Embedded AI - Milestone 3

Nicholas Callahan
ncallah3@emich.edu

Ryan Piza
rpiza@emich.edu

June 16, 2026

Team 5: Technical Write-Up

Visualization Script: Basic and Advanced & NN Definition: MobileNetV2

1 What limitation/missing feature did you identify?

The `nn_dataflow` tool output defaults to using stdout. A user can use the output redirect operator, `>`, to send the output to a file, but the tool does not provide a built-in way to visualize the results.

The `nn_dataflow` tool has defined several neural networks, such as AlexNet, VGG-16, GoogLeNet, and ResNet-152, but is missing many other NN architectures. MobileNetV2 is popular for use on resource-constrained hardware [1] and could benefit from a dedicated dataflow scheduling tool focused on energy efficiency.

2 What files/classes did you modify?

No existing files or classes were modified. Three new Python files were added that are standalone:

- **visualization/display_results_basic.py**: a script that reads a `nn_dataflow` JSON output file and generates a three-panel bar chart saved as a `.png` file.
  - Uses only the Python standard library (`json`, `sys`, `os`, `re`) and `matplotlib`. Matplotlib is not a dependency of `nn_dataflow` and is installed separately via pip.
- **visualization/display_results_energy.py**: a script that reads a `nn_dataflow` JSON output file and generates a bar chart to break down energy usage, saved as a `.png` file.
  - Uses only the Python standard library (`json`, `sys`, `os`, `re`) and `matplotlib`. Matplotlib is not a dependency of `nn_dataflow` and is installed separately via pip.
- **nn_dataflow/nns/mobilenet_v2.py**: a NN definition for MobileNetV2 that follows the same structure and conventions as existing definitions in `./nn_dataflow/nns/`.
  - The file uses the existing `Network`, `InputLayer`, `ConvLayer`, `PoolingLayer`, and `FCLayer` classes from `nn_dataflow.core`.

3 What new capability does the tool now have?

The tool now has two scripts that create a visual summary of any `nn_dataflow` search result. The scripts validate the following inputs:

- Checks that a JSON filename argument was provided.
- Verifies the file exists and has a `.json` extension.
- Catches JSON missing required keys:
  - Basic: `total_cost`, `total_time`, `active_node_pes`
  - Energy: `cost_op`, `cost_access`, `cost_noc`

- Validates the output filename for empty input, invalid characters, and existing file conflicts.

The tool can now run dataflow scheduling on MobileNetV2 by passing `mobilenet_v2` as the network argument to `nn_dataflow_search.py`. This enables energy, delay, and ED product optimization searches on a more modern architecture that was not previously supported.

4 What difficulties did you encounter?

The initial attempt was to approximate per-layer energy and latency by distributing the totals proportionally by active PE count. Looking at the key–value pairs in the output of a JSON file, the `schedules` key was discovered. It provided information, including per-layer cost and time values from the optimal schedule found for each layer. The script was updated to use these directly in order to produce a more accurate result.

Also, the `plt.show()` call prompts a warning when using Ubuntu/WSL to run `nn_dataflow` because there is no interactive display; everything is done through the terminal. This was handled by wrapping the call in a `try/except` block and printing the `explorer.exe` command the user needs to open the saved file on Windows.

For `display_results_energy.py`, most of the code was entirely reused from `display_results_basic.py`, so the main problems related to the formatting of the graph. We decided to change the x-axis (layer name) rotation to be straight, so that the layer names are more readable for neural networks with many layers and do not overlap one another. There was another problem with some neural networks, where the total energy use for a given layer was too large and was not scaled properly, so the program now automatically adds a buffer to the y-axis heights, which solved this problem.

For MobileNetV2, many challenges were encountered. A core challenge was understanding how the project's layer API maps to MobileNetV2's architecture. Another was verifying without being able to run inference. Because `nn_dataflow` does not execute the network, there is no prediction output to validate against. Correctness was determined by running the `nn_dataflow` test tool and comparing the network against Table 2 of the MobileNetV2 paper [2] to confirm that every block's output channels, spatial dimensions, and expansion ratios matched.

5 GitHub Repository

Forked repository: `https://github.com/rpiza12/nn_dataflow`
The most recent relevant commit: `0238389`. The others, [`329a35c`, `eef5a9b`, `ba7a05a`], were to remove files automatically generated by windows.
Pull request: `https://github.com/stanford-mast/nn_dataflow/pull/29`

References

[1] GeeksforGeeks. MobileNet V2 architecture in computer vision. `https://www.geeksforgeeks.org/computer-vision/mobilenet-v2-architecture-in-computer-vision/`, 2024. Accessed: 22 April 2026.

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[1] [2]