Accelerating GPT-2 Inference with TVM

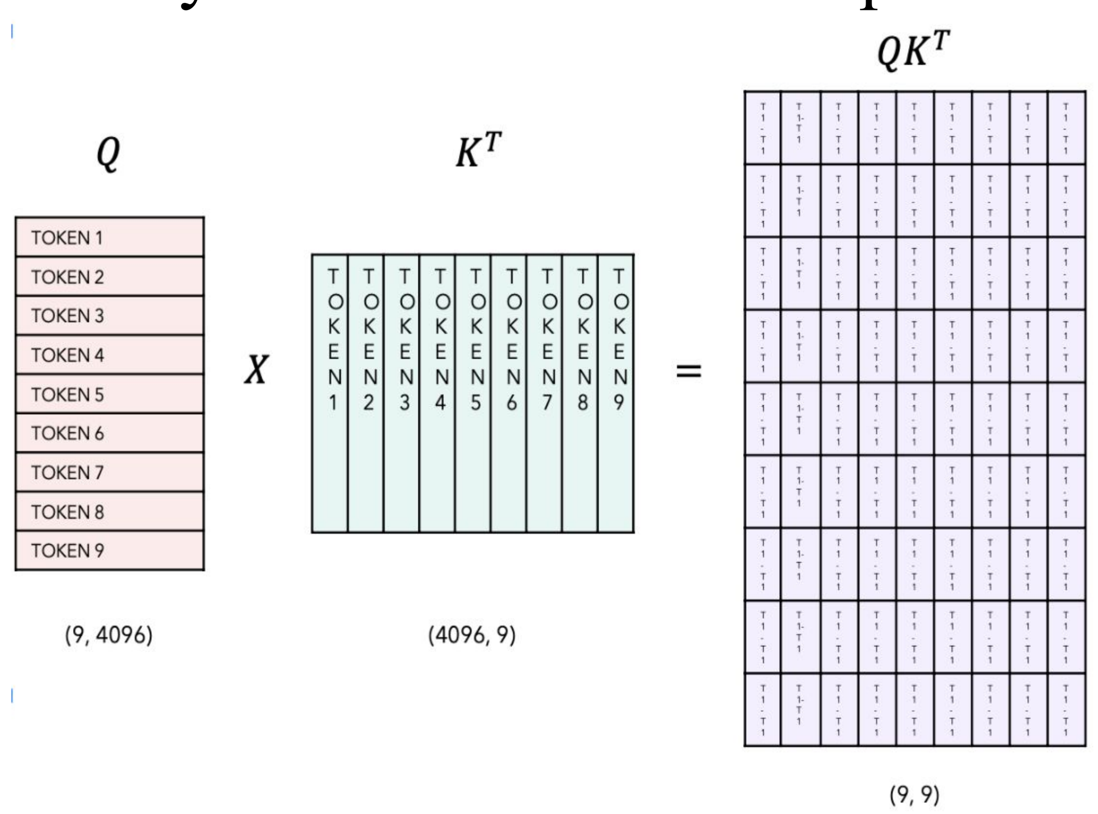
Rick Zhou, Charlie Ruan
{xzhou2, cfruan}@andrew.cmu.edu
Carnegie Mellon University Computer Science Department

Autoregressive Inference

Under this setting, a decoder-only transformer's computation is mainly **matrix-matrix**, and **matrix-vector** multiplications.

