Mycroft Orchestrator: Technical Documentation

The Mycroft Orchestrator is an intelligent multi-agent coordination system that leverages natural language processing to automatically route user queries to specialized financial and patent intelligence workflows. Built on n8n workflow automation platform and powered by Llama 3, it serves as a central dispatch hub that interprets user intent and delegates tasks to appropriate analytical agents.

Overview

What is the Mycroft Orchestrator?

The Mycroft Orchestrator is a sophisticated routing layer that sits between users and specialized analytical tools. Instead of requiring users to know which tool to use and how to format their requests, the orchestrator accepts natural language queries and intelligently determines:

- Which specialized agent should handle the request
- What parameters are needed
- How to properly format and validate the request
- Where to route the request for processing

Key Benefits

Natural Language Interface: Users can ask questions in plain English without needing to understand underlying API structures or parameter formats.

Intelligent Routing: The system automatically identifies the appropriate tool based on query context and intent.

Type-Safe Validation: All parameters are validated against defined schemas before routing, preventing errors and ensuring data integrity.

Local Processing: Uses locally-hosted Llama 3 for privacy and control over the language model.

Extensible Design: New tools and agents can be easily added to the orchestration system.

Architecture

High-Level Flow

User Query → Chat Interface (Webhook Trigger) → LLM Analyzer (Ollama Llama 3) → JSON Parser & Validator → Parameter Validation & URL Construction → HTTP Router (Dynamic Endpoint) → Specialized Agent (Patent/Financial/News/Forecast)

Component Breakdown

Chat Trigger: Entry point that receives user queries through a public webhook endpoint. Provides the conversational interface.

Ollama Chat Model: Local LLM (Llama 3) that processes natural language and determines intent. Configured to analyze queries and extract structured information.

The Brain (LLM Chain): Specialized prompt that instructs the LLM to act as a tool dispatcher. Contains detailed descriptions of all available tools and their parameter schemas.

The Processor (Code Node): JavaScript-based validation engine that:

- Parses LLM output into structured JSON
- Validates parameters against tool schemas
- Constructs properly formatted URLs with query parameters
- Handles errors and edge cases

The Router (HTTP Request): Dynamic HTTP client that dispatches validated requests to the appropriate agent webhook endpoints.

Available Tools & Capabilities

1. Patent Monitor

Purpose: Monitors and analyzes recent patent filings from the United States Patent and Trademark Office (USPTO).

Endpoint: `http://localhost:5678/webhook/patent-monitor`

Parameters:

- `days_back` (integer, required) - Number of days to look back for patent filings
- `email` (string, optional) - Email address for report delivery

Example Queries:

- "Show me patents from the last 7 days"
- "Monitor patents filed in the last 30 days"
- "Get patent filings from the past 2 weeks"

Use Cases:

- Competitive intelligence monitoring
- Technology trend analysis
- Prior art research
- Innovation landscape mapping

---

2. Financial Metrics

Purpose: Analyzes SEC filings (10-K annual reports, 10-Q quarterly reports) for comprehensive financial insights.

Endpoint: `http://localhost:5678/webhook/sec-metrics`

Parameters:

- `ticker` (string, required) - Stock ticker symbol (e.g., "AAPL", "TSLA", "MSFT")
- `narrative` (boolean, required) - Whether to include narrative analysis from Management Discussion & Analysis (MD&A) sections

Example Queries:

- "Get financial metrics for AAPL"
- "Analyze TSLA with detailed insights"
- "Show me numbers for Microsoft"
- "Give me a full financial report on NVDA"

Intelligence: The system automatically determines whether narrative analysis is needed based on keywords like "detailed", "insights", "analysis", "full report" in the query.

---

3. News Sentiment

Purpose: Aggregates recent news articles and performs sentiment analysis using FinBERT, a financial domain-specific language model.

Endpoint: `http://localhost:5678/webhook/news-sentiment`

Parameters:

- `company` (string, required) - Company name to search for in news sources
- `email` (string, optional) - Email address to send the sentiment report

Example Queries:

- "What's the sentiment on Amazon lately?"
- "Analyze news about Microsoft"
- "How is Tesla being portrayed in the media?"

