context level must be able to specify model preferences
- Model configuration properties must include:
  - provider: The LLM service provider (OpenAI, Anthropic, etc.)
  - model: Specific model name/version
  - temperature: Creativity setting
  - maxTokens: Response length limit
  - contextWindow: Available context size
- Model properties must be inheritable and overridable

1.2.6 Basic UI

- The application must provide a simple UI to:
  - View the context hierarchy
  - Select a context to view its properties
  - Add or modify properties on a selected context
  - Visualize which properties are inherited vs. locally defined
  - Configure LLM model settings at each level

1.3 Deliverables

- Complete implementation of the context system classes
- MongoDB schema and query implementations
- LLM model configuration integration
- Simple UI for managing contexts and properties
- Unit tests demonstrating property inheritance
- Docker setup for MongoDB instance

Part 2: Agent Configuration & Personality

Value to Users: This component allows users to create specialized AI agents with distinct personalities and capabilities tailored to specific research tasks. Users can configure agents that match their research needs without coding skills, enabling even non-technical researchers to create powerful AI research assistants optimized for different cognitive tasks.Scientific Context: This system builds upon cognitive modeling and artificial personality theories from cognitive psychology and AI research. It implements

computational models of personality traits similar to those used in cognitive architectures and agent-based modeling. The approach is related to psychometric models of human personality (like the Five Factor Model) adapted for AI agent behavior and specialized for cognitive tasks in research contexts.

Software Requirements Specification

2.1 Purpose

Extend the context system to support specialized agent types with configurable personality traits and parameters. This allows creating agents with different roles and behaviors within the research system.

2.2 Functional Requirements

2.2.1 Agent Types

- The system must support four agent types:
  - Researcher: Focuses on information gathering and discovery
  - Analyzer: Specializes in critical evaluation and pattern recognition
  - Synthesizer: Combines information and generates insights
  - Author: Specializes in organizing and presenting information clearly
- Each agent type must have default traits appropriate to its role
- Agent type must be stored as a property in the context system

2.2.2 Personality Traits

- Each agent must have configurable personality traits stored as properties
- Required trait dimensions:
  - Curiosity (0.0-1.0): Interest in exploring new information
  - Thoroughness (0.0-1.0): Attention to detail and completeness
  - Creativity (0.0-1.0): Ability to generate novel ideas
  - Analytical (0.0-1.0): Capacity for critical thinking
  - Communication (0.0-1.0): Clarity and effectiveness of expression
- The Author agent must have additional traits:
  - Structure (0.0-1.0): Ability to organize information logically
  - Clarity (0.0-1.0): Capacity for clear explanations
  - Persuasiveness (0.0-1.0): Ability to present compelling arguments
- Traits must be inherited and overridable through the context system

2.2.3 LLM Model Integration

- Each agent type must have recommended LLM models suited to their role:
  - Researcher: Models with strong knowledge retrieval capabilities
  - Analyzer: Models with strong reasoning and critical thinking

    - Synthesizer: Models with creative and integrative capabilities
    - Author: Models with excellent writing and organization skills
- The system must allow assigning different models to different agents
- The interface must provide model recommendations based on agent type
- The system must support customizing prompt templates for each model

2.2.4 MongoDB Schema Extensions

- Extend the Context collection schema to support agent-specific fields
- Add fields for LLM model configurations per agent
- Implement indexes for efficient agent querying by type and traits
- Create aggregation pipelines for trait inheritance resolution
- Implement queries for finding agents with specific trait profiles

2.2.5 Configuration Interface

- The application must provide a UI to:
    - Create and configure agents of different types
    - Adjust personality traits with sliders or input fields
    - Select and configure LLM models for each agent
    - Visualize which traits are inherited vs. locally defined
    - Preview the agent's configured personality

2.2.6 Parameter Inheritance Visualization

- The interface must clearly show which parameters come from which context level
- Different visual indicators must distinguish local vs. inherited properties
- The interface must allow overriding inherited properties
- Model configuration inheritance must be clearly visualized