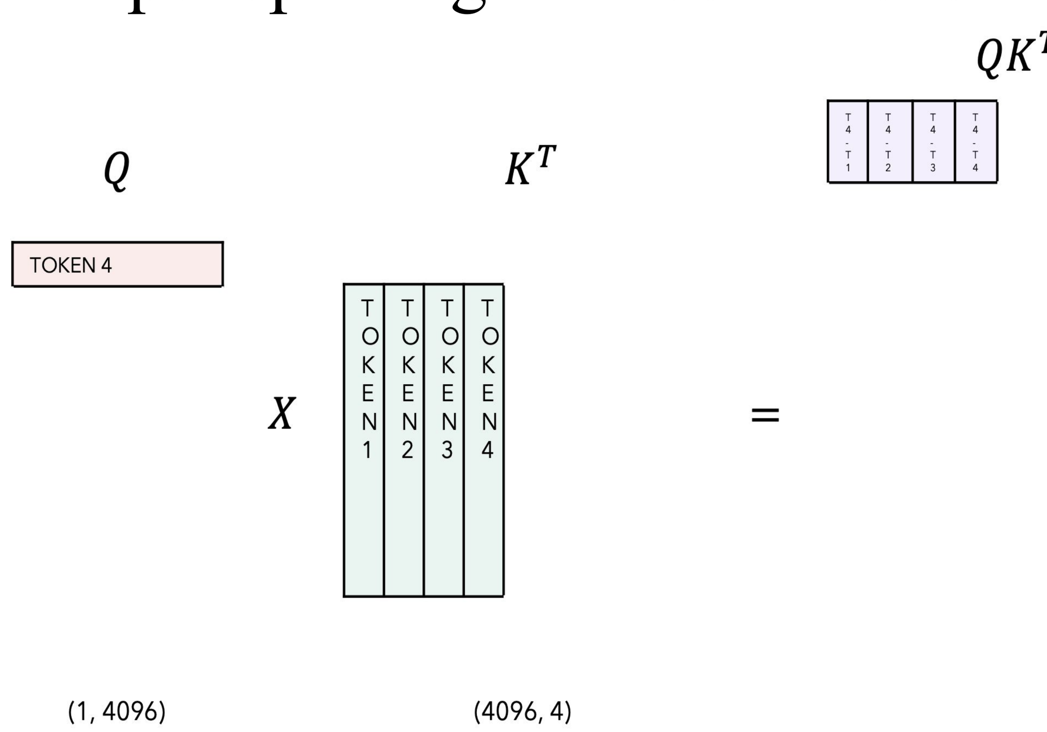
Phase 1: Prefill

Model's input (prompt) has multiple tokens, hence making the workload mainly **matrix-matrix** multiplication



Phase 2: Decode

Model's input is generated token from previous step, hence a single token with **matrix-vector** multiplications. Keeps repeating until conditions met.



(Visualizations from [3])

Optimizing GPT-2 Inference in TVM

Step 1. Define the workload in domain specific language w/ symbolic shapes.

```python
class GPT2LMHeadModel(nn.Module):
    def __init__(self, config: GPT2Config):
        self.transformer = GPT2Model(config)
        self.lm_head = nn.Linear(config.n_embd, config.vocab_size, bias=False)
        self.vocab_size = config.vocab_size
        self.dtype = "float32"

    def forward(self, inputs: Tensor, total_seq_len: tir.Var, attention_mask):…

    def prefill(self, inputs: Tensor, total_seq_len: tir.Var):…

    def decode(self, inputs: Tensor, total_seq_len: tir.Var):
        batch_size, seq_len = inputs.shape
        attention_mask = op.full(…
        return self.forward(inputs, total_seq_len, attention_mask)

    def softmax_with_temperature(self, logits: Tensor, temperature: Tensor):
        return op.softmax(logits / temperature, axis=-1)
```

Step 2. Optimize with TVM [1]. Among the steps: **operator-level optimizations**.

