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

File Location

`src/utils/do_maths.py`

Overview

The `do_maths.py` file implements mathematical calculation functionality for the multi-agent framework. It provides a function to solve mathematical problems using a combination of local pattern matching and, when needed, LLM assistance. The file enables the system to handle basic arithmetic, percentages, powers, and other common mathematical operations.

Key Responsibilities

- Parse and solve mathematical expressions from natural language
- Handle various mathematical formats (percentages, fractions, powers, etc.)
- Provide accurate calculation results with explanations
- Fall back to LLM for complex calculations
- Store calculation results in the knowledge base
- Extract numerical expressions from text descriptions

Core Functionality

Main Function Definition

```python
@log_function_call
def do_maths(kb: KnowledgeBase, prompt: str, input2: str = "-") -> str:
    """
    Performs mathematical calculations based on the input prompt.
    Enhanced to handle a wide range of math expressions including percentages.
    Falls back to LLM for complex calculations.

    Parameters:
        kb (KnowledgeBase): The knowledge base instance
        prompt (str): The math expression or question to solve
        input2 (str): Optional additional input (required by function mapping)

    Returns:
        str: The result of the calculation with explanation
    """
    print(f"Calculating mathematical expression: {prompt}")

    # Try to solve with local calculation logic
    result, answer = attempt_local_calculation(prompt)

    # If local calculation failed, use LLM as fallback
    if result is None:
        print(f"Local calculation failed, using LLM for: {prompt}")
        context = (
```

```
            "You are a math problem solver. Calculate the answer to the given problem.
"
            "For simple arithmetic, just provide the answer. "
            "For complex calculations, show your work step by step."
        )
        llm_result = run_open_ai_ns(prompt, context)

        # Store the LLM result
        kb.set_item("math_result", llm_result)
        kb.set_item("math_answer", llm_result)
        kb.set_item("final_report", llm_result)

        print(f"LLM math calculation result: {llm_result}")
        return llm_result

    # Store the local calculation result in the knowledge base
    kb.set_item("math_result", result)
    kb.set_item("math_answer", answer)
    kb.set_item("final_report", answer)

    print(f"Math calculation result: {answer}")
    return answer
```

Local Calculation Logic

The function for performing calculations locally using pattern matching:

```
def attempt_local_calculation(prompt):
    """
    Attempts to perform a calculation locally using regex pattern matching.
    Supports various formats including percentages, fractions, and basic arithmetic.

    Parameters:
        prompt (str): The mathematical expression to evaluate

    Returns:
        tuple: (result, answer_text) if successful, (None, None) if failed
    """
    try:
        # Normalize the prompt
        normalized_prompt = prompt.lower().replace('×', '*').replace('÷', '/')

        # Pattern for percentage calculations
        percentage_pattern = r'(?:what\s+is\s+)?(\d+\.?\d*)%\s+(?:of)\s+(\d+\.?\d*)'
        percentage_of_pattern = r'(?:what\s+is\s+)?(\d+\.?\d*)\s+(?:percent\s+of)\s+
(\d+\.?\d*)'

        # Pattern for basic arithmetic operations with "what is" optional
        basic_math_pattern = r'(?:what\s+is\s+)?(\d+\.?\d*)\s*([\+\-\*\/])\s*(\d+\.?
\d*)'

        # Pattern for square root
        sqrt_pattern = r'(?:what\s+is\s+)?(?:the\s+)?(?:square\s+root\s+of)\s+(\d+\.?
```

```
\d*)'

        # Pattern for powers
        power_pattern = r'(?:what\s+is\s+)?(\d+\.?\d*)\s+
(?:to\s+the\s+power\s+of|raised\s+to\s+the\s+power\s+of|\^)\s+(\d+\.?\d*)'

        # Check for percentage calculation
        percentage_match = re.search(percentage_pattern, normalized_prompt)
        percentage_of_match = re.search(percentage_of_pattern, normalized_prompt)
        if percentage_match:
            percentage = float(percentage_match.group(1))
            base_value = float(percentage_match.group(2))
            result = (percentage / 100) * base_value
            answer = f"{percentage}% of {base_value} is {result}"
            return result, answer
        elif percentage_of_match:
            percentage = float(percentage_of_match.group(1))
            base_value = float(percentage_of_match.group(2))
            result = (percentage / 100) * base_value
            answer = f"{percentage}% of {base_value} is {result}"
            return result, answer

        # Check for square root
        sqrt_match = re.search(sqrt_pattern, normalized_prompt)
        if sqrt_match:
            num = float(sqrt_match.group(1))
            if num < 0:
                return None, "Error: Cannot take square root of a negative number"
            result = math.sqrt(num)
            answer = f"The square root of {num} is {result}"
            return result, answer

        # Check for powers
        power_match = re.search(power_pattern, normalized_prompt)
        if power_match:
            base = float(power_match.group(1))
            exponent = float(power_match.group(2))
            result = math.pow(base, exponent)
            answer = f"{base} raised to the power of {exponent} is {result}"
            return result, answer

        # Check for basic arithmetic
        math_match = re.search(basic_math_pattern, normalized_prompt)
        if math_match:
            num1 = float(math_match.group(1))
            op = math_match.group(2)
            num2 = float(math_match.group(3))

            if op == '+':
                result = num1 + num2
                answer = f"The result of {num1} + {num2} is {result}"
            elif op == '-':
```

```
                result = num1 - num2
                answer = f"The result of {num1} - {num2} is {result}"
            elif op == '*':
                result = num1 * num2
                answer = f"The result of {num1} * {num2} is {result}"
            elif op == '/':
                if num2 == 0:
                    return None, "Error: Division by zero"
                result = num1 / num2
                answer = f"The result of {num1} / {num2} is {result}"
            else:
                return None, None

            return result, answer

        # Handle simple numeric inputs
        if normalized_prompt.strip().replace(' ', '').isdigit():
            result = float(normalized_prompt.strip())
            answer = f"The number is {result}"
            return result, answer

        # No pattern matched, will fall back to LLM
        return None, None

    except Exception as e:
        print(f"Error in local calculation: {str(e)}")
        return None, None
```

Key Features

1. **Local Calculation**: Uses regex pattern matching for efficient local calculations
2. **Fallback Mechanism**: Falls back to LLM for complex or unrecognized expressions
3. **Format Support**: Handles percentages, square roots, powers, and basic arithmetic
4. **Input Normalization**: Normalizes input for consistent processing
5. **Error Handling**: Handles division by zero and other calculation errors
6. **Natural Language Support**: Works with conversational math questions ("what is...")
7. **Results Formatting**: Provides human-readable answers with explanations

Integration

- Used by Nova to handle mathematical calculation tasks
- Stores results in the knowledge base for retrieval
- Leverages LLM capabilities for complex problems
- Accessible through function mapping system

Workflow

1. Receives a mathematical expression or question
2. Normalizes the input (lowercase, symbol replacement)
3. Attempts to match known mathematical patterns
4. If a pattern matches, performs the calculation locally
5. If no pattern matches or an error occurs, falls back to LLM

6. Formats the result with an explanation
7. Stores the result in the knowledge base
8. Returns the formatted answer

Implementation Notes

- Uses regular expressions for pattern matching
- Handles multiple syntax variants for the same operation
- Provides specific error messages for common errors
- Uses the math library for square roots and powers
- Formats results with both the calculation and the answer
- Stores both raw numerical results and formatted answers