2.3 Deliverables

- Agent configuration classes extending the context system
- LLM model integration for different agent types
- MongoDB schema extensions and query implementations
- Configuration UI for creating and modifying agents
- Model selection and configuration interface
- Visual indicators for property inheritance
- Tests demonstrating trait inheritance and override capabilities

Part 3: Conversation Engine

Value to Users: This component provides a natural interface for interacting with multiple specialized AI agents simultaneously. Users can engage in dynamic conversations where different agents contribute unique perspectives, mimicking a collaborative research team. This approach reduces cognitive load for researchers while increasing the diversity of insights generated.Scientific Context: This system implements principles from conversation analysis, discourse theory, and multi-agent systems research. It draws on computational linguistics and dialogue management systems used in advanced conversational AI. The turn-taking mechanics are informed by research on human group dynamics and collaborative knowledge construction, adapted for human-AI and AI-AI interaction patterns.

Software Requirements Specification

3.1 Purpose

Implement a conversation system that enables interaction between the user and multiple agents. This system must support turn-based conversations with agents responding based on their configured personalities.

3.2 Functional Requirements

3.2.1 Message Management

- The system must store and display conversation messages in chronological order
- Messages must have: sender ID, content, timestamp, and metadata
- The system must support different message types: user, agent, and system
- The conversation must persist in MongoDB for retrieval across sessions

3.2.2 Multi-Model Conversation Flow

- The system must support conversations with agents using different LLM models
- Each agent must generate responses using its configured model
- The system must handle different response formats from different models
- Messages must include metadata about which model generated each response
- The system must implement fallback mechanisms if a model is unavailable

3.2.3 MongoDB Schema for Conversations

- Create a Conversations collection to store conversation metadata
- Create a Messages collection to store individual messages
- Include fields for LLM model information in message metadata
- Implement appropriate indexes for efficient message retrieval
- Support pagination and efficient loading of large conversations
- Implement real-time updates using MongoDB change streams

3.2.4 Conversation Flow

- The system must implement a round-robin conversation flow between agents
- When a user sends a message, agents should respond in sequence
- The order of agent responses should be configurable
- The system must support at least 3 agents in a single conversation
- The system must optimize token usage across multiple LLM calls

3.2.5 Agent Response Generation

- Agent responses must reflect their configured personality traits and type
- Each agent must use its assigned LLM model for response generation
- The system must translate personality traits into model parameters
- The system must construct appropriate prompts based on agent type
- Responses must be formatted differently based on agent type
- The system must simulate different response patterns:
  - Researcher: Information discovery and questions
  - Analyzer: Critical evaluation and pattern identification
  - Synthesizer: Connection-making and summary
  - Author: Clarity-focused explanation and organization

3.2.6 Conversation UI

- The interface must provide:
  - A message list showing the conversation history
  - Indicators for which agent is speaking
  - Indicators showing which LLM model generated each response
  - Visual distinction between message types
  - A typing indicator when agents are "thinking"
  - Message formatting with Markdown support
  - A simple input mechanism for user messages
  - Model status indicators (availability, response time)

3.3 Deliverables

- Conversation management system
- LLM integration for agent responses
- MongoDB schema and query implementations for conversations
- Round-robin agent conversation implementation
- Agent response generation based on type, personality, and assigned model
- Conversation UI with message history and input
- Tests for conversation flow and model integration

Part 4: Memory Generation

Value to Users: This system automatically extracts, organizes, and preserves valuable insights from research conversations. Users no longer need to manually take notes or worry about losing important information during brainstorming sessions. The system creates a persistent, structured knowledge base that grows organically through natural conversation, dramatically reducing information loss and improving research continuity.Scientific Context: This component implements principles from cognitive psychology research on memory formation, particularly episodic and semantic memory models. It utilizes information extraction techniques from natural language processing combined with knowledge representation approaches from cognitive science. The confidence and importance scoring systems are informed by research on human memory salience, adapted to computational contexts for optimal information retrieval.

Software Requirements Specification

4.1 Purpose

Create a system that extracts and stores structured memories from conversations. These memories represent the knowledge gained during research and can be referenced by agents in future interactions.