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

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

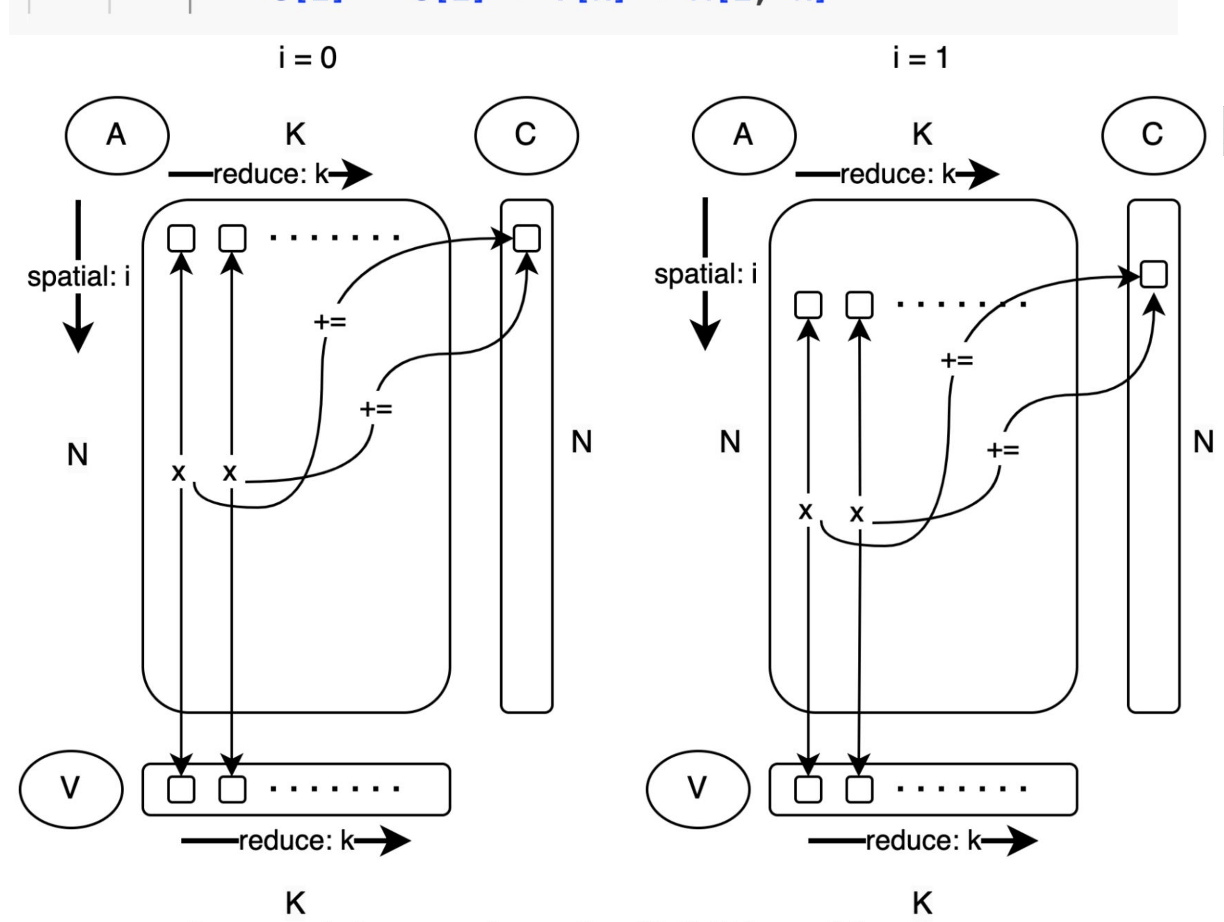
Table: Performance gain obtained from each operator-level optimization

Matmul optimizes **matrix-matrix** multiplication → gives speedup to **prefill**.
GEMV optimizes **matrix-vector** multiplication → gives speedup to **decode**.
Reduction optimizes ops like **softmax / LayerNorm** → gives speedup to **decode**.

Case Study: How TVM Optimizes GEMV

Workload of GEMV is essentially (N, K) @ (K) → (N)

