Task: DAG (Directed Acyclic Graph) Executor in Python

Objective:

Your task is to implement a DAG executor in plain Python + NumPy. The executor should allow defining and running a computation pipeline where each step (node) depends on the availability of its inputs. Execution should be data-driven rather than following a fixed order.

Requirements:

1. Graph Representation
    - Implement a way to define a pipeline as a DAG.
    - Each node in the DAG represents a computational step with inputs and outputs.
2. Execution Logic
    - Nodes should execute only when all their required inputs are available.
    - Execution should respect data dependencies rather than the order in which nodes are added.
3. Concurrency & Optimization (Bonus)
    - If possible, nodes that have independent dependencies should execute in parallel.
    - Consider efficient memory usage and computational performance.
4. Modularity & Readability
    - Your solution should be clean, modular, and easy to understand.
    - Include appropriate comments and docstrings.

Input & Output:

- Define a simple API for constructing and executing the DAG.
- Provide an example DAG with at least 3–5 nodes to demonstrate the executor.

Constraints:

- Use only built-in Python libraries and NumPy.
- Avoid external frameworks for graph processing (e.g., NetworkX).
- Your implementation should work with arbitrary numerical computations.

Evaluation Criteria:

1. Correctness – Does the executor correctly execute nodes based on dependencies?
2. Optimization – Is execution efficient in terms of time and space?
3. Readability – Is the code well-structured and easy to follow?
4. Modularity – Is the solution easily extendable?
5. Concurrency (Bonus) – Does the implementation support parallel execution?

Test Example

This example processes a random 2D NumPy array and computes statistics before and after normalization.

Processing Steps:

1. Generate Random Data → Creates a 2D array with random values.
2. Compute Raw Stats → Computes min, max, mean, and std of the original array.
3. Normalize Array → Scales the array to the range [0,1].
4. Compute Normalized Stats → Computes min, max, mean, and std of the normalized array.
5. Merge and Print Stats → Collects and prints the results.

Use https://mermaid.live/ to display the diagram, since it is not exported in PDF.

```
graph TD;
    A[Generate Random Data] --> B[Compute Raw Stats];
    A --> C[Normalize Array];
    C --> D[Compute Normalized Stats];
    B --> E[Merge and Print Stats];
    D --> E;
```

Step	Description	NumPy Function
Generate Random Data	Creates a 2D array with random values.	`np.random.randint()`
Compute Raw Stats	Computes min, max, mean, std of the raw array.	`np.min()`, `np.max()`, `np.mean()`, `np.std()`
Normalize Array	Scales data to [0,1] using min-max normalization.	`(arr - min) / (max - min)`
Compute Normalized Stats	Computes min, max, mean, std of the normalized array.	`np.min()`, `np.max()`, `np.mean()`, `np.std()`
Merge and Print Stats	Collects both sets of stats, combines them into single dictionary and prints them.	-

---

Submission

- Provide a single Python script or a small module.
- Include a README explaining how to use your DAG executor.

Copyright