Nova AI Coordinator: Detailed Evaluation System Flow with Function Descriptions & Code Snippets

1. User Input → Task Processing → Answer Generation

1. **User submits a question** to the Nova AI Coordinator
   - Input processed in `main.py` (entry point)

2. **Task creation and processing**
   - `Nova.create_task_list_from_prompt_async()` in `nova.py`
     - *Analyzes user prompt to identify intents and creates appropriate tasks*
     - *Converts high-level user requests into structured Task objects*
     - *Handles multi-intent detection and task categorization*
     - **Key code**:

       python

       ```
       multiple_intents = await self.identify_multiple_intents_async(prompt)
       for intent_info in multiple_intents:
           intent_text = intent_info["intent"]
           category = await open_ai_categorisation_async(intent_text, csv_path)
           task = await self.create_task_for_category(intent_text, category)
           tasks.append(task)
       ```

3. **Task handling and answer generation**
   - `Nova.handle_task_async()` in `nova.py`
     - *Main orchestration function that processes tasks and delegates to appropriate agents*
     - *Manages the full lifecycle of a task from assignment to result delivery*
     - *Contains the evaluation and fallback logic for answer improvement*
     - **Key code**:

       python

       ```
       # Use the specific agent implementation based on agent name
       if task.agent == "Emil":
           result = await agents["Emil"].handle_task_async(task)
       elif task.agent == "Lola":
           result = await agents["Lola"].handle_task_async(task)
       elif task.agent == "Ivan":
           result = await agents["Ivan"].handle_task_async(task)
       else:
           result = await agents["Nova"].handle_task_async(task)
       ```

- For general knowledge: `answer_general_question()` in `general_knowledge.py`
  - *Generates answers to general knowledge questions using LLM*
  - *Maintains conversation context for coherent multi-turn interactions*
  - *Tracks entities mentioned in questions and answers*
  - **Key code**:

```python
# Create LLM prompt with enhanced context
if len(conversation.get("answers", [])) > 0:
    context_prompt = """
    You are continuing a conversation. I will provide the conversation history and a
    ANY pronouns (it, its, they, etc.) or phrases like "this country" MUST refer to
    """
```

- For math: `do_maths()` in `utils/do_maths.py`
  - *Handles mathematical calculations with deterministic results*
  - *Bypasses evaluation since math answers can be verified programmatically*
  - *Returns formatted calculation results*
  - **Key code**:

```python
# Solve the math expression
result = eval(cleaned_expression)
return f"The result of {cleaned_expression} is {result}"
```

2. Evaluation System Initialization and Answer Quality Check

4. **Evaluation system initialization**

- `initialize_evaluation_system()` in `evaluation.py`
  - *Sets up the evaluation configuration with default or custom parameters*
  - *Registers evaluation settings in the knowledge base for persistence*
  - *Configures critical parameters like quality threshold and model selection*
  - **Key code**:

```python
# Default configuration
default_config = {
    "evaluation_enabled": True,
    "quality_threshold": 0.7,
    "use_internet_search": True,
    "evaluation_model": "gpt-4.1-nano",
    "fallback_evaluation_model": "gpt-3.5-turbo",
    "max_retries": 2,
    "debug_output": True
}

# Merge with provided config
active_config = default_config.copy()
if config:
    active_config.update(config)
```

- Called during startup in `main.py`
- Sets up configuration parameters

5. **Answer evaluation triggered**
   - Inside `Nova.handle_task_async()` in `nova.py`
     - *Checks whether the task requires evaluation (primarily for general questions)*
     - *Extracts original question and generated answer for quality assessment*
     - *Calls the evaluation function to assess answer quality*
     - **Key code**:

```python
# For general knowledge questions, evaluate the answer quality and use fallbacks if
if task.function_name == "answer_general_question":
    # Get the question from task args
    question = task.args.get("prompt", "")

    # Get evaluation config
    config = self.kb.get_item("evaluation_config") or {}
    if config.get("evaluation_enabled", True):
        try:
            # Evaluate answer quality
            evaluation = await evaluate_answer_quality(self.kb, question, result)
```

