Project Milestone Report website: https://rickzx.github.io/618-project/

0. Summary of Work Completed

After submitting the proposal, we have been relatively on schedule. We have completed and verified the implementation of GPT 2 in MLC-LLM, allowing the model to be optimized by TVM, specifically with the operator-level optimizer DLight that we focus on. After specifying the algorithm to be optimized (i.e. GPT-2 inference), we ran some preliminary benchmarks to compare the performance across different platforms. We also ran benchmarks to determine how each rule of DLight affects the performance of GPT-2 inference.

1. Implementing GPT-2 in MLC LLM

Since the goal of our project is to optimize GPT-2's inference performance with TVM, we need to first define the algorithm to be optimized, as mentioned in lectures about domain-specific programming languages. In this case, we leverage the framework MLC LLM to define our workload, allowing us to subsequently use TVM to optimize it.

We have implemented GPT 2 in MLC LLM in this GitHub pull request: https://github.com/mlc-ai/mlc-llm/pull/1314. We mainly follow the GPT-2 architecture implemented in huggingface, but conforming to the semantics accepted by TVM.

2. Preliminary Result: Benchmarks across platforms

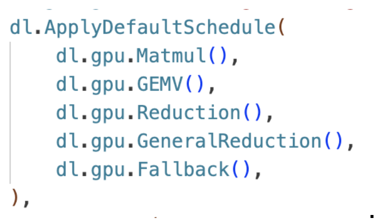
After implementing GPT 2, we ran its inference with TVM on the CUDA backend (with a single RTX 4090), on a Macbook Pro (Intel), and on the WebGPU backend (on the same Macbook Pro). We use the number of tokens per second to describe the performance. Specifically, we prefill 322 tokens (the entire Gettysburg Address) and let the model generate 64 tokens autoregressively.

	Prefill (tokens/sec)	Decode (tokens/sec)
RTX 4090	4419.0	102.9
MacBook Pro (Intel)	28.9	1.1
WebGPU (same MacBook Pro)	10.96	0.81

From the result, one thing to notice is that, despite leveraging the same hardware, we are receiving worse results on the WebGPU backend. This makes one of the goal for us to explore better operator-level optimization for the WebGPU backend to utilize its hardware better, hopefully closing the gap. Note that we mentioned in the proposal that we will also be testing on the ROCm backend (AMD GPU) – due to hardware issues, we will supplement the result in the final submission.

3. Preliminary Result: Effects of each operator-level optimization

Recall that, as mentioned in the proposal, we focus on operator-level optimization within TVM, and such optimization is done in a sub-framework called DLight. The usage of DLight can be illustrated as below:



```
dl.ApplyDefaultSchedule(
    dl.gpu.Matmul(),
    dl.gpu.GEMV(),
    dl.gpu.Reduction(),
    dl.gpu.GeneralReduction(),
    dl.gpu.Fallback(),
),
```

After implementing GPT-2 in MLC LLM, another benchmarking we did was to determine how much effect each rule has on the performance. Below are the results after running the same experiment on RTX 4090 (prefilling 322 tokens, generating 64 tokens autoregressively):

Schedule	Prefill (tokens/sec)	Prefill Speedup	Decode (tokens/sec)	Decode Speedup
Baseline	70.0	1x	3.5	1x
Matmul	3321.4	47.4x	3.5	1x
GEMV	70.1	1x	53.5	15.3x
Reduction	70.1	1x	60.7	17.3x
Full	4419.0	63.1x	102.9	29.4x

“Baseline” denotes without using any optimization, and “full” denotes using all optimization. Note that we consider `Reduction()` and `GeneralReduction()` as a single rule; we also note that due to the current implementation, we need to include `Fallback()` in all schedules, hence considering it as part of the baseline.

We observe that matrix multiplication contributes the most speedup for the prefill stage, which makes sense because during prefill, most operations are matrix-matrix multiplication. This is due to how we have multiple values in the Q matrix. Meanwhile, GEMV, which optimizes matrix-vector multiplication, contributes to the speedup during decoding. Similarly, this makes sense because we only have a single query token in the Q matrix during the decoding stage.

4. Reflection upon Goals and Deliverables

We believe that we are on track to achieving the goals set in the project proposal. Since we are done with specifying the workload in the domain-specific language, we now focus on optimizing it. Specifically, the gap includes finding the hyperparameters/transformations for us to tweak each DLight rule, allowing us to bridge the performance gap between WebGPU and native MacBook. Besides, we also need to have a deeper understanding of how each rule in DLight affects the intermediate representation of the program. Here is a list of goals:

- Understand how each DLight rule affects the program (not just the performance)
- Locate the hyperparameters/transformations that allow us to optimize the scheduler
- Determine the characteristics of WebGPU to try to bridge the gap between WebGPU and native backend

5. Plan for the poster session

For the final presentation, we plan to divide our poster into the following sections:

1. An overview of how large language model (LLM) inference works (with figures)
2. An overview of the workflow of optimizing an LLM using TVM and MLC-LLM (with code snippets and figures)
3. Benchmark results on various platforms (with tables and graphs)
4. How each DLight rule affects the program and speedup (with code snippets and tables)
5. How we tuned the DLight scheduler to bridge the gap between WebGPU and the native backend (with tables to illustrate results)

6. Potential issues

One potential issue/concern we have is whether we can bridge the gap between WebGPU and the native backend. It is possible that the shader language of WebGPU does not have the expressibility to achieve the same performance as the native backend, or its primitives simply have more overhead. In that case, we will probably discuss the findings and post how each degree of freedom in the DLight scheduler affects the program and its performance. Thus, closing the gap would be considered a “nice to have” goal.

7. Updated schedule

For each task, Rick and Charlie will work together.

Week 1 (11/13 - 11/19)	☑ ~~Brainstorm project scope and submit proposal~~ ☑ ~~Start implementing GPT 2 in MLC-LLM~~
Week 2 (11/20 - 11/26)	☑ ~~Finish and verify GPT 2 implementation in MLC-LLM~~
Week 3 (11/27 - 12/3)	☑ ~~Benchmark speedup for different backends with default scheduler~~ ☑ ~~Analyze the effects of each scheduling rule for each rule on CUDA~~ ☑ ~~Complete milestone report (due 12/2)~~
12/7 - 12/8	☐ Learn how each scheduler rule transforms the program
12/9 - 12/10	☐ Learn mechanisms of tuning the DLight scheduler (through machine learning, grid search, etc.); locate the hyperparameters
12/11 - 12/12	☐ Tune scheduler to achieve a better speedup for WebGPU
12/13 - 12/14	☐ Complete the final report and the poster (due 12/14)