Nova AI Coordinator: Session History Retrieval Enhancements

Background

The Nova AI Coordinator system uses a multi-agent architecture to process user requests:

- **Nova**: Central coordinator that routes requests
- **Emil**: Energy modeling specialist
- **Ivan**: Technical specialist for code/image generation
- **Lola**: Content and report specialist

One key feature is the ability to recall previous conversation history, which we've significantly enhanced.

Initial Issues

1. **Missing Method Error**:
   Error in `utils/general_knowledge.py`:
   `'KnowledgeBase' object has no attribute 'get_session_details'`
2. **Inaccurate Content Generation**:
   LLM was "enhancing" factual session data with fabricated content in `answer_general_question` function
3. **Rigid Query Detection**:
   System could only recognize specific phrasings via regex in `answer_general_question` (e.g., "session 4" but not "sessions 4")
4. **Limited Retrieval Options**:
   Only session number-based retrieval was supported in `answer_general_question`

Solution 1: Fix Session Data Retrieval

Key Improvements in `utils/general_knowledge.py`:

1. Bypass LLM "enhancement" for factual session data
2. Add flags to skip evaluation system for history queries
3. Use direct data formatting for accuracy

Example Before:

```
[Quality Evaluation: Initial answer scored 0.4/1.0. Used more_detailed_llm to improve
answer.]

In session 2, the topics discussed included an overview of the historical context of the
subject...
```

Actual Session 2 Content:

```json
{
  "id": 2,
  "timestamp": "2025-04-29T10:21:14.722295",
  "prompts": ["what is the weather for spain today"],
  "results": ["The weather for Spain today is not provided in the conversation history."]
}
```

Solution 2: Dynamic Query Detection

In `utils/general_knowledge.py`, function `answer_general_question()`:

```python
# File: utils/general_knowledge.py
# Function: answer_general_question()

# Step 1: Use LLM to detect if this is a history query and extract identifiers
history_detection_context = """
You are analyzing a query to determine if it's asking about past conversations or session histo
If it is a history query, identify what specific session or conversation the user is asking abo

Return a JSON object with this structure:
{
    "is_history_query": true/false,
    "session_id": null or number,
    "reference_type": "session" or "date" or "topic" or null,
    "confidence": 0.0-1.0
}
"""

# Use LLM to analyze if this is a history query
history_analysis_json = run_open_ai_ns(prompt, history_detection_context, model="gpt-4.1-nano")
```

This allows the system to understand queries like:

- "What was in session 4?"
- "Tell me about sessions 4"
- "What did we discuss in the fourth session?"

Solution 3: Date-Based Retrieval

In `utils/general_knowledge.py`, function `answer_general_question()`, inside the date reference handler:

```python
# File: utils/general_knowledge.py
# Function: answer_general_question()
# Section: reference_type == "date" handler

# Extract and parse the date
from dateutil import parser as date_parser
parsed_date = date_parser.parse(date_text, fuzzy=True)
target_date = parsed_date.strftime("%Y-%m-%d")

# Find sessions on the target date
matching_sessions = []
for session in sessions:
    if "timestamp" in session:
        session_time = datetime.datetime.fromisoformat(session["timestamp"])
        session_date = session_time.strftime("%Y-%m-%d")

        # Compare dates
        if session_date == target_date:
            matching_sessions.append(session)
```

Handles natural language queries like:

- "What was discussed on May 2, 2025?"
- "Show me the sessions from May 2"
- "What did we talk about on 5/2/25?"

Solution 4: Enhanced Time Display

In `utils/general_knowledge.py`, function `answer_general_question()`, in both session and date handlers:

```python
# File: utils/general_knowledge.py
# Function: answer_general_question()
# For both session-specific and date-based queries:

# Extract and format time
session_time_str = "unknown time"
if "timestamp" in session:
    session_datetime = datetime.datetime.fromisoformat(session["timestamp"])
    session_time_str = session_datetime.strftime("%I:%M %p")  # Format as "03:45 PM"

# Include time in output
formatted_response += f"Session {session_id} (at {session_time_str}):\n"
```