   - Specifically for general questions

6. **Evaluation prompt construction and LLM call**

- Inside `evaluate_answer_quality()` in `evaluation.py`
  - *Constructs a prompt for the LLM to evaluate answer quality*
  - *Sends the evaluation request to the LLM API*
  - *Handles response parsing and validation*
  - **Key code**:

```python
# Simplify the evaluation prompt for better reliability
evaluation_prompt = f"""
Evaluate the quality of this answer to the given question.

Question: "{question}"
Answer: "{answer}"

Rate from 0.0 to 1.0 where 1.0 is perfect.
List 1-3 strengths and 1-3 weaknesses.
Make 1-2 improvement suggestions.

RETURN ONLY VALID JSON with this structure:
{{
    "score": 0.X,
    "strengths": ["strength1", "strength2"],
    "weaknesses": ["weakness1", "weakness2"],
    "improvement_suggestions": ["suggestion1"],
    "passed": true/false
}}
"""
```

- Calls `run_open_ai_ns_async()` in `utils/open_ai_utils.py`
  - *Asynchronous wrapper for OpenAI API calls*
  - *Handles authentication, request formatting, and response processing*
  - *Manages timeouts and retry logic for API communication*
  - **Key code**:

```python
async def run_open_ai_ns_async(prompt, system_message, model="gpt-4.1-nano", tempera
    try:
        response = await client.chat.completions.create(
            model=model,
            messages=[
                {"role": "system", "content": system_message},
                {"role": "user", "content": prompt}
            ],
            temperature=temperature
        )
        return response.choices[0].message.content
    except Exception as e:
        print(f"Error calling OpenAI: {str(e)}")
        raise
```

- Uses model specified in config (typically "gpt-4.1-nano")

3. LLM Response Handling and Retry Logic

7. **JSON parsing and validation**
   - Still in `evaluate_answer_quality()` in `evaluation.py`
     - *Attempts to parse the LLM response as valid JSON*
     - *Implements retry logic to handle transient failures*
     - *Validates that required fields are present in the response*
     - **Key code**:

```python
# Try to parse JSON response
try:
    eval_result = json.loads(eval_result_json)

    # Validate required fields
    if "score" not in eval_result:
        raise ValueError("Missing 'score' field")

    # Set passed field correctly
    eval_result["passed"] = eval_result.get("score", 0.0) >= threshold

    # Success - exit retry loop
    break
```

   - Tries to parse LLM response as JSON

- Uses retry logic

8. **Regex extraction fallback**
   - If JSON parsing fails, calls `extract_evaluation_data()` in `evaluation.py`
     - *Uses regex patterns to extract structured data from unstructured text*
     - *Attempts to recover key fields like score, strengths, and weaknesses*
     - *Provides a fallback mechanism when JSON parsing fails*
     - **Key code**:

```python
# Try to extract score with multiple patterns
score_patterns = [
    r'"score":\s*(0\.\d+|1\.0|1|0)',
    r'score.*?(\d+\.?\d*)\s*\/\s*1',
    r'(\d+\.?\d*)\s*\/\s*1',
    r'rated\s+(\d+\.?\d*)',
    r'score\s+of\s+(\d+\.?\d*)'
]

for pattern in score_patterns:
    score_match = re.search(pattern, text, re.IGNORECASE)
    if score_match:
        try:
            eval_data["score"] = float(score_match.group(1))
            eval_data["passed"] = eval_data["score"] >= threshold
            break
        except:
            continue
```