Use Cases:

- Market sentiment tracking
- Reputation monitoring
- Event impact analysis
- Investment decision support

---

4. Forecasting

Purpose: Generates price forecasts using historical market data combined with sentiment analysis.

Endpoint: `http://localhost:5678/webhook/forecasting`

Parameters:

- `ticker` (string, required) - Stock ticker symbol to forecast
- `email` (string, required) - Email address to receive forecast results
- `horizon` (integer, optional, default: 30) - Number of days to forecast into the future

Example Queries:

- "Forecast NVDA for the next 45 days"
- "Predict Tesla stock for 60 days"
- "What will Apple stock do over the next month?"

Technical Implementation

Workflow Node Analysis

Node 1: When chat message received

```
{
  "type": "@n8n/n8n-nodes-langchain.chatTrigger",
  "parameters": {
    "public": true,
    "initialMessages": "Give financial metrics for AAPL",
    "options": {}
  }
}
```

This node creates a public webhook that accepts chat messages. The `initialMessages` parameter provides a default example query to help users understand the system's capabilities.

Node 2: Ollama Chat Model

```
{
  "type": "@n8n/n8n-nodes-langchain.lmChatOllama",
  "parameters": {
    "model": "llama3:latest",
    "options": {}
  }
}
```

Connects to a locally-running Ollama instance serving the Llama 3 model. This provides the natural language understanding capability without requiring external API calls.

Node 3: The Brain (LLM Chain)

This is the system's intelligence core. It contains a sophisticated prompt that:

1. **Defines the LLM's role**: "You are a tool dispatcher"
2. **Specifies output format**: JSON-only responses with no markdown or explanations
3. **Documents all available tools**: Comprehensive descriptions of each tool's purpose and parameters
4. **Provides parameter intelligence**: Special logic for determining the `narrative` parameter based on query keywords
5. **Enforces structure**: Exact JSON schema that must be followed

Key Prompt Features:

- Clear separation of required vs. optional parameters
- Examples of valid ticker symbols
- Semantic understanding rules (e.g., "insights" → narrative: true)
- Strict formatting requirements to ensure parseable output

Node 4: The Processor (Code Node)

This JavaScript node performs several critical functions:

1. JSON Extraction & Cleaning

```
raw = raw
  .replace(/```(json)?/gi, '')  // Remove markdown code blocks
  .replace(/\\n/g, '')          // Remove escaped newlines
  .replace(/\n/g, '')           // Remove newlines
  .trim();
```

2. Tool Configuration Management

```
const toolConfigs = {
  "patent-monitor": {
    url: "http://localhost:5678/webhook/patent-monitor",
    schema: {
      days_back: "number",
      email: "string"
    }
  },
  // ... other tools
};
```

3. Type-Safe Validation

```
function validateParams(tool, params) {
  // Checks each parameter against expected type
  // Returns validation result with error details if invalid
}
```

4. URL Construction

```
function buildUrlWithParams(baseUrl, params) {
  // Manually constructs query string
  // Properly encodes parameter names and values
  // Returns complete URL ready for HTTP request
}
```

5. Error Handling

- JSON parsing failures
- Unknown tool names
- Type mismatches
- Missing required parameters

Node 5: The Router (HTTP Request)

Uses n8n's HTTP Request node with dynamic URL configuration:

```
{
  "parameters": {
    "url": "={{ $json.url }}"
  }
}
```

The `={{ $json.url }}` expression pulls the fully-constructed URL from the Processor node's output, enabling dynamic routing to different agent endpoints.

