##
Document: MESO Test Script Running.pdf
Description: Documentation and Verification Guide for the MESO IMC Architecture Simulation.
Created: 2025-12-09
Last Modified: 2026-01-21
##

Step 1: Simulation of MESO IMC to understand the output Error:

a) Script to run: Functional_Simulation/TestScripts/MESODotProductCheck.py
b) Overview of the script: This is a Python 3 script developed for the verification and quantitative error analysis of the MESO In-Memory Computing (IMC) dot product module. It compares the output of the IMC simulation against a standard high-precision floating-point (float32) matrix multiplication.
c) Execution Instructions:
    a. Prerequisites: Python3 environment, torch, numpy, cmath libraries.
    b. The MESO IMC implementation file (Functional_Simulation/DotProductMESO/ MLDotProduct_MESO.py) must be accessible via the specified path (`sys.path.append('../')`).
    c. Command: Execute the script directly from your terminal: python3 MESODotProductCheck.py
d) Configuration Variables (editable in code): The script's testing scope and behavior are controlled by variables within the Functional_Simulation/TestScripts/ MESODotProductCheck.py file:

Variable	Location	Description	Default	Changes
bit_width (Global)	Line 68 (Input)	The quantization bit width. (Changed interactively)	8	Choose any value between 2 and 32 (range enforced by code)
mode_input (Global)	Line 92 (Input)	Determines the range of the input matrices (Changed interactively)	3 (“ALL_POS”)	Choose between the three available modes
sim_input (Global)	Line 113 (Input)	Allows users to set depth and sequence length or use default simulation settings (Adjustable interactively).	2 (“PRE”)	Choose between the two available options.
Depth	Line 148	The feature Depth	64	Adjust single-head depth to vary complexity. Change interactively via **sim_input option “1”.** Very large

				values may cause memory-related failures.
sizeL	Line 149	The sequence length	128	Adjust sequence length to vary complexity. Change interactively via **sim_input option "1".** Very large values may cause memory-related failures.
batch	Line 150	The batch size for the input tensors	1	Increase for more realistic, parallelized testing.
num_tests	Line 147	The number of random input tests performed for each sequence length	100 ("PRE"), 10 ("USER")	Increase for higher statistical confidence in the error metric. For sim_input='1', which selects the "USER" mode, allowing user to give the depth and sequence length, this defaults to 1.
device	Line 50	The execution device	mps or cuda if available, else cpu	Hardcode to "cpu" or "cuda" if you need to force a specific device.
torch.manual_seed(2)	Line 230	The seed for random number generation.	2	Change the seed to generate different random input data for testing.

e) Testing and Iteration Logic: The script uses a nested loop structure to comprehensively test the IMC module for the default setting:
   a. Outer Loop (Sequence Length): It iterates based on the exponent l1 defined in range(6, 14, 1). The sequence length (sizeL) is calculated as 2^{l1}. This systematically tests input sizes from **64** up to **8192**.
   b. Inner Loop (Test Cases): For a fixed sizeL, this loop runs num_tests (default 100) times. In each iteration, new random input tensors (Weight and BT) of shape (batch, sizeL, Depth) are generated based on the selected Simulation Mode.
   c. Calculation and Comparison:
      a. IMC output (Output) = model(Weight, BT) is calculated using the MESO simulation.
      b. Reference Output (OutNew)=torch.matmul(Weight,BT.permute(0,2,1)) is calculated using standard PyTorch floating-point operations.

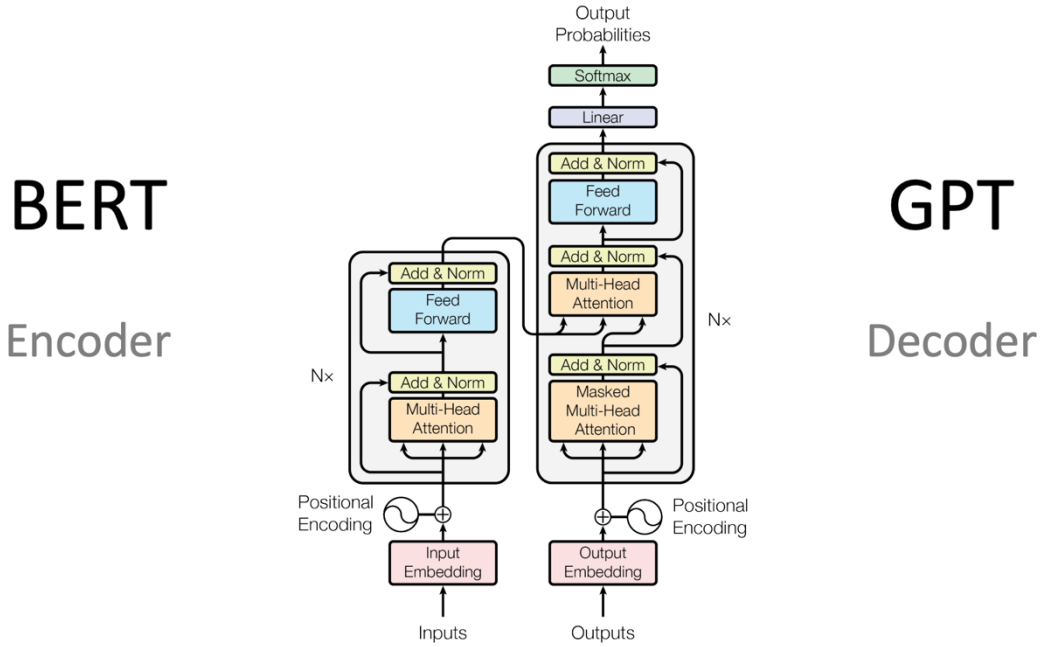
d. Error Metric: The percentage error (ErrorC) is calculated as: Error= |OutNew-Output|*100/ |OutNew|. The mean error across all num_tests is printed for each sequence length.

f) Select sim_input (Simulation Mode) = “1” (Line 113) to interactively set depth and sequence length. This mode runs a single test case only. Set depth between 2 and 256 (Line 124) and sequence length (Line 133) between 2 and 2048.