   - Uses regex patterns to extract key elements

9. **Fallback model attempt**
   - Still in `evaluate_answer_quality()` in `evaluation.py`
     - *Attempts to use a different LLM model if the primary model fails*
     - *Provides redundancy to increase overall system reliability*
     - *May use a simpler model with different parameters for better JSON parsing*
     - **Key code**:

```python
# If primary model failed, try fallback model
if eval_result is None and fallback_model and fallback_model != model:
    try:
        if debug_output:
            print(f"🔍 Trying fallback model: {fallback_model}")

        eval_result_json = await run_open_ai_ns_async(
            evaluation_prompt,
            eval_context,
            model=fallback_model,
            temperature=0.3  # Slightly higher temperature for variety
        )
```

- If primary model failed

10. **Default evaluation fallback**
    - Last resort in `evaluate_answer_quality()` in `evaluation.py`
        - *Provides a guaranteed minimum evaluation response when all else fails*
        - *Includes error information to help with debugging*
        - *Ensures the system can continue operating despite failures*
        - **Key code**:

```python
# If all attempts failed, use a more informative fallback
if eval_result is None:
    eval_result = {
        "score": 0.5,
        "strengths": ["Answer contained some information"],
        "weaknesses": [f"Evaluation failed: {last_error[:50]}..."],
        "improvement_suggestions": ["Provide more specific information"],
        "passed": False,
        "error": last_error
    }
```

- If all attempts failed

4. Evaluation Results Storage and Decision Making

11. **Store evaluation results in knowledge base**
    - End of `evaluate_answer_quality()` in `evaluation.py`
        - *Persists evaluation results in the knowledge base for later reference*
        - *Maintains evaluation history for session tracking*

- *Enables analysis of evaluation patterns over time*
- **Key code**:

```python
# Store evaluation in KB
await kb.set_item_async("last_evaluation_result", eval_result)
await kb.set_item_async("last_evaluation_time", datetime.datetime.now().isoformat())

# Append to evaluation history
eval_history = await kb.get_item_async("evaluation_history") or []
eval_history.append({
    "question": question,
    "answer": answer,
    "evaluation": eval_result,
    "timestamp": datetime.datetime.now().isoformat()
})
await kb.set_item_async("evaluation_history", eval_history)
```

12. **Update conversation context**
    - Still in `evaluate_answer_quality()` in `evaluation.py`
      - *Updates the current conversation with evaluation metadata*
      - *Maintains continuity across multiple interactions*
      - *Provides context for follow-up questions and answers*
      - **Key code**:

```python
# Add to conversation summary
conversation = kb.get_item("current_conversation") or {}
if "evaluations" not in conversation:
    conversation["evaluations"] = []

conversation["evaluations"].append({
    "question": question,
    "score": eval_result.get("score", 0.0),
    "passed": eval_result.get("passed", False),
    "strengths": eval_result.get("strengths", []),
    "weaknesses": eval_result.get("weaknesses", [])
})
kb.set_item("current_conversation", conversation)
```

13. **Threshold check and fallback decision**
    - Back in `Nova.handle_task_async()` in `nova.py`

- *Checks if the evaluation score passes the quality threshold*
- *Decides whether to use the original answer or seek improvement*
- *Triggers fallback strategy selection if improvement is needed*
- **Key code**:

```python
# Check if evaluation passed threshold
if not evaluation["passed"]:
    print(f"Answer quality below threshold ({evaluation['score']}). Attempting fallb

    # Determine the best fallback strategy
    available_alternatives = ["internet_search", "more_detailed_llm", "database_look
    strategy = await determine_fallback_strategy(self.kb, question, evaluation, avai
```

5. Fallback Strategy Selection and Execution

14. **Determine fallback strategy**
    - `determine_fallback_strategy()` in `evaluation.py`
      - *Uses LLM to intelligently select the best fallback strategy*
      - *Analyzes question type and evaluation feedback*
      - *Returns a recommended strategy with justification*
      - **Key code**:

```python
# Create strategy selection prompt
strategy_prompt = f"""
Determine the best fallback strategy for improving the following answer to a questic
The answer has been evaluated and needs improvement.

Question: "{question}"

Evaluation:
- Score: {evaluation.get('score', 'N/A')}
- Strengths: {', '.join(evaluation.get('strengths', ['None']))}
- Weaknesses: {', '.join(evaluation.get('weaknesses', ['None']))}

Available strategies:
{', '.join(available_alternatives)}

Return your recommendation in JSON format:
{{
    "recommended_strategy": "strategy_name",
    "reason": "brief explanation for this choice"
}}
"""
```