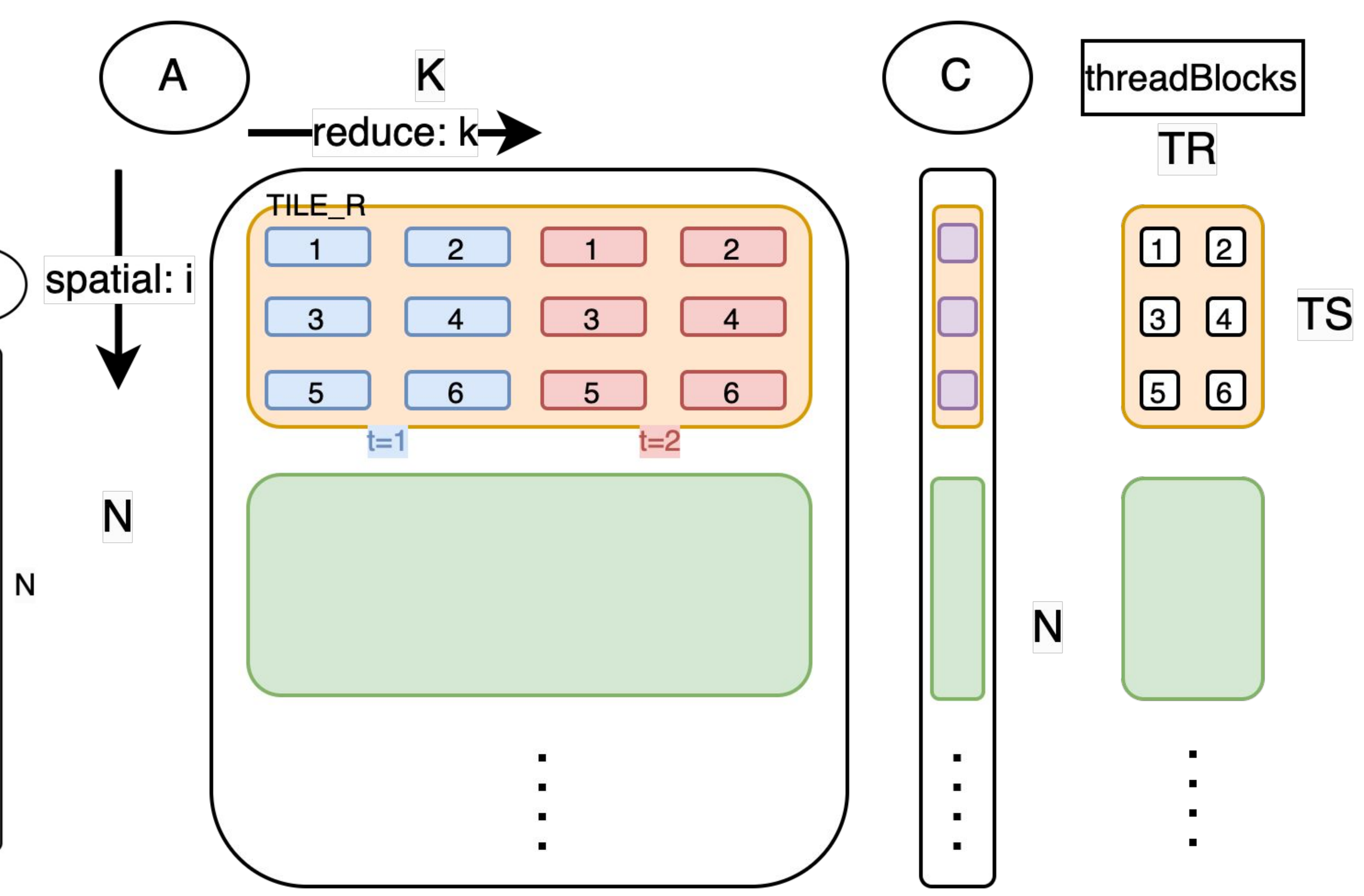
```python
def NK_gemv(A: np.ndarray, V: np.ndarray, C: np.ndarray):
    # A: (N, K), V: (K), C: (N)
    C = np.empty(V)
    for i in range(N):  # spatial axis
        for k in range(K):  # reduce axis
            C[i] = C[i] + V[k] * A[i, k]
```



Sequential execution of a GEMV workload.

Parallelizing GEMV:

1. N rows → groups of TS rows, 1 group per block
2. Each block spawns TS x TR threads
3. Each thread works on TILE_S x TILE_R a time