4.2 Functional Requirements

4.2.1 Memory Extraction

- The system must automatically extract structured memories from conversation messages
- Memory extraction must leverage specialized LLM models for information extraction
- Each memory must include: content, type, source, timestamp, confidence, and importance
- The system must support at least four memory types:
  - Facts: Objective information with high confidence
  - Insights: Interpretations or conclusions
  - Questions: Uncertainties requiring further research
  - Actions: Decisions or tasks to be performed
- The system must track which LLM model extracted each memory

4.2.2 Memory Classification

- The system must classify extracted memories by type
- Classification must use a specialized LLM model optimized for categorization
- Classification must consider message content and context
- The system must assign confidence scores to each memory
- The system must calculate importance scores based on type and confidence
- Classification quality must be tracked to improve future extractions

4.2.3 LLM Model Selection for Memory Tasks

- The system must use appropriate specialized models for memory-related tasks:
  - Extract specialized models for precise information extraction
  - Classify models for accurate categorization
  - Summarize models for condensing lengthy content
  - Evaluate models for confidence scoring
- The system must allow configuring which models to use for different memory tasks
- The system must implement fallbacks if preferred models are unavailable

4.2.4 MongoDB Schema for Memories

- Create a Memories collection with appropriate schema
- Include fields for LLM model information used in extraction and classification
- Implement indexes for efficient memory retrieval by type, source, and importance
- Support querying memories by content similarity
- Implement efficient aggregation for memory statistics
- Store confidence scores and importance calculations

4.2.5 Memory Management

- The system must store all generated memories in MongoDB
- The system must support filtering memories by:
  - Type (fact, insight, question, action)
  - Source (which agent or user generated it)
  - Importance (minimum threshold)
  - Model used for extraction
- Memories must persist across application sessions

4.2.6 Memory Visualization

- The interface must provide a panel to view extracted memories
- Memories must be displayed with visual indicators for type, source, and importance
- The panel must indicate which LLM model extracted each memory
- The panel must support sorting and filtering memories
- The visualization must update in real-time as new memories are generated

4.3 Deliverables

- Memory extraction and classification system
- LLM integration for memory-related tasks
- MongoDB schema and query implementations for memories
- Memory storage and retrieval functionality
- Memory visualization panel with filtering options

- Tests demonstrating memory extraction from different message types
- Performance comparison of different models for memory tasks

Part 5: Research Workflow Integration

Value to Users: This component guides researchers through a structured, yet flexible research process that maximizes productivity and insight generation. Users benefit from a scaffolded approach to complex research tasks, with specialized phases that ensure thoroughness while maintaining efficiency. The system helps researchers maintain focus on different cognitive aspects of research at appropriate times, leading to more comprehensive and rigorous outcomes.Scientific Context: This system implements principles from scientific methodology and research design frameworks, particularly phased approaches to inquiry and discovery. It draws on cognitive load theory and task switching research from cognitive psychology to optimize the research process. The entity extraction and knowledge graph components utilize techniques from computational knowledge representation, ontology engineering, and semantic networks to create machine-understandable representations of research domains.

Software Requirements Specification

5.1 Purpose

Connect all previous components into a cohesive research workflow that guides users through a structured research process with multi-agent collaboration.

5.2 Functional Requirements

5.2.1 Research Topic Management

- The system must allow users to specify a research topic
- The topic must be shared across all agents in the conversation
- The system must maintain a list of previous research topics
- The system must support reloading previous research sessions

5.2.2 Research Phases

- The workflow must support four research phases:
  - Research: Initial information gathering (led by Researcher agent)
  - Analysis: Critical evaluation of findings (led by Analyzer agent)
  - Synthesis: Integration of information (led by Synthesizer agent)
  - Report: Generation of a research summary (led by Author agent)
- The system must track the current phase and transition between phases
- Each phase must have phase-specific agent prompts and behaviors

- Each phase should utilize appropriate LLM models optimized for that phase's tasks