- Uses LLM to determine best strategy
- Calls `run_open_ai_ns_async()` in `utils/open_ai_utils.py`
- Returns strategy recommendation

15. **Execute fallback strategy**
    - Back in `Nova.handle_task_async()` in `nova.py`
        - *Implements the selected fallback strategy*
        - *Routes to different improvement paths based on strategy*
        - *Calls specialized functions for each strategy type*
        - **Key code**:

```python
# Execute the recommended fallback strategy
if strategy["recommended_strategy"] == "internet_search":
    print("Using internet search fallback...")
    from utils.internet_search import internet_search
    search_results = await internet_search(self.kb, question)

    # Generate improved answer using search results
    improved_result = await generate_improved_answer(
        self.kb, question, result, evaluation, search_results
    )
```

- Different code paths based on strategy
- `internet_search()` in `utils/internet_search.py`
  - *Simulates an internet search with LLM-generated results*
  - *Formats search results with titles, snippets, and sources*
  - *Returns structured search data for answer improvement*
  - **Key code**:

```python
# In a production environment, this would call a real search API
# For now, simulate with an LLM
search_prompt = f"""
Simulate internet search results for the query: "{query}"

Provide results in the following JSON format:
{{
    "search_results": [
        {{
            "title": "Result Title 1",
            "url": "https://example.com/page1",
            "snippet": "Brief excerpt from the page showing relevant content...",
            "source": "example.com"
        }},
        // Additional results...
    ],
    "featured_snippet": "A direct answer to the query if available"
}}
"""
```

16. **Generate improved answer**
    - `generate_improved_answer()` in `evaluation.py`
      - *Creates a better answer based on evaluation feedback*

- *Incorporates search results or other additional information*
- *Uses LLM to synthesize an improved response*
- **Key code**:

```python
# Create improvement prompt
improvement_prompt = f"""
Improve the following answer to a question based on evaluation feedback
and additional information (if provided).

Question: "{question}"

Original Answer: "{original_answer}"

Evaluation:
- Score: {evaluation.get('score', 'N/A')}
- Strengths: {', '.join(evaluation.get('strengths', ['None']))}
- Weaknesses: {', '.join(evaluation.get('weaknesses', ['None']))}
- Improvement Suggestions: {', '.join(evaluation.get('improvement_suggestions', ['N
"""

# Add search results if available
if search_results and search_results.get("search_results"):
    improvement_prompt += "\n\nAdditional Information from Search:\n"
    # Add featured snippet if available
    if search_results.get("featured_snippet"):
        improvement_prompt += f"Featured Answer: {search_results['featured_snippet']
```

- Creates an improvement prompt
- Adds search results if available
- Calls `run_open_ai_ns_async()` in `utils/open_ai_utils.py`
- Returns improved answer

6. Final Answer Processing and Delivery

17. **Store improved answer**
    - Back in `Nova.handle_task_async()` in `nova.py`
      - *Updates the task result with the improved answer*
      - *Records that a fallback strategy was used*
      - *Maintains metadata about which strategy was applied*
      - **Key code**:

```python
# Update the result
task.result = improved_result
result = improved_result

# Note that we used a fallback
await kb.set_item_async("used_fallback", True)
await kb.set_item_async("fallback_method", strategy["recommended_strategy"])
```