K=8, TILE_R=2, TILE_S=1, TR=2, TS=3

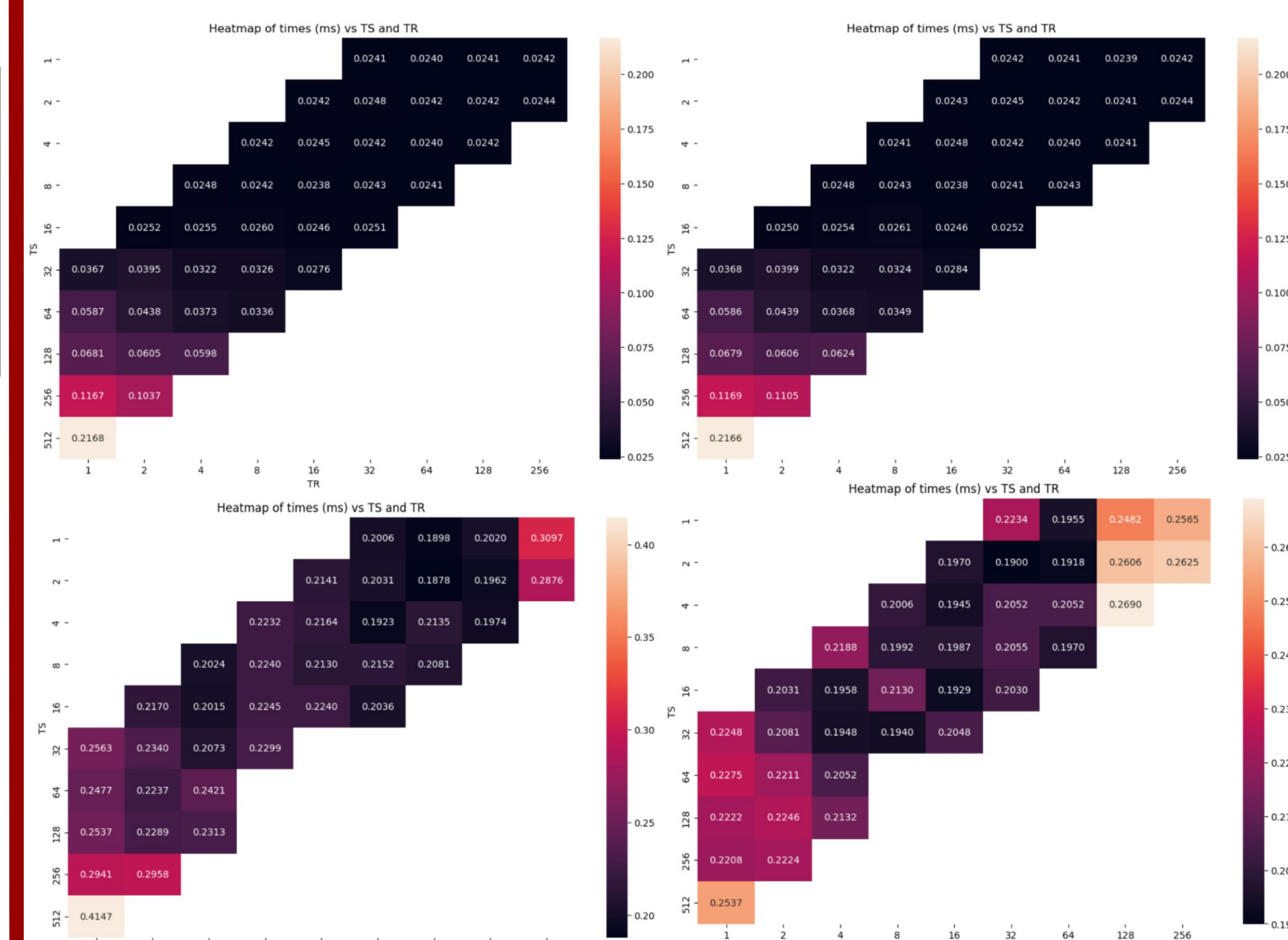
Tuning GEMV to Optimize Decode

Search space:

Make TILE_R multiples of 8; consider two schedules:

Sch. 1: Load vectors over N, compute over K
Set TILE_R=8, search over TILE_S

Sch. 2: Load vectors over K, computer over N
Set TILE_S=1, search over TILE_R



How TS and TR affect GEMV given a set pair of TILE_S and TILE_R. Top figures are from RTX 4090; bottom from M2 Ultra. Left figures are from schedule 1; right figures from schedule 2.

	Decode (tokens/sec)
RTX 4090 - Untuned	338.614
RTX 4090 - Tuned	343.402
M2 Ultra - Untuned	117.239
M2 Ultra - Tuned	117.477

Comparison of GPT-2 inference with tuned/untuned GEMV.

References

[1] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, Tvm: An automated end-to-end optimizing compiler for deep learning (2018), arXiv:1802.04799 [cs.LG].
[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI blog 1, 9 (2019)
[3] MLC Team, MLC-LLM (2023) https://github.com/mlc-ai/mlc-llm
[4] U Jamil, LLaMA Explained (2023) https://github.com/hkproj/pytorch-llama-notes
[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. K¨opf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, Pytorch: An imperative style, high-performance deep learning library (2019), arXiv:1912.01703 [cs.LG].
[6] J. Thickstun, D. Hall, C. Donahue, and P. Liang, Anticipatory music transformer (2023), arXiv:2306.08620 [cs.SD].
[7] MLC Team, Dlight-Bench (2023) https://github.com/mlc-ai/dlight-bench