File Breakdown: `src/utils/general_knowledge.py`

File Location

`src/utils/general_knowledge.py`

Overview

The `general_knowledge.py` file implements functionality for answering general knowledge questions and history-related queries in the multi-agent system. It serves as the primary handler for non-specialized questions and for providing context about past interactions, making the system more conversational and context-aware.

Key Responsibilities

- Answer general knowledge questions using LLM capabilities
- Detect and respond to history-related queries about past conversations
- Extract information from session history for contextual responses
- Handle different types of history queries (math, models, general)
- Store answers in the knowledge base
- Format responses appropriately

Core Functionality

Constants

```
LOCATIONS = [
    "Spain", "UK", "France", "Germany", "Italy",
    "Portugal", "Belgium", "Netherlands", "Greece",
    "Croatia", "Sweden", "Norway", "Denmark",
    "Finland", "Ireland", "Switzerland", "Austria"
]
```

Main Function

```
@log_function_call
def answer_general_question(kb: KnowledgeBase, prompt: str, input2="-"):
    """
    Answers general knowledge questions using an LLM.
    FIXED version with improved history detection and retrieval.

    Parameters:
        kb (KnowledgeBase): The knowledge base
        prompt (str): The question to answer
        input2 (str): Optional secondary input (required by function mapping)

    Returns:
        str: The answer to the question
    """
```

```
    print(f"Answering general knowledge question: {prompt}")

    # Store the question in the knowledge base
    kb.set_item("last_question", prompt)

    # Skip empty prompts
    if not prompt or prompt.strip() == "":
        return "It seems you didn't provide a question. How can I help you?"

    # Check if the question is about previous session content
    prompt_lower = prompt.lower()

    # Get session history
    history = kb.get_item("session_history")
    if not history:
        print("No session history found in KB!")
        return "I don't have any session history recorded yet. This might be our first
interaction."

    # Enhanced keywords for context detection
    past_indicators = ["did", "was", "asked", "last", "previous", "earlier", "before",
"history", "recall"]
    question_keywords = ["question", "query", "prompt", "ask"]
    math_keywords = ["math", "calculation", "equation", "+", "-", "*", "/", "plus",
"minus", "multiply", "divide"]
    model_keywords = ["model", "energy", "electricity", "solar", "wind", "country",
"location", "build"]

    # COMPLETELY REDESIGNED DETECTION LOGIC
    is_history_query = False
    query_type = None

    # Step 1: Check if it contains past tense indicators
    contains_past = any(word in prompt_lower for word in past_indicators)

    # Step 2: If it has past indicators, determine what type of history
    if contains_past:
        print(f"DETECTED: Contains past tense indicators")

        # Step 3: See what specific topic they're asking about
        if any(word in prompt_lower for word in model_keywords):
            is_history_query = True
            query_type = "model"
            print(f"DETECTED: Query about past models")
        elif any(word in prompt_lower for word in math_keywords):
            is_history_query = True
            query_type = "math"
            print(f"DETECTED: History query about math")
        elif any(word in prompt_lower for word in question_keywords):
            is_history_query = True
            query_type = "general"
            print(f"DETECTED: Query about past questions")
```

```
        else:
            # If no specific category, it could still be a general history query
            is_history_query = True
            query_type = "general"
            print(f"DETECTED: General query about history")

    print(f"HISTORY CHECK: is_history_query={is_history_query}, query_type=
{query_type}")

    if is_history_query:
        print(f"Processing as a history query about: {query_type}")

        try:
            # Handle based on query type
            if query_type == "model" and history.get("energy_models"):
                print(f"Found {len(history['energy_models'])} energy model entries")

                # Get the most recent model
                latest_model = history["energy_models"][-1]

                # Extract model details
                model_prompt = latest_model['prompt']
                model_result = latest_model['result']

                # Check if the question is asking about a country
                country_query = any(word in prompt_lower for word in ["country",
"location", "where", "place"])

                # If specifically asking about country
                if country_query:
                    # Try to extract country from stored data or prompt
                    country = extract_country_from_model_data(latest_model,
model_prompt)

                    if country:
                        response = f"You asked for an energy model for {country}."
                    else:
                        response = "I couldn't determine which country you asked for
in your energy model request."
                else:
                    # Create detailed response about the model
                    response = f"In a previous session, you asked me to build an
energy model. "
                    response += f"The request was: '{model_prompt}'"

                # Store response
                kb.set_item("general_answer", response)
                kb.set_item("final_report", response)

                print(f"Returning model history response")
                return response
```

```
            elif query_type == "math" and history.get("math_questions"):
                print(f"Found {len(history['math_questions'])} math question entries")

                # Get the most recent math question
                latest_math = history["math_questions"][-1]

                response = (
                    f"In a previous session, you asked the math question:
'{latest_math['prompt']}'\n\n"
                    f"The result was: {latest_math['result']}"
                )

                # Store response
                kb.set_item("general_answer", response)
                kb.set_item("final_report", response)

                print(f"Returning math history response")
                return response

            # Fallback to general session summary
            elif history.get("sessions"):
                print(f"Providing general session summary")

                # Get the previous session
                prev_session = history["sessions"][-2] if len(history["sessions"]) > 1
else history["sessions"][-1]

                # Create a summary
                response = f"In session {prev_session['id']}, you asked about:\n\n"

                for prompt_item in prev_session["prompts"]:
                    if prompt_item.strip():  # Skip empty prompts
                        response += f"- {prompt_item}\n"

                # Store response
                kb.set_item("general_answer", response)
                kb.set_item("final_report", response)

                print(f"Returning general history response")
                return response

            else:
                print(f"No relevant history found")
                return "I don't have any relevant history about that topic."

        except Exception as e:
            error_msg = f"Error processing history query: {str(e)}"
            print(f"ERROR: {error_msg}")

            # Use the LLM to create a helpful fallback response
            context = (
                "You are Nova, a helpful AI assistant. The user is asking about
```

```
previous interactions. "
                "Explain that you're having trouble retrieving that specific
information accurately, "
                "and offer to help with their current needs instead."
            )

            result = run_open_ai_ns(prompt, context)

            # Store the result
            kb.set_item("general_answer", result)
            kb.set_item("final_report", result)

            return result

    # Standard processing for non-session-related questions
    print("Not a history-related query, using standard LLM processing")
    context = (
        "You are Nova, a helpful AI assistant. Answer the following question
accurately and concisely. "
        "If it's a factual question, provide the most up-to-date information you have.
"
        "If you're unsure about the answer, acknowledge your limitations."
    )

    try:
        print(f"Sending request to LLM...")
        result = run_open_ai_ns(prompt, context)
        print(f"Received response from LLM")

        # Create a formatted answer that includes the question for clarity
        formatted_answer = f"Question: {prompt}\nAnswer: {result}"

        # Store both the raw result and the formatted answer in the knowledge base
        kb.set_item("general_answer", result)
        kb.set_item("formatted_general_answer", formatted_answer)

        # IMPORTANT: Also store in final_report to ensure it appears in output
        kb.set_item("final_report", result)

        return result
    except Exception as e:
        error_msg = f"Error processing general knowledge question: {str(e)}"
        print(f"ERROR: {error_msg}")

        # Store error in knowledge base
        kb.set_item("general_error", error_msg)

        # Return a user-friendly message
        return f"I'm sorry, I encountered an error while trying to answer your
question: {str(e)}"
```