18. **Add improved answer to conversation**
    - Still in `Nova.handle_task_async()` in `nova.py`
      - *Updates conversation history with the improved answer*
      - *Maintains record of original and improved answers*
      - *Preserves evaluation score for comparison*
      - **Key code**:

```python
# Store the improved answer in conversation
if "improved_answers" not in conversation:
    conversation["improved_answers"] = []

conversation["improved_answers"].append({
    "question": question,
    "original_answer": result,
    "improved_answer": task.result,
    "original_score": evaluation.get("score", 0.0)
})
kb.set_item("current_conversation", conversation)
```

19. **Final result returned to user**
    - End of `Nova.handle_task_async()` in `nova.py`
      - *Returns either original or improved answer to the caller*
      - *Completes the task handling process*
      - *Delivers the best available answer to the user*
      - **Key code**:

```python
# Log the result
kb.log_interaction(prompt, f"Task {i} completed with result",
                   agent="System", function="process_prompt_tasks")

return result
```

- Either returns original answer (if passed evaluation) or improved answer (if fallback was used)
- Result is then processed in `process_prompt_tasks()` in `main.py`
  - *Formats results for display to the user*
  - *Combines results from multiple tasks if needed*
  - *Generates the final response seen by the user*
  - **Key code**:

```python
# Combine results
if not results:
    return "No results found for any tasks."
elif len(results) == 1:
    return results[0]
else:
    combined = "\n\n" + "-"*40 + "\n\n".join([str(result) for result in results]) +
    return combined
```

- Displayed to user in the conversation summary

Visual Function Call Map

```
main.py
  │  # Entry point and overall orchestration
  │  # Key code: interactive_async_main()
  ▼
Nova.create_task_list_from_prompt_async() [nova.py]
  │  # Converts user prompt to structured tasks
  │  # Key code: multiple_intents = await self.identify_multiple_intents_async(prompt)
  ▼
Nova.handle_task_async() [nova.py]
  │  # Main task processing and orchestration
  │  # Key code: result = await agents["Nova"].handle_task_async(task)
  │
  ├─► answer_general_question() [general_knowledge.py] or other task handler
  │    │  # Generates initial answer using LLM
  │    │  # Key code: result = run_open_ai_ns(context_prompt, "You must maintain
conversation context...")
  │    ▼
  │    run_open_ai_ns() [utils/open_ai_utils.py] - Generates initial answer
  │      # Key code: response = client.chat.completions.create(model=model, messages=[...])
  │
  ▼
evaluate_answer_quality() [evaluation.py]
  │  # Assesses answer quality using LLM
  │  # Key code: evaluation_prompt = f"Evaluate the quality of this answer to the given
question..."
  │
  ├─► run_open_ai_ns_async() [utils/open_ai_utils.py] - Primary LLM evaluation
  │    # Makes asynchronous call to OpenAI API
  │    # Key code: response = await client.chat.completions.create(model=model, messages=
[...])
  │
  ├─► extract_evaluation_data() [evaluation.py] - If JSON parsing fails
  │    # Uses regex to extract structured data from text
  │    # Key code: score_match = re.search(pattern, text, re.IGNORECASE)
  │
  ├─► run_open_ai_ns_async() [utils/open_ai_utils.py] - Fallback model if needed
  │    # Uses alternative model for evaluation
  │    # Key code: eval_result_json = await run_open_ai_ns_async(..., model=fallback_model)
  │
  ▼
If evaluation["passed"] == False:
  │  # Answer quality below threshold
  │  # Key code: if not evaluation["passed"]: print(f"Answer quality below threshold...")
  ▼
determine_fallback_strategy() [evaluation.py]
  │  # Selects best strategy to improve answer
```

```
  │  # Key code: strategy_prompt = f"Determine the best fallback strategy..."
  │
  ├─► run_open_ai_ns_async() [utils/open_ai_utils.py] - Strategy recommendation
  │    # Uses LLM to recommend improvement strategy
  │    # Key code: strategy_json = await run_open_ai_ns_async(strategy_prompt,
strategy_context)
  │
  ▼
Switch based on recommended strategy:
  │  # Routes to appropriate improvement method
  │  # Key code: if strategy["recommended_strategy"] == "internet_search": ...
  │
  ├─► internet_search() [utils/internet_search.py] - If search strategy
  │    │  # Simulates internet search for more information
  │    │  # Key code: search_response = await run_open_ai_ns_async(search_prompt,
search_context)
  │    ▼
  │    generate_improved_answer() [evaluation.py] - With search results
  │      # Creates better answer using search results
  │      # Key code: improvement_prompt += "\n\nAdditional Information from Search:\n"
  │
  ├─► generate_improved_answer() [evaluation.py] - If other strategies
  │    │  # Creates better answer using evaluation feedback
  │    │  # Key code: improvement_prompt = f"Improve the following answer..."
  │    ▼
  │    run_open_ai_ns_async() [utils/open_ai_utils.py] - Generate better answer
  │      # Uses LLM to create improved response
  │      # Key code: improved_answer = await run_open_ai_ns_async(improvement_prompt,
improvement_context)
  │
  ▼
Final answer returned to user via process_prompt_tasks() [main.py]
  # Formats and delivers final answer to user
  # Key code: combined = "\n\n" + "-"*40 + "\n\n".join([str(result) for result in results])
```