5.2.3 Entity Extraction and Knowledge Graph

- The system must extract entities (concepts, people, organizations) from conversations
- Entity extraction should use specialized models for named entity recognition
- Extracted entities must be stored in Neo4j with confidence scores and relationships
- The system must build a knowledge graph connecting related entities
- The knowledge graph must update as new entities are discovered

5.2.4 Neo4j Integration

- Implement a Neo4j schema with appropriate node and relationship types
- Create Cypher queries for entity creation and relationship management
- Implement efficient graph traversal for entity relationship discovery
- Support confidence scoring and relationship weighting
- Provide visualizations of the knowledge graph structure

5.2.5 Research Phase Management

- The system must track the current research phase
- The interface must display the current phase and progress
- The system must provide phase transition controls
- Each phase must have specific visualization and UI elements
- The system should optimize LLM model selection for each phase

5.2.6 Integrated UI

- The interface must integrate all components:
  - Context hierarchy and agent configuration
  - Conversation panel with agent responses
  - Memory visualization panel
  - Knowledge graph visualization (using Neo4j data)
  - Research phase tracking
  - Report request controls
  - Model selection and status indicators

5.3 Deliverables

- Complete research workflow implementation
- LLM integration optimized for each research phase
- Neo4j schema and query implementations for entity graph
- Entity extraction and knowledge graph functionality
- Research phase management system
- Fully integrated UI with all components

- End-to-end test demonstrating a complete research session
- Docker Compose setup with MongoDB and Neo4j instances

Part 6: Report Generation System

Value to Users: This component transforms raw research data into polished, insightful reports tailored to different audiences and purposes. Users save countless hours of report writing while benefiting from AI-powered organization of complex information. The system ensures no valuable insights are overlooked and presents information in optimal formats for different stakeholders, from executive summaries to comprehensive technical reports.Scientific Context: This system applies principles from document generation, information design, and technical communication research. It implements computational approaches to narrative structure, rhetorical organization, and information hierarchy used in natural language generation systems. The report format variations draw on cognitive research about information consumption patterns for different audiences and purposes, while the interactive features utilize principles from human-computer interaction to enhance information exploration and comprehension.

Software Requirements Specification

6.1 Purpose

Create a comprehensive report generation system using the Author agent to transform research memories and knowledge into well-structured, insightful reports with various formats and detail levels.

6.2 Functional Requirements

6.2.1 Author Agent Specialization

- The Author agent must be specialized for report generation
- The agent must use high-quality LLM models optimized for long-form content creation
- The agent must analyze memories and knowledge graph data
- The agent must organize information logically and cohesively
- The agent must generate reports that reflect its personality traits
- Author agent configuration must allow customization of report style and format

6.2.2 Memory Analysis and Organization

- The system must analyze all collected memories from the research session
- Analysis should use specialized models for pattern recognition and thematic analysis
- Memories must be grouped by topic and relevance
- The system must identify key themes and patterns across memories
- The system must prioritize memories based on importance and confidence

- Contradictory memories must be flagged and handled appropriately

6.2.3 Report Structure Generation

- The system must generate logical report structures based on:
  - Research topic complexity
  - Available memory types and quantities
  - Entity relationships from the knowledge graph
  - Research phase contributions
- Structure generation should use specialized models for content organization
- Report structures must include:
  - Executive summary
  - Key findings section
  - Detailed analysis sections
  - Entity relationship explanations
  - Unanswered questions section
  - Recommendations section

6.2.4 Report Formats and Detail Levels

- The system must support multiple report formats:
  - Executive Brief: Short, high-level summary (1-2 pages)
  - Standard Report: Balanced detail and comprehensiveness (5-7 pages)
  - Comprehensive Analysis: In-depth exploration of all aspects (10+ pages)
- Each format must have appropriate section organization and detail levels
- The system must support both narrative and bullet-point presentation styles
- Reports must include automatically generated visualizations:
  - Knowledge graph visualization
  - Entity relationship diagrams
  - Confidence/importance matrices for key findings