Helper Function

```python
def extract_country_from_model_data(model_data, prompt_text):
    """
    Extract country information from model data or prompt text.

    Parameters:
        model_data (dict): The stored model data
        prompt_text (str): The prompt text from which to extract country

    Returns:
        str or None: Extracted country name or None if not found
    """
    # Check if country is directly stored in model data
    if model_data.get("country"):
        return model_data["country"]

    # Check model result for common patterns
    if isinstance(model_data.get("result"), str):
        result_text = model_data["result"]

        # Pattern: "Created X model for {country}"
        country_match = re.search(r"for\s+([A-Za-z]+)(?:\.|\s|$)", result_text)
        if country_match:
            return country_match.group(1)

    # Extract from prompt text
    prompt_lower = prompt_text.lower()
    for location in LOCATIONS:
        if location.lower() in prompt_lower:
            return location

    # Look for "for X" pattern in prompt
    for_match = re.search(r"for\s+([A-Za-z]+)(?:\.|\s|$)", prompt_lower)
    if for_match:
        country_candidate = for_match.group(1).capitalize()
        if country_candidate in LOCATIONS:
            return country_candidate

    return None
```

Key Features

1. **History Query Detection**: Sophisticated detection of queries about past interactions
2. **Topic Classification**: Categorizes history queries by topic (math, models, general)
3. **Contextual Responses**: Retrieves relevant historical information for context-aware answers
4. **LLM Integration**: Uses LLM for answering general knowledge questions
5. **Country Extraction**: Specialized handling for questions about country-specific models
6. **Error Handling**: Graceful handling of missing history data and processing errors
7. **Response Formatting**: Structures responses for clarity and consistency

Integration

- Used by Nova to handle general knowledge questions and history queries
- Accesses session history from the knowledge base
- Leverages LLM capabilities for answering factual questions
- Stores responses in the knowledge base for future reference
- Coordinates with the history tracking system

Workflow

1. Receive a question from the user
2. Store the question in the knowledge base
3. Check if the question is about past interactions:
    - Look for past tense indicators and question keywords
    - Determine the type of history query (math, models, general)
4. For history queries:
    - Retrieve relevant information from session history
    - Format an appropriate response based on the query type
    - Return the historical context to the user
5. For general knowledge questions:
    - Use the LLM to generate an informative response
    - Format the answer for clarity
6. Store the final answer in the knowledge base
7. Return the response to the user

Implementation Notes

- Uses keyword matching and pattern recognition for query classification
- Implements specialized handling for different types of history queries
- Provides fallback mechanisms when history data is incomplete
- Uses the LLM for creating contextual responses when exact history is unavailable
- Structures responses to include the original question for context
- Includes detailed logging for debugging and monitoring