Setup & Installation

Prerequisites

Before setting up the Mycroft Orchestrator, ensure you have:

1. **n8n workflow automation platform** (self-hosted or cloud)
2. **Ollama** installed with Llama 3 model
3. **Specialized agent workflows** deployed and accessible
4. **Network connectivity** between all components

Step 1: Install Ollama and Llama 3

```
# Install Ollama (Linux/macOS)
curl -fsSL https://ollama.com/install.sh | sh

# Pull Llama 3 model
ollama pull llama3:latest

# Verify installation
ollama list
```

Windows: Download the installer from https://ollama.com/download

Step 2: Start Ollama Service

```
# Start Ollama server (runs on port 11434 by default)
ollama serve
```

Verify it's running by visiting: `http://localhost:11434`

Step 3: Configure n8n Credentials

1. Navigate to n8n credentials settings
2. Create a new "Ollama API" credential
3. Configure connection settings:
    - **Base URL**: `http://localhost:11434`
    - Leave authentication empty for local installations

Step 4: Deploy Agent Workflows

Ensure the following webhook endpoints are active in your n8n instance:

- `http://localhost:5678/webhook/patent-monitor`
- `http://localhost:5678/webhook/sec-metrics`
- `http://localhost:5678/webhook/news-sentiment`
- `http://localhost:5678/webhook/forecasting`

Note: Each agent must be a separate n8n workflow configured with webhook triggers.

Step 5: Import Orchestrator Workflow

1. Copy the provided JSON configuration
2. In n8n, go to **Workflows → Import from File/URL**
3. Paste the JSON or upload the file
4. Save the workflow

Step 6: Update Credentials

After importing, connect the Ollama Chat Model node to your Ollama API credential:

1. Double-click the "Ollama Chat Model" node
2. Select your configured Ollama credential from the dropdown
3. Save the node

Step 7: Activate the Workflow

Click the **Active** toggle in the top-right corner to enable the orchestrator.

Step 8: Get the Chat URL

1. Click on the "When chat message received" node
2. Copy the webhook URL provided
3. This URL is your chat interface endpoint

Testing Your Installation

Send a test query to the chat endpoint:

```
curl -X POST http://localhost:5678/webhook/{your-webhook-id} \
  -H "Content-Type: application/json" \
  -d '{"chatInput": "Get financial metrics for AAPL"}'
```

You should receive a response indicating the request was routed successfully.

Usage Guide

Interacting with the Orchestrator

The orchestrator accepts natural language queries through its chat interface. You can access it via:

1. **Direct webhook URL**: Post JSON payloads with `chatInput` field
2. **n8n Chat Widget**: Embed the chat interface in a webpage
3. **API Integration**: Connect your application to the webhook endpoint

Query Patterns

Patent Monitoring:

- "Show me patents from the last 7 days"
- "Monitor patents filed in the last month"

Financial Analysis:

- "Get financial metrics for AAPL" (metrics only)
- "Analyze Tesla with detailed insights" (metrics + narrative)

- "Give me a full report on Microsoft" (comprehensive analysis)

Sentiment Analysis:

- "What's the sentiment on Amazon lately?"
- "Analyze news about Apple"

Forecasting:

- "Forecast NVDA for the next 45 days"
- "Predict Tesla for 2 months"

Future Enhancements

Multi-Tool Queries

Sequential Execution: "Get TSLA metrics then forecast for 30 days"

- Parse compound queries
- Execute tools in order
- Pass results between tools

Parallel Execution: "Compare sentiment for AAPL and MSFT"

- Identify independent operations
- Execute simultaneously
- Aggregate results

Context Management

Session Persistence: Remember previous queries in a conversation

- Store user preferences
- Maintain context across interactions
- Enable follow-up questions without repetition

User Profiles: Store preferences per user

- Default parameters (email, timeframes)
- Favorite tools
- Custom query shortcuts

Performance & Reliability

Tool Result Caching: Store recent results to avoid redundant processing

- Cache key: tool + parameters + timestamp
- TTL-based expiration
- Significant speedup for repeated queries

Intent Confidence Scoring: Assess certainty before routing

- Score: 0-100% confidence
- Threshold: Require confirmation for low scores
- Reduces errors from ambiguous queries

Appendix: Technical Specifications

Platform: n8n workflow automation (v1.x)

Language Model: Llama 3 (via Ollama)

Programming: JavaScript (ES6+)

Communication: HTTP/REST

Data Format: JSON

Deployment: Self-hosted / Local

Scalability: Horizontal (add more agent instances)

Authentication: Currently public (can be restricted)

Rate Limiting: Not implemented (recommended for production)