6.2.5 LLM Model Selection for Report Generation

- The system must select appropriate models for different report components:
  - Summary models for executive summaries
  - Narrative models for detailed explanations
  - Technical models for specialized content
  - Creative models for engaging introductions and conclusions
- The system must allow configuration of which models to use for each report component
- The system must optimize token usage across report sections
- The system must track model performance for different report types

6.2.6 MongoDB Schema for Reports

- Create a Reports collection with appropriate schema
- Include fields for LLM models used in report generation
- Store report metadata, structure, and content
- Implement version control for report iterations
- Track relationships between reports and source memories
- Support efficient querying and filtering of reports

6.2.7 Report Generation UI

- The interface must provide:
  - Report configuration options (format, detail level, style)
  - Model selection for report generation
  - Real-time report generation progress indicators
  - Preview of report sections as they are generated
  - Export options (PDF, HTML, Markdown)
  - Report history and version comparison
  - Feedback mechanism for report quality

6.2.8 Interactive Report Features

- The system must support interactive elements in reports:
  - Expandable/collapsible sections
  - Tooltips for entity information
  - Interactive knowledge graph visualization
  - Source memory references with direct links
  - Confidence indicators for findings
  - Model attribution for different sections

6.3 Deliverables

- Author agent specialization implementation
- LLM integration for report generation tasks
- Memory analysis and organization system
- Report structure generation algorithm
- Multiple report format implementations
- MongoDB schema for reports
- Report generation UI with model selection
- Interactive report features
- Export functionality
- Tests for report generation and quality

Database Configuration

MongoDB Setup

The application must use MongoDB (version 4.4+) with the following collections:

1. Contexts: Stores all context entities (Domain, Project, Room, Agent)
   - Required fields: *id, name, level, parentId, properties*
1. Conversations: Stores conversation metadata
   - Required fields: *id, title, createdAt, updatedAt, participantIds, status*
1. Messages: Stores individual conversation messages
   - Required fields: *id, conversationId, sender, content, timestamp, metadata*
1. Memories: Stores extracted memories from conversations
   - Required fields: *id, content, type, source, timestamp, confidence, importance*
1. Reports: Stores generated research reports
   - Required fields: *id, title, format, detailLevel, structure, content, createdAt, updatedAt, sourceMemoryIds, versionNumber*
1. ModelConfigs: Stores LLM model configurations
   - Required fields: *id, provider, model, parameters, capabilities, usage*

Neo4j Setup

The application must use Neo4j (version 4.3+) with the following node types:

1. Entity: Represents concepts, people, organizations discovered in research
   - Required properties: id, name, type, confidence, mentions
1. Relationship Types:
   - RELATED_TO: Basic relationship between entities
   - PART_OF: Hierarchical relationship
   - CONTRADICTS: Conflicting information
   - SUPPORTS: Supporting evidence

Docker Setup Requirements

The application must include a Docker Compose configuration with:

1. MongoDB service
1. Neo4j service
1. Node.js API service
1. React frontend service

1. Appropriate volume configurations for data persistence
1. Network configuration for service communication

Evaluation Criteria

Your submission will be evaluated based on the following criteria:

1. Functionality: Does your implementation meet all the requirements?
1. Database Design: Have you implemented effective MongoDB and Neo4j schemas?
1. LLM Integration: How effectively have you integrated and utilized different models?
1. Code Quality: Is your code well-structured, readable, and maintainable?
1. System Architecture: How well have you designed the overall system?
1. UI/UX: Is your interface intuitive, responsive, and visually clear?
1. Performance: Are your database queries and operations optimized?
1. Innovation: Have you added any creative touches or thoughtful improvements?

Submission Instructions

1. Create a GitHub repository with your implementation
1. Include a README with:
    - Setup and running instructions (including Docker configuration)
    - Database schema documentation
    - LLM model integration documentation
    - Overview of your implementation approach
    - Any assumptions or design decisions
    - Known limitations or future improvements
1. Ensure all code is properly commented and documented
1. Include screenshots of your application in action
1. Submit the repository link by the deadline

    We encourage you to focus on meeting the core requirements first, then add extra features or improvements if time permits. Good luck!