AutoGTCO: Graph and Tensor Co-Optimize for Image Recognition with Transformers on GPU

Yang Bai[1], Xufeng Yao[2], Qi Sun[1], Bei Yu[1]

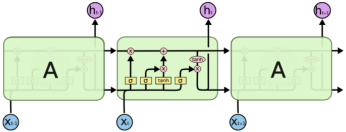
[1]The Chinese University of Hong Kong
[2]SmartMore
{ybai,byu}@cse.cuhk.edu.hk

Nov. 1, 2021

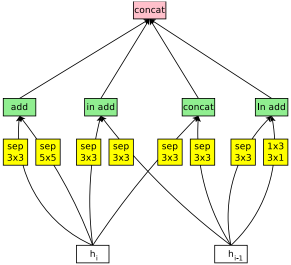
Outline

Introduction

- Deep Learning Models

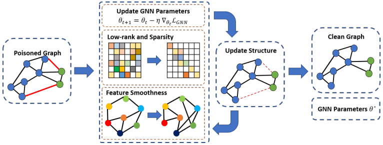


LSTM



Normal Cell

NasNet



graph neural network

- Modern Accelerators

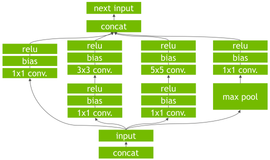
NVIDIA GPU

AMD GPU

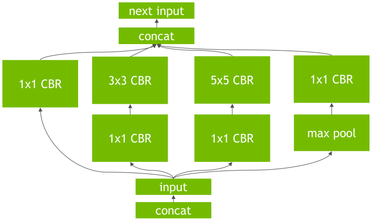
Google TPU

Graphcore IPU

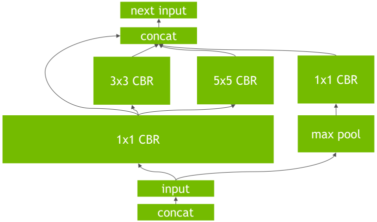
- Fuses kernels – Vertically (Conv, BN, ReLU) and Horizontally (Reuse Inputs)



