A. Artifact Appendix

A.1 Abstract

On-device Large Language Models (LLMs) are transforming mobile AI, catalyzing applications like UI automation without privacy concerns. Nowadays the common practice is to deploy a single yet powerful LLM as a general task solver for multiple apps, i.e. LLMaaS, or LLM service. We identify a key system challenge in this paradigm: current LLMs lack the elasticity to serve apps' requests that have diversified Service-Level Objectives (SLOs) on inference latency. To tackle this, we present `ElastiLM`, an on-device LLM service that elasticizes both the model and the prompt dimension of a full LLMaaS. It incorporates (1) a one-shot neuron-reordering method, which leverages the intrinsic permutation consistency in transformer models to generate high-quality elasticized sub-models with minimal runtime switching overhead; (2) a dual-head tiny language model, which efficiently and effectively refines the prompt and orchestrates the elastification between model and prompt. We implement such an elastic on-device LLM service on multiple COTS smartphones, and evaluate `ElastiLM` on both standalone NLP/mobile-agent datasets and end-to-end synthesized traces. On diverse SLOs, `ElastiLM` outperforms 7 strong baselines in (absolute) accuracy by up to 14.83% and 10.45% on average, with less than 1% TTFT switching overhead, on-par memory consumption and less than 100 offline GPU hours.

A.2 Artifact check-list (meta-information)

- **Model: LLaMA-7B, Llama3-8B, Llama3-8B-instruct, Vicuna-7B, Orca-mini-3B**
- **Data set: ARC_E, OBQA, PIQA, SCIQ, Octopus, LlamaTouch**
- **Run-time environment: Ubuntu22, Android**
- **Hardware: Cloud A40 GPU server, MI14 smartphone**
- **Execution: Python and bash scripts; C++ binaries**
- **Metrics: Accuracy under given SLOs**
- **Output: Logs and figures**
- **Experiments: (1) Elasticize the LLM and run end-to-end experiments of `ElastiLM` and its baselines. (2) Deploy the elasticized LLM on mobile devices.**
- **How much disk space required (approximately)?: over 500GB**
- **How much time is needed to prepare workflow (approximately)?: 1 hour**
- **How much time is needed to complete experiments (approximately)?: over 200 hours**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: MIT**
- **Data licenses (if publicly available)?: MIT**
- **Archived (provide DOI)?: Not yet**

A.3 Description

A.3.1 How to access

Our code is publicly available at GitHub[1], and we have packed our code into a docker image, which can be downloaded from DockerHub[2].

A.3.2 Preparation

We recommend you directly using the docker image, or checking the software version details in the docker image and manually installing them.

[1] `https://github.com/yinwangsong/ElastiLM`

[2] `https://hub.docker.com/r/yinwangsong2000/elastilm_ae`

```
docker pull yinwangsong2000/elastilm_ae
```

Then, enter the docker container and go ahead for the following.

A.3.3 Experiments on cloud servers

Install the modified third-party libs.

```
cd transformers/
pip install -e .
cd ../scores/
pip install -e .
cd ../LLMLingua/
pip install -e .
```

Please set your HF_HOME as /data/share.

To calibrate the anchor layers, run

```
python3 ELASTICLLM/Anchor_layers/llama_layers.py
...
python3 ELASTICLLM/Anchor_layers/orca_mini_3b_layers.py
```

Profile the importance of permutation-consistent units and recover each submodel with LoRA.

```
# '02.sh' means identifying the top 20%
# important permutation consistent units.
# '0' means the GPU rank.
bash LLMPruner/scripts/02.sh 0
bash LLMPruner/scripts/03.sh 0
bash LLMPruner/scripts/04.sh 0
...
bash LLMPruner/scripts/09.sh 0
```

Run the following instructions to train the dual heads of the TLM.

```
python3 ELASTICLLM/train_slm/llama/train_mobilebert_scorehead.py
python3 ELASTICLLM/train_slm/llama/train_mobilebert_decisionhead.py
...
python3 ELASTICLLM/train_slm/llama3/train_mobilebert_scorehead.py
python3 ELASTICLLM/train_slm/llama3/train_mobilebert_decisionhead.py
```

Profiling contextual sparsity.

```
bash ELASTICLLM/Contextual_sparsity/run_c4_mlp.sh 0
```

Preparing for LaCo.

```
CUDA_VISIBLE_DEVICES=0 python3 ELASTICLLM/Layer_pruning/LaCo/prune_llama.py
CUDA_VISIBLE_DEVICES=0,1 python3 ELASTICLLM/Layer_pruning/LaCo/prune_llama3.py
..
CUDA_VISIBLE_DEVICES=0 python3 ELASTICLLM/Layer_pruning/LaCo/prune_orcamini.py
```

Preparing for AttnDrop/ShortGPT.

```
# AttnDrop
python3 ELASTICLLM/Layer_pruning/AttnDrop/prune_llama.py
...
python3 ELASTICLLM/Layer_pruning/AttnDrop/prune_llama3.py

# ShortGPT
python3 ELASTICLLM/Layer_pruning/ShortGPT/prune_llama.py
...
python3 ELASTICLLM/Layer_pruning/ShortGPT/prune_llama3.py
```

Run each dataset by

```
bash ELASTICLLM/scripts/ARC_E.sh 0
...
```

. You will see the results under ELASTICLLM/scripts/res/[model]_[dataset].txt

Synthesize the traces.

```
python3 ELASTICLLM/e2e/traces/generate_traces.py
```

You will see trace_0.json, trace_0.25.json and trace_-0.25.json under the same directory. The number (e.g., 0) means the skewness of the trace.

Then, run the end-to-end experiments.

```
# 0 1 2 3: GPU ranks; 1: model id
bash ELASTICLLM/e2e/scripts/run_e2e.sh 0 1 2 3 1
bash ELASTICLLM/e2e/scripts/run_e2e.sh 0 1 2 3 2
...
bash ELASTICLLM/e2e/scripts/run_e2e.sh 0 1 2 3 5
```

You will see the results in
ELASTICLLM/e2e/scripts/res/res_[model].txt.

A.3.4 Experiments on smartphones

Compiling.

```
cd deployment/mllm/scripts
export $ANDROID_NDK=/path/to/your/NDK
bash ./build_android.sh
```

The compiled binary file demo_elastic_llama_lora will be located in deplyment/mllm/bin-arm/.

We pre-uploaded an elasticized oraca_mini_3b model with the corresponding fine-tuned LoRA weights and the tiny language model on Google Drive[3].

Please download them and put them in deployment/mllm/models/ by

```
gdown <file-id>
```

Run the demo by

```
cd deployment/mllm/scripts
./run\_elastic\_llama\_lora.sh
```

A.4 Notes

A.4.1 The results

Draw the figures by ELASTICLLM/Drawing/draw.ipynb.

Reproduce the demo video by Termux.

[3] `https://drive.google.com/drive/folders/1RAKabZHfubIXmpzFMqDaGv7ki5SjXjSi?usp=sharing`