Evaluation System Improvements Summary

1. **Retry Mechanism**
   - Added multiple attempts before falling back to default
   - Configurable `max_retries` parameter
   - Wait periods between retries
   - **Key code**:

```python
retry_count = 0
while retry_count < max_retries:
    try:
        # LLM call and JSON parsing
        # Success - break from retry loop
    except Exception as e:
        retry_count += 1
        if retry_count < max_retries:
            await asyncio.sleep(1)  # Wait before retry
```

2. **Fallback Model**
   - Alternative LLM when primary model fails
   - Configurable `fallback_evaluation_model` parameter
   - Different temperature setting for variety
   - **Key code**:

```python
if eval_result is None and fallback_model and fallback_model != model:
    try:
        eval_result_json = await run_open_ai_ns_async(
            evaluation_prompt,
            eval_context,
            model=fallback_model,
            temperature=0.3
        )
```

3. **Simplified Evaluation Prompt**
   - Shorter, clearer instructions
   - Focused on essential requirements
   - Better JSON format guidance
   - **Key code**:

```python
evaluation_prompt = f"""
Evaluate the quality of this answer to the given question.

Question: "{question}"
Answer: "{answer}"

Rate from 0.0 to 1.0 where 1.0 is perfect.
List 1-3 strengths and 1-3 weaknesses.
Make 1-2 improvement suggestions.

RETURN ONLY VALID JSON with this structure:
{{
    "score": 0.X,
    "strengths": ["strength1", "strength2"],
    "weaknesses": ["weakness1", "weakness2"],
    "improvement_suggestions": ["suggestion1"],
    "passed": true/false
}}
"""
```

4. **Enhanced JSON Parsing**
   - More robust extraction function
   - Multiple regex patterns for different response formats
   - Validation of extracted data
   - **Key code**:

```python
# Try to extract score with multiple patterns
score_patterns = [
    r'"score":\s*(0\.\d+|1\.0|1|0)',
    r'score.*?(\d+\.?\d*)\s*\/\s*1',
    r'(\d+\.?\d*)\s*\/\s*1',
    r'rated\s+(\d+\.?\d*)',
    r'score\s+of\s+(\d+\.?\d*)'
]
```

5. **Better Error Reporting**
   - More informative fallback results
   - Detailed error messages in logs
   - Preservation of original error for debugging
   - **Key code**:

python

```python
if eval_result is None:
    eval_result = {
        "score": 0.5,
        "strengths": ["Answer contained some information"],
        "weaknesses": [f"Evaluation failed: {last_error[:50]}..."],
        "improvement_suggestions": ["Provide more specific information"],
        "passed": False,
        "error": last_error
    }
```