Example output:

```
Session 14 (at 12:06 PM):
  Question 1: What is the capital of UK?
  Answer 1: {'status': 'success', 'message': 'Created electricity Electricity model for
All'...
```

Enhanced Query Flow

1. **Input Detection**:
   `answer_general_question()` in `utils/general_knowledge.py` uses LLM to analyze query type
2. **Data Retrieval**:
   Session data retrieved via `kb.get_item("session_history")` based on reference type
3. **Direct Formatting**:
   Session data formatted without LLM "enhancement" in the same function
4. **Evaluation Bypassing**:
   `DIRECT_SESSION_DATA` prefix and flags in `utils/evaluation.py` ensure factual reporting

```
User Query → Nova → answer_general_question() → LLM Detection → Data Retrieval → Direct
Formatting → Response
```

Function Mapping

The implementation integrates with Nova's existing function mapping system:

```
# File: src/agents/Nova_function_map_enhanced.csv
Key: general_question
Function: utils.general_knowledge.answer_general_question
```

1. User asks about session history
2. Nova routes to `answer_general_question` function in `utils/general_knowledge.py`
3. Function detects history query and retrieves session data
4. Response bypasses evaluation pipeline with special flags in `utils/evaluation.py`

This maintains compatibility with the existing function registry while adding new capabilities.

Evaluation Bypassing

In `utils/evaluation.py`, function `evaluate_answer_quality()`:

```python
# File: utils/evaluation.py
# Function: evaluate_answer_quality()

# Check if this is a direct session data response that should skip evaluation
if answer and isinstance(answer, str) and answer.startswith("DIRECT_SESSION_DATA:"):
    print("🔍 Skipping evaluation for direct session data")
    # Remove the marker prefix before returning to user
    clean_answer = answer.replace("DIRECT_SESSION_DATA: ", "")

    # Store the clean answer back in KB
    await kb.set_item_async("general_answer", clean_answer)
    await kb.set_item_async("final_report", clean_answer)

    # Return perfect evaluation
    return {
        "score": 1.0,
        "strengths": ["Accurate session data reporting", "Direct information retrieval", "Factu
        "weaknesses": [],
        "improvement_suggestions": [],
        "passed": True
    }
```

Results

Date-Based Retrieval Example:

```
DIRECT_SESSION_DATA: On May 02, 2025, I found 4 session(s):

Session 14 (at 12:06 PM):
  Question 1: What is the capital of UK?
  Answer 1: {'status': 'success', 'message': 'Created electricity Electricity model for
All'...

Session 15 (at 12:08 PM):
  Question 1: What is the capital of UK?
  ...
```

Session-Based Retrieval Example:

```
DIRECT_SESSION_DATA: In session 2 (from 2025-04-29T10:21:14.722295, at 10:21 AM), the
following was discussed:

Question: what is the weather for spain today
Answer: The weather for Spain today is not provided in the conversation history.
```

Future Extensions

The new architecture supports additional retrieval methods:

1. **Topic-Based Retrieval**:
   Find all sessions discussing a particular topic - extend the `reference_type == "topic"` handler in `answer_general_question()`
2. **Entity-Based Retrieval**:
   Find sessions where specific entities (countries, people) were discussed - add a new reference type in the LLM detection system
3. **Content-Based Search**:
   Search for specific terms across all session history - add a new reference type and handler

These extensions can be added by implementing the appropriate sections in the reference_type handlers.

Conclusions

✅ **Fixed Data Accuracy**: Session history now shows actual content, not fabricated information

✅ **Enhanced Flexibility**: System handles various natural language query formats

✅ **Added Date Retrieval**: Users can now retrieve sessions by date

✅ **Improved Context**: Time information provides better context for session history

✅ **Future-Proof Design**: Architecture supports additional retrieval methods

---

Dependencies

- `python-dateutil` package for natural language date parsing
- Requires appropriate datetime handling in session storage

Implementation Notes

- Primary changes are in:
  - `utils/general_knowledge.py`: function `answer_general_question()`
  - `utils/evaluation.py`: function `evaluate_answer_quality()`
- No changes were required to:
  - Knowledge base structure (`core/knowledge_base.py`)
  - Function mapping system
  - Agent initialization