Parallel Model Splitting

Overview

When working with large, complex AI models, the computational demands can be immense. To speed up the inference or training process, you can leverage the power of parallel computing by splitting the model across multiple devices, such as CPU cores or GPUs.

What happens if the GPU fails ? Read : Fault Tolerance for Parallel Model Splitting

How it Works

The key steps in this parallel model splitting approach are:

1. **Split the Model**: The original AI model is divided into multiple "parts", with each part containing a subset of the model's layers. These parts are then assigned to different computing devices.
2. **Distribute the Input Data**: The input data that needs to be processed by the model is divided into smaller "chunks" and distributed across the available devices.
3. **Run the Model Parts in Parallel**: Each device runs its assigned model part on the corresponding input data chunk, with the computations happening simultaneously across the devices.
4. **Combine the Outputs**: After the model parts have finished processing their assigned input chunks, the individual outputs are combined back into a single, complete output.

By splitting the model and distributing the workload across multiple devices, you can leverage the combined computing power to process the input data much faster than running the entire model on a single device.

Benefits

- **Increased Speed**: Parallelizing the model execution can significantly reduce the overall processing time, especially for large and computationally intensive

AI models.

- **Efficient Resource Utilization**: By distributing the workload, you can make better use of the available computing resources (CPU cores, GPUs) on your system.
- **Scalability**: As the number of available devices increases, the parallel processing approach can scale accordingly to handle even larger models and datasets.

Usage Example

Here's an example of how the `ModelSplitter` class can be used to split a PyTorch model across multiple devices:

```
model = MyModel()
splitter = ModelSplitter(model, num_devices=4)
input_data = torch.randn(100, 64)
output = splitter.forward(input_data)
```

In this example, the `ModelSplitter` class takes the original `MyModel` instance and splits it across 4 devices. The `forward()` method then distributes the input data, runs the model parts in parallel, and combines the outputs.

Conclusion

Splitting an AI model across parallel or multi-threaded compute is a powerful technique for accelerating the processing of large and complex models. By leveraging the combined computing power of multiple devices, you can significantly reduce the overall runtime and make more efficient use of your available resources.