Step 2: Multi30k German to English SLM transformer model simulation:

a) Script to run:
Functional_Simulation/TestScripts/Transformer_Simulation/scr/TransformerTop.py

b) Overview of the script: This script, TransformerTop.py, serves as the **top-level verification module** for mapping a **small language model** (SLM) with encoder-decoder architecture



onto the **MESO IMC architecture**. The primary goal is to perform functional validation and quantitative error analysis by selectively routing the critical dot-product operations within the **Multi-Head Attention** layer to either:

a. **MESO IMC Simulation:** (mode = 1)
b. **Standard CMOS GPU Reference:** (mode = 0)

The simulation is configured for a **German-English translation task** using the Multi30k database.

c) Execution Instructions:
   a. Prerequisites: Python3 environment, torch, numpy, spacy, regex, sacrebleu libraries.
   b. Necessary Local modules: All the files in Functional_Simulation/TestScripts/Transformer_Simulation/scr/ and all the files in Functional_Simulation/DotProductMESO/ should be accessible.
   c. Command: Execute the script directly from your terminal: python3 TransformerTop.py
   d. Execution Flow:
      1. Configuration Loading: The script imports all constants from Config.py
      2. Model Initializaion: The transformer model is initialized with parameters and moved to the chosen device (mps or cpu here).
      3. Training/Evaluation: The run() function initiates training and validation over the specified number of epochs.

4. Hardware Selection: During the dot-product calculation in attention layers (in Attention.py), the execution path is determined by global mode loaded from Config.py.

d) Configuration Variables (editable in Config.py): The script's testing scope and behavior are controlled by variables within the Functional_Simulation/TestScripts/ Transformer_Simulation/scr/Config.py file:

Variable	Location	Description	Default	Changes
mode	Line 81	IMC simulation/ CMOS GPU simulation selector	1	Choose 1 for MESO IMC simulation, 0 for CMOS GPU simulation
bit_width	Line 82	Quantization bit-width for the MESO IMC architecture	8	Choose between (2,32). Larger bit-widths may increase run-time exponentially.
d_model	Line 60	Dimensionality of embeddings (Hidden size)	512	Increase or decrease based on your system memory to test different complexity levels
n_layers	Line 61	Number of transformer layers	6	Increase or decrease to test different complexity levels
n_heads	Line 62	Number of attention heads	8	Increase or decrease based on your system memory to test different complexity levels
max_len	Line 59	Maximum sequence Length	128	Increase or decrease based on your system memory to test different complexity levels. Choose max_len>80.
epoch	Line 72	Maximum training epochs	50	
init_lr	Line 67	Initial learning rate for the Adam optimizer.	1e-4	Ideal learning rate is between 1e-5 to 1e-3.
warmup	Line 71	Number of steps to gradually increase learning rate in the beginning of training.	4000	Increase or decrease it to check the effect on val_loss and BLEU scores.
patience	Line 70	Number of epochs to wait before reducing learning rate if no improvement.	3	Increase or decrease it to check the effect on val_loss and BLEU scores.
label_smoothing	Line 75	Regularization factor	0.1	Higher values reduce

				overconfidence; may increase val_loss
resume_training	Line 78	Option to load previous best checkpoint and train from this	True	Choose "True" to load the previous best database and resume training. Choose "False" to start training from scratch.
strict_cofig	Line 79	Option to control the overlap requirements between the current configuration and the best checkpoint configuration.	True	Choose "True" to prevent errors.

e) Training and Evaluation: The TransformerTop.py module includes standard machine learning routines:
   a. Training (train() function): Iterates through the train_iter (data loaded via DatasetLoads.py). Calculates **Cross-Entropy Loss** and performs **backpropagation** and **gradient clipping** (clip=1.0). Uses the **Adam optimizer** with a learning rate scheduler (ReduceLROnPlateau).
   b. Evaluation (evaluate_inference() function): Calculates **Cross-Entropy Loss** on the validation set (valid_iter) through teacher forcing. Calculates the **BLEU score** (Bilingual Evaluation Understudy) via greedy autoregression (sacrebleu library used), a standard metric for machine translation quality, by comparing the model's output words to the target words.
   c. Run (run() function): Manages the training loop across epochs. Applies the learning rate scheduler after the warmup phase (first 4,000 training steps). Saves the model state to the "saved/" directory whenever a new best validation loss is achieved, with a maximum of six checkpoints retained. Records epoch-wise metrics for training loss, validation loss, and BLEU score into local text files in the "saved/" directory (train_loss.txt, test_loss.txt, and bleu.txt).