Nova AI Coordinator: Detailed Evaluation System Flow with File and Function Mapping

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`
   - Creates tasks based on the user's prompt
3. **Task handling and answer generation**
   - `Nova.handle_task_async()` in `nova.py`
   - Processes the task and generates an initial answer
   - For general knowledge: `answer_general_question()` in `general_knowledge.py`
   - For math: `do_maths()` in `utils/do_maths.py`

2. Evaluation System Initialization and Answer Quality Check

4. **Evaluation system initialization**
   - `initialize_evaluation_system()` in `evaluation.py`
   - Called during startup in `main.py`
   - Sets up configuration parameters:

```python
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
}
```

5. **Answer evaluation triggered**
   - Inside `Nova.handle_task_async()` in `nova.py`
   - Specifically for general questions:

```python
if task.function_name == "answer_general_question":
    question = task.args.get("prompt", "")
    if config.get("evaluation_enabled", True):
        evaluation = await evaluate_answer_quality(kb, question, result)
```

6. **Evaluation prompt construction and LLM call**
   - Inside `evaluate_answer_quality()` in `evaluation.py`
   - Creates the evaluation prompt
   - Calls `run_open_ai_ns_async()` in `utils/open_ai_utils.py`
   - 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`
   - Tries to parse LLM response as JSON
   - Uses retry logic:

```python
retry_count = 0
while retry_count < max_retries:
    try:
        # LLM call
        eval_result_json = await run_open_ai_ns_async(...)

        # JSON parsing
        try:
            eval_result = json.loads(eval_result_json)
            # Validation and exit if successful
            break
        except json.JSONDecodeError:
            # Fallback to regex extraction
    except Exception as e:
        # Handle errors
        retry_count += 1
```

8. **Regex extraction fallback**
   - If JSON parsing fails, calls `extract_evaluation_data()` in `evaluation.py`
   - Uses regex patterns to extract key elements:

```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',
    # additional patterns
]
```

9. **Fallback model attempt**
   - Still in `evaluate_answer_quality()` in `evaluation.py`
   - If primary model failed:

```python
if eval_result is None and fallback_model and fallback_model != model:
    try:
        # Try with fallback model
        eval_result_json = await run_open_ai_ns_async(
            evaluation_prompt,
            eval_context,
            model=fallback_model,
            temperature=0.3
        )
        # JSON parsing code for fallback results
    except Exception as e:
        # Error handling
```

10. **Default evaluation fallback**
    - Last resort in `evaluate_answer_quality()` in `evaluation.py`
    - If all attempts failed:

```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
    }
```

4. Evaluation Results Storage and Decision Making

11. **Store evaluation results in knowledge base**

- End of `evaluate_answer_quality()` in `evaluation.py`

python

```python
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`

python

```python
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`

python

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

    # Determine the best fallback strategy
    available_alternatives = ["internet_search", "more_detailed_llm", "database_lookup"]
    strategy = await determine_fallback_strategy(kb, question, evaluation, available_altern
```

5. Fallback Strategy Selection and Execution

14. **Determine fallback strategy**
    - `determine_fallback_strategy()` in `evaluation.py`
    - Uses LLM to determine best strategy:

```python
strategy_prompt = f"""
Determine the best fallback strategy for improving the following answer to a question.
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"
}}
```

    - 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`
    - Different code paths based on strategy:

python

```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(kb, question)

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

elif strategy["recommended_strategy"] == "more_detailed_llm":
    print("Using more detailed LLM fallback...")
    improved_result = await generate_improved_answer(
        kb, question, result, evaluation
    )

else:
    # For any other strategy
    print(f"Using general improvement fallback...")
    improved_result = await generate_improved_answer(
        kb, question, result, evaluation
    )
```

16. **Generate improved answer**
    - `generate_improved_answer()` in `evaluation.py`
    - Creates an improvement prompt:

python

```python
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', ['None']))}
"""
```

- 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`

python

```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`

```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`
    - 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`
    - Displayed to user in the conversation summary

Visual Function Call Map

```
main.py
  |
  ▼
Nova.create_task_list_from_prompt_async() [nova.py]
  |
  ▼
Nova.handle_task_async() [nova.py]
  |
  ├─► answer_general_question() [general_knowledge.py] or other task handler
  |    |
  |    ▼
  |    run_open_ai_ns() [utils/open_ai_utils.py] - Generates initial answer
  |
  ▼
evaluate_answer_quality() [evaluation.py]
  |
  ├─► run_open_ai_ns_async() [utils/open_ai_utils.py] - Primary LLM evaluation
  |
  ├─► extract_evaluation_data() [evaluation.py] - If JSON parsing fails
  |
  ├─► run_open_ai_ns_async() [utils/open_ai_utils.py] - Fallback model if needed
  |
  ▼
If evaluation["passed"] == False:
  |
  ▼
determine_fallback_strategy() [evaluation.py]
  |
  ├─► run_open_ai_ns_async() [utils/open_ai_utils.py] - Strategy recommendation
  |
  ▼
Switch based on recommended strategy:
  |
  ├─► internet_search() [utils/internet_search.py] - If search strategy
  |    |
  |    ▼
  |    generate_improved_answer() [evaluation.py] - With search results
  |
  ├─► generate_improved_answer() [evaluation.py] - If other strategies
  |    |
  |    ▼
  |    run_open_ai_ns_async() [utils/open_ai_utils.py] - Generate better answer
  |
  ▼
Final answer returned to user via process_prompt_tasks() [main.py]
```

Evaluation System Improvements Summary

1. **Retry Mechanism**
   - Added multiple attempts before falling back to default
   - Configurable `max_retries` parameter
   - Wait periods between retries
2. **Fallback Model**
   - Alternative LLM when primary model fails
   - Configurable `fallback_evaluation_model` parameter
   - Different temperature setting for variety
3. **Simplified Evaluation Prompt**
   - Shorter, clearer instructions
   - Focused on essential requirements
   - Better JSON format guidance
4. **Enhanced JSON Parsing**
   - More robust extraction function
   - Multiple regex patterns for different response formats
   - Validation of extracted data
5. **Better Error Reporting**
   - More informative fallback results
   - Detailed error messages in logs
   - Preservation of original error for debugging