# Enhancing Nova with LLM-Based Query Detection

## The Problem: Hardcoded Pattern Matching

* Current implementation relies on hardcoded word lists
* Cannot adapt to natural language variations
* Requires manual updates for new patterns
* Misses context-dependent queries

python

# Current pattern matching approach

past\_indicators = ["did", "was", "asked", "had", "previous", "earlier", "last"]

question\_words = ["what", "which", "when", "where", "how", "tell me"]

is\_history\_query = (any(word in prompt\_lower for word in past\_indicators) and

any(word in prompt\_lower for word in question\_words))

## The Solution: LLM-Based Detection

* Use the LLM to understand query semantics
* Detect history queries and follow-ups based on meaning
* Preserve conversation context for better understanding
* Adapt to any phrasing or language pattern

python

# LLM-based approach

query\_type = await self.detect\_query\_type\_with\_llm(prompt, previous\_prompts)

is\_history\_query = query\_type["is\_history\_query"]

## Process Flow Map: LLM-Based Query Detection

User Input

│

▼

create\_task\_list\_from\_prompt\_async

│

├──────────────────┬─────────────────┐

│ │ │

▼ ▼ ▼

detect\_query\_type\_with\_llm identify\_multiple\_intents\_async

│ │

├──────────┐ │

│ │ │

▼ ▼ ▼

History Follow-up Multiple Intent

Query Detection Detection

│ │ │

│ └─────────────┐ │

│ │ │

└─────────────┐ │ │

▼ ▼ ▼

create\_task\_for\_category

│

▼

Task List

## Flow Breakdown

1. **User Input** → The process begins when a user submits a prompt
2. **create\_task\_list\_from\_prompt\_async**
   * Primary function that orchestrates the task creation
   * Calls LLM detection to understand query intent
   * Creates appropriate tasks based on query type
3. **detect\_query\_type\_with\_llm**
   * Uses LLM to analyze the prompt with conversation context
   * Classifies the prompt as history query, follow-up, or standalone
   * Returns structured classification results
4. **identify\_multiple\_intents\_async**
   * Called after query type determination
   * Uses LLM to split multi-intent prompts
   * Checks for follow-up queries using LLM detection
   * Returns a list of distinct intents
5. **Query Classification Branches**
   * **History Query**: Creates a task to answer questions about previous interactions
   * **Follow-up Detection**: Preserves conversation context for contextual queries
   * **Multiple Intent Detection**: Splits compound requests into separate tasks
6. **create\_task\_for\_category**
   * Final stage that creates appropriate tasks for each intent
   * Routes tasks to the proper agent (Nova, Emil, Lola, Ivan)
   * Sets function parameters based on intent type

## Function Responsibilities

| **Function** | **Responsibility** | **Old Method** | **New Method** |
| --- | --- | --- | --- |
| detect\_query\_type\_with\_llm | Query classification | N/A (New) | LLM-based semantic analysis |
| create\_task\_list\_from\_prompt\_async | Task creation orchestration | Hardcoded history detection | LLM-based classification |
| identify\_multiple\_intents\_async | Intent separation | Hardcoded follow-up detection | LLM-based follow-up detection |
| create\_task\_for\_category | Task routing | Unchanged | Unchanged |

## Key Improvements

| **Feature** | **Hardcoded Approach** | **LLM-Based Approach** |
| --- | --- | --- |
| Follow-up detection | Based on pronouns and keywords | Based on semantic meaning |
| History query detection | Requires specific patterns | Understands any phrasing |
| Context awareness | Limited | Full conversation context |
| Adaptability | Requires manual updates | Automatically adapts |
| Maintenance | High | Low |

## The Core: LLM-Based Query Detection

python

@log\_function\_call

async def detect\_query\_type\_with\_llm(self, prompt: str, previous\_prompts=None):

"""Use LLM to detect if a prompt is a history query or follow-up question."""

# Context for the LLM detection

context = """

You are a query analysis assistant. Analyze the question and determine:

1. If it's a HISTORY query (asking about previous interactions)

2. If it's a FOLLOW-UP question (relies on previous context)

3. If it's a STANDALONE question (makes sense without context)

For a HISTORY query, respond with: "TYPE: HISTORY"

For a FOLLOW-UP, respond with: "TYPE: FOLLOW-UP"

For a STANDALONE, respond with: "TYPE: STANDALONE"

"""

# Create message with conversation context

if previous\_prompts:

previous\_context = "\n".join([f"Previous Q{i+1}: {p}"

for i, p in enumerate(previous\_prompts)])

message = f"{previous\_context}\n\nNew Question: {prompt}"

else:

message = f"Question: {prompt}"

# Get LLM classification

response = await run\_open\_ai\_ns\_async(message, context)

# Return structured results

return {

"is\_history\_query": "TYPE: HISTORY" in response.upper(),

"is\_follow\_up": "TYPE: FOLLOW-UP" in response.upper(),

"is\_standalone": "TYPE: STANDALONE" in response.upper()

}

## Updating History Query Detection

python

# BEFORE: Hardcoded patterns

past\_indicators = ["did", "was", "asked", "had", "previous", "earlier", "last"]

question\_words = ["what", "which", "when", "where", "how", "tell me"]

is\_history\_query = (any(word in prompt\_lower for word in past\_indicators) and

any(word in prompt\_lower for word in question\_words))

# AFTER: LLM-based detection

query\_type = await self.detect\_query\_type\_with\_llm(prompt, previous\_prompts)

is\_history\_query = query\_type["is\_history\_query"]

## Updating Follow-up Detection

python

# BEFORE: Hardcoded patterns

reference\_words = [

"it", "its", "it's", "they", "them", "their", "there", "this", "that",

"these", "those", "he", "she", "his", "her", "previous", "last"

]

words = re.findall(r'\b\w+\b', prompt.lower())

is\_follow\_up = any(word in reference\_words for word in words)

# AFTER: LLM-based detection

query\_type = await self.detect\_query\_type\_with\_llm(prompt, previous\_prompts)

is\_follow\_up = query\_type["is\_follow\_up"]

## Results: Detection Accuracy

* Correctly identifies context-dependent queries without explicit references
* Properly categorizes questions about past interactions
* Differentiates between standalone questions and follow-ups
* Understands the conversation context holistically

## Benefits of LLM-Based Approach

* **Improved User Experience**: Better handling of natural conversation flow
* **More Accurate Task Routing**: Correct detection means correct handling
* **Reduced Maintenance**: No need to update keyword lists
* **Future-Proof**: Adapts to new language patterns automatically
* **Simplified Code**: Removes complex pattern matching logic

## Graceful Fallback

The implementation includes fallback to original pattern matching when LLM fails:

python

except Exception as e:

print(f"Error in LLM query detection: {str(e)}")

# Fallback to original heuristics if LLM call fails

is\_history = (any(word in prompt\_lower for word in history\_words) and

any(word in prompt\_lower for word in question\_words))

is\_follow\_up = any(word in prompt\_lower for word in follow\_up\_words)

or len(prompt\_lower.split()) <= 4

## Conclusion

* LLM-based detection provides more natural conversation handling
* Implementation is simple but powerful
* Fallback mechanisms ensure reliability
* Code is more maintainable and adaptable
* User experience significantly improved