detailed breakdown comparing the flow chart to the actual output, highlighting key files and code snippets:

1. User Prompt Intake

* Flow Chart: Yellow box "User Prompt"
* Code Location: main.py

user\_prompt = input("\n\nUser prompt: ")

* Actual Implementation: Successfully captured the multi-intent prompt "what is 7 \* 7, the author of the harry potter books, what is the capital of france and build a model"

1. Coordination Agent (Nova)

* Flow Chart: Salmon-colored "Coordination Agent (Nova)" box
* Code Location: nova.py

class Nova(BaseAgent):

def create\_task\_list\_from\_prompt(self, prompt: str) -> List[Task]:

# Multiple intent identification logic

multiple\_intents = self.identify\_multiple\_intents(prompt)

* Key Method: identify\_multiple\_intents() in nova.py
* Actual Implementation: Successfully identified 4 separate intents

1. Extract Metadata from Prompt

* Flow Chart: Green "Extract Metadata from Prompt" box
* Code Location: functions\_registery.py

def extract\_model\_parameters(prompt):

params = {

"locations": [],

"generation\_types": [],

"energy\_carriers": [],

"model\_type": "single"

}

# Extract relevant parameters from prompt

* Actual Implementation: Partially worked - detected no parameters initially, required user interaction

1. Centralized Task List Creator

* Flow Chart: Red "Centralized Task List Creator" box
* Code Location: nova.py

def create\_task\_list\_from\_prompt(self, prompt: str) -> List[Task]:

tasks = []

for intent\_info in multiple\_intents:

category = open\_ai\_categorisation(intent\_text, csv\_path)

task = self.create\_task\_for\_category(intent\_text, category)

tasks.append(task)

return tasks

* Actual Implementation: Successfully created 4 distinct tasks with correct categorization

1. Dependency Graph & Parameter Analysis

* Flow Chart: Enhanced boxes for Dependency Graph and Parameter Analysis
* Code Location: nova.py

def create\_task\_for\_category(self, intent\_text: str, category: str) -> Task:

# Determine appropriate agent and function based on category

if category.lower() == "do\_maths":

target\_agent = "Nova"

function\_name = "do\_maths"

elif category.lower() == "Energy Model":

target\_agent = "Emil"

function\_name = "process\_emil\_request"

# Collect missing parameters

missing\_energy\_params = []

if not params.get("locations"):

missing\_energy\_params.append("location")

* Actual Implementation: Worked well, especially for energy model task requiring additional parameters

1. Missing Arguments Check

* Flow Chart: Green "Check for Missing Arguments" box
* Code Location: nova.py

def get\_energy\_parameters\_from\_user(self, missing\_params: List[str]) -> Dict[str, Any]:

collected\_args = {}

for param in missing\_params:

# Interactive parameter collection

print(f"Nova: Could you provide {param} for the energy model?")

user\_response = input("> ").strip()

collected\_args[param] = user\_response

return collected\_args

* Actual Implementation: Successfully collected missing parameters (location: all, generation: wind, carrier: methane)

1. Execution Strategy

* Flow Chart: Decision box "Parallel or Sequential?"
* Code Location: main.py

for i, task in enumerate(top\_level\_tasks, 1):

print(f"\n📋 Task {i}/{len(top\_level\_tasks)}: {task.name}")

# Sequential execution

if task.agent == "Nova":

nova.handle\_task(task)

elif task.agent == "Emil":

emil.handle\_task(task)

* Actual Implementation: Used sequential execution

1. Task Delegation

* Flow Chart: Various assistant and function boxes
* Code Location: main.py

if task.agent == "Nova":

nova.handle\_task(task)

elif task.agent == "Emil":

emil.handle\_task(task)

* Actual Implementation: Correctly delegated tasks (3 to Nova, 1 to Emil)

1. Function Execution

* Flow Chart: Blue function execution boxes
* Relevant files: do\_maths.py, general\_knowledge.py, functions\_registery.py
* Actual Implementation: Successfully executed each task type

1. Results Aggregation

* Flow Chart: Green "Task Results Aggregation" box
* Code Location: main.py

all\_results = []

for task in top\_level\_tasks:

if hasattr(task, 'result') and task.result:

all\_results.append(task.result)

* Actual Implementation: Combined results from all 4 tasks

1. Quality Control

* Flow Chart: Enhanced "Quality Control Analysis" box
* Current Implementation: Minimal implementation
* Potential improvement area

Discrepancies between Flow Chart and Actual Implementation:

1. No true parallel task execution
2. Limited error handling
3. Minimal quality control analysis
4. No sophisticated dependency graph creation

Potential Improvements:

* Implement parallel task processing
* Enhance error handling mechanisms
* Add more robust quality control
* Develop a more sophisticated dependency resolution system