# ***Flow Breakdown of Async Tests with Code Mapping***

## **Test 1: Multi-Intent Processing**

Input: "what is 30% of 200 and tell me about renewable energy"

1. main.py: async\_main() → Initial prompt collection

└─ 2. main.py: asyncio.gather() → Runs task creation in parallel

└─ 3. nova.py: create\_task\_list\_from\_prompt\_async() → Processes prompt

└─ 4. nova.py: identify\_multiple\_intents\_async() → Identifies 2 intents

├─ 5a. nova.py: run\_open\_ai\_ns\_async() → LLM call to split intents

├─ 6a. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes intent 1 as "do\_maths"

└─ 6b. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes intent 2 as "general\_question"

└─ 7. nova.py: create\_task\_for\_category() → Creates 2 separate tasks

8. main.py: process\_prompt\_tasks() → Sequential processing of the 2 tasks

├─ 9a. nova.py: handle\_task\_async() → Executes math task

│ └─ 10a. do\_maths.py: do\_maths() → Calculates 30% of 200

│ └─ 11a. do\_maths.py: attempt\_local\_calculation() → Local percentage calculation

└─ 9b. nova.py: handle\_task\_async() → Executes general knowledge task

└─ 10b. general\_knowledge.py: answer\_general\_question() → Answers renewable energy question

└─ 11b. open\_ai\_utils.py: run\_open\_ai\_ns() → LLM call to answer question

## **Test 2: Concurrent Different Operation Types**

Inputs: "what is 25 \* 4", "create an energy model for solar in France", "what is the capital of Spain"

1. main.py: async\_main() → Collects 3 prompts

└─ 2. main.py: asyncio.gather() → Parallel task creation for all 3 prompts

├─ 3a. nova.py: create\_task\_list\_from\_prompt\_async() → For prompt 1 (math)

│ └─ 4a. nova.py: identify\_multiple\_intents\_async() → Identifies 1 intent

│ └─ 5a. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes as "do\_maths"

│ └─ 6a. nova.py: create\_task\_for\_category() → Creates math task

│

├─ 3b. nova.py: create\_task\_list\_from\_prompt\_async() → For prompt 2 (energy model)

│ └─ 4b. nova.py: identify\_multiple\_intents\_async() → Identifies 1 intent

│ └─ 5b. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes as "Energy Model"

│ └─ 6b. nova.py: create\_task\_for\_category() → Creates energy model task

│ └─ 7b. functions\_registery.py: extract\_model\_parameters() → Extracts model parameters

│ └─ 8b. nova.py: get\_energy\_parameters\_from\_user\_async() → Prompts for missing params

│

└─ 3c. nova.py: create\_task\_list\_from\_prompt\_async() → For prompt 3 (general knowledge)

└─ 4c. nova.py: identify\_multiple\_intents\_async() → Identifies 1 intent

└─ 5c. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes as "general\_question"

└─ 6c. nova.py: create\_task\_for\_category() → Creates general knowledge task

9. main.py: asyncio.gather() → Parallel processing of all 3 prompts' tasks

├─ 10a. main.py: process\_prompt\_tasks() → For prompt 1 (math)

│ └─ 11a. nova.py: handle\_task\_async() → Executes math task

│ └─ 12a. do\_maths.py: do\_maths() → Math calculation

│ └─ 13a. open\_ai\_utils.py: run\_open\_ai\_ns() → LLM fallback for calculation

│

├─ 10b. main.py: process\_prompt\_tasks() → For prompt 2 (energy model)

│ └─ 11b. emil.py: handle\_task\_async() → Delegated to Emil

│ └─ 12b. emil.py: verify\_parameters\_async() → Verifies parameters

│ └─ 13b. functions\_registery.py: process\_emil\_request() → Processes energy model

│ └─ 14b. functions\_registery.py: create\_comprehensive\_model() → Creates model

│ └─ 15b. functions\_registery.py: create\_simple\_comprehensive\_xml() → Creates XML file

│

└─ 10c. main.py: process\_prompt\_tasks() → For prompt 3 (general knowledge)

└─ 11c. nova.py: handle\_task\_async() → Executes general knowledge task

└─ 12c. general\_knowledge.py: answer\_general\_question() → Answers question

└─ 13c. open\_ai\_utils.py: run\_open\_ai\_ns() → LLM call to answer question

## **Test 3: Multiple Math Calculations**

Inputs: 5 math calculations

1. main.py: async\_main() → Collects 5 math prompts

└─ 2. main.py: asyncio.gather() → Parallel task creation for all 5 prompts

└─ 3. nova.py: create\_task\_list\_from\_prompt\_async() → For each prompt (×5)

