ARTIFACT APPENDIX

A. Abstract

This artifact appendix illustrates how to utilize the proposed LLMCompass framework to evaluate hardware designs running Large Language Model (LLM) workloads. An x86 machine capable of running Python programs is required to reproduce all the results in Figure 5-12.

B. Artifact Checklist

- **Run-time environment:** Python 3.9
- **Hardware:** An x86 machine.
- **Metrics:** Throughput and latency of serving LLM inference. Area and cost of the hardware design.
- **Output:** CSV logs and Figure 5-12 in the paper.
- **Experiments:** Scripts are provided to automate the experimental flow and generate the graphs.
- **How much disk space required (approximately)?:** 1GB.
- **How much time is needed to prepare workflow (approximately)?:** Less than an hour.
- **How much time is needed to complete experiments (approximately)?:** 10 hours.
- **Publicly available?:** On GitHub: https://github.com/HenryChang213/LLMCompass_ISCA_AE.
- **Archived?:** On Zenodo: https://doi.org/10.5281/zenodo.10896161.

C. Description

1) How to access: The artifact is available on Zenodo: https://doi.org/10.5281/zenodo.10896161 and includes all the source code, scripts, and data that are sufficient to reproduce all the experiments in the paper. The artifact is also available on GitHub: https://github.com/HenryChang213/LLMCompass_ISCA_AE.

2) Hardware dependencies: The artifact works on x86 CPUs and has been verified on an Intel Xeon Gold 6242R CPU @ 3.10GHz.

3) Software dependencies: The artifact requires Python3 and has been verified with Python 3.9.19. We recommend using Anaconda to create a virtual environment and install the required packages as below:

```
$ conda create -n llmcompass_ae python=3.9
$ conda activate llmcompass_ae
$ pip3 install scalesim
$ conda install pytorch==2.0.0 -c pytorch
$ pip3 install matplotlib
$ pip3 install seaborn
$ pip3 install scipy
```

4) Installation: We use Python module method so no installation required. One can download the artifact as below:

```
$ git clone https://github.com/
HenryChang213/LLMCompass_ISCA_AE.git
$ cd LLMCompass_ISCA_AE
$ git submodule init
$ git submodule update --recursive
```

D. Experiment workflow

After setting up the environment, one can reproduce the experiments by running the following scripts. These scripts will first generate CSV files and then visualize the results as in the paper. There is no dependency on these scripts and feel free to run them in parallel.

```
# Figure 5 (around 100 min)
$ cd ae/figure5
$ bash run_figure5.sh

# Figure 6 (around 1 min)
$ cd ae/figure6
$ bash run_figure6.sh

# Figure 7 (around 20 min)
$ cd ae/figure7
$ bash run_figure7.sh

# Figure 8 (around 40 min)
$ cd ae/figure8
$ bash run_figure8.sh

# Figure 9 (around 30 min)
$ cd ae/figure9
$ bash run_figure9.sh

# Figure 10 (around 45 min)
$ cd ae/figure10
$ bash run_figure10.sh

# Figure 11 (around 5 min)
$ cd ae/figure11
$ bash run_figure11.sh

# Figure 12 (around 4 hours)
$ cd ae/figure12
$ bash run_figure12.sh
```

Profiling results on real world hardware has been provided in advance (as illustrated in Sec. III-C) and is out of the scope of this artifact.

E. Evaluation and expected result

After running each script above, the corresponding figures will be generated under the corresponding directory as suggested by its name:

- Figure 5a and Figure 5b: `ae\figure5\ab`
- Figure 5c and Figure 5f: `ae\figure5\cf`
- Figure 5d and Figure 5e: `ae\figure5\de`
- Figure 5g: `ae\figure5\g`
- Figure 5h: `ae\figure5\h`
- Figure 5i-l: `ae\figure5\ijkl`
- Figure 6: `ae\figure6`
- Figure 7: `ae\figure7`
- Figure 8: `ae\figure8`
- Figure 9: `ae\figure9`
- Figure 10: `ae\figure10`

- Figure 11: `ae\figure11`
- Figure 12: `ae\figure12`

For comparison, a copy of the expected results can be found in `ae\expected_results`. The figures generated by the scripts should be identical to these expected results.

Generally, the figures generated by the scripts should look identical to the figures in the paper. Sometimes there can be some mismatch because we are still actively improving the framework after the original paper submission. However, the mismatch should not exceed 5%.

The only exception is Figure 7b, where we ran the experiment with the updated code and found out the impact of compute system design on the decoding latency should be less significant than the trend shown in Figure 7b. This is mainly because we add some customized code to better support matrix-vector multiplications, while in the original paper submission we use the matrix multiplication code to simulate matrix-vector multiplications. However, this does not affect the conclusion in the paper that compute system design affects prefill more significant than decoding. We will update this figure in the camera-ready submission.

F. Experiment customization

1) Customized hardware design: In addition to the provided hardware configurations, users can describe their own hardware design with the provided hardware description template `configs\template.json`. More examples can be found in `configs\` and users need to bring their own numbers and set the parameters in the `json` file.

An alternative way is to build up the hardware design bottom-up with the provided Python Class defined in `hardware_model\`. The user needs to define their own `ComputeModule`, `MemoryModule`, `IOModule`, and `InterConnectModule`, and combine them into a `System`.

2) Customized LLM computational graph: In addition to the provided Multi-Head-Attention Transformer, users can describe their own computational graph with the provided operators and primitives, including `Matmul`, `LayerNorm`, `Softmax`, `GeLU`, and `AllReduce`. An example is shown in `software_model\transformer`. The user needs to initialize the operators and combine them into a computational graph in a similar way to PyTorch.