└─ 4. nova.py: identify\_multiple\_intents\_async() → Identifies intents

└─ 5. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes all as "do\_maths"

└─ 6. nova.py: create\_task\_for\_category() → Creates math tasks

7. main.py: asyncio.gather() → Parallel processing of all 5 prompts' tasks

└─ 8. main.py: process\_prompt\_tasks() → For each prompt (×5)

└─ 9. nova.py: handle\_task\_async() → Executes math tasks in parallel

└─ 10. do\_maths.py: do\_maths() → Math calculations

├─ For prompts 1-3: attempt\_local\_calculation() → Local calculation succeeds

│ └─ 11a. do\_maths.py: Returns results directly (30.0, 70.0, 110.0)

└─ For prompts 4-5: attempt\_local\_calculation() → Local calculation fails

└─ 11b. open\_ai\_utils.py: run\_open\_ai\_ns() → LLM fallback for calculation

## **Test 4: Complex Energy Modeling**

Input: "build a comprehensive energy model for wind and solar in France, Germany and Spain with electricity and hydrogen as carriers"

1. main.py: async\_main() → Processes complex prompt

└─ 2. nova.py: create\_task\_list\_from\_prompt\_async() → Processes energy model request

└─ 3. nova.py: identify\_multiple\_intents\_async() → Identifies 1 intent

└─ 4. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes as "Energy Model"

└─ 5. nova.py: create\_task\_for\_category() → Creates energy model task

└─ 6. functions\_registery.py: extract\_model\_parameters() → Extracts parameters

├─ Locations: ['spain', 'france', 'germany']

├─ Generation types: ['wind', 'solar', 'hydro']

└─ Energy carriers: ['electricity', 'hydrogen']

7. main.py: process\_prompt\_tasks() → Processes energy model task

└─ 8. emil.py: handle\_task\_async() → Delegates to Emil

└─ 9. emil.py: verify\_parameters\_async() → Checks parameters

└─ 10. functions\_registery.py: process\_emil\_request() → Processes request

└─ 11. functions\_registery.py: extract\_model\_parameters() → Re-extracts parameters

└─ 12. functions\_registery.py: create\_comprehensive\_model() → Creates model

└─ 13. functions\_registery.py: create\_simple\_comprehensive\_xml() → Creates XML

├─ BUG: Locations processed as individual characters

└─ File created successfully

## **Test 5: Error Handling**

Input: "calculate the square root of -1"

1. main.py: async\_main() → Processes math error prompt

└─ 2. nova.py: create\_task\_list\_from\_prompt\_async() → Processes math request

└─ 3. nova.py: identify\_multiple\_intents\_async() → Identifies 1 intent

└─ 4. open\_ai\_utils.py: open\_ai\_categorisation\_async() → Categorizes as "do\_maths"

└─ 5. nova.py: create\_task\_for\_category() → Creates math task

6. main.py: process\_prompt\_tasks() → Processes math task

└─ 7. nova.py: handle\_task\_async() → Executes math task

└─ 8. do\_maths.py: do\_maths() → Calculates square root

└─ 9. do\_maths.py: attempt\_local\_calculation() → Detects negative number error

└─ 10. open\_ai\_utils.py: run\_open\_ai\_ns() → LLM fallback handles error case

└─ Returns "i" with mathematical explanation

## **Key Insights from Flow Analysis:**

1. **Parallel Processing Architecture**:  
   * asyncio.gather() in main.py enables true parallel execution
   * async/await pattern used consistently throughout the codebase
2. **Task Delegation Chain**:  
   * Nova → intent identification → categorization → task creation → agent delegation
3. **Fallback Mechanisms**:  
   * Local calculation errors trigger LLM fallbacks (Test 5)
   * Parameter extraction includes validation and user prompting (Test 2)
4. **Bug Location**:  
   * Location processing issue in create\_comprehensive\_model() (Test 4)
   * Parser issues in attempt\_local\_calculation() for certain math expressions (Test 3)
5. **Performance Bottlenecks**:  
   * LLM API calls (run\_open\_ai\_ns/run\_open\_ai\_ns\_async) are the slowest operations
   * Multiple consecutive API calls for intent detection and categorization

This flow mapping gives a clear picture of the execution path for each test, highlighting where parallel processing occurs and where bottlenecks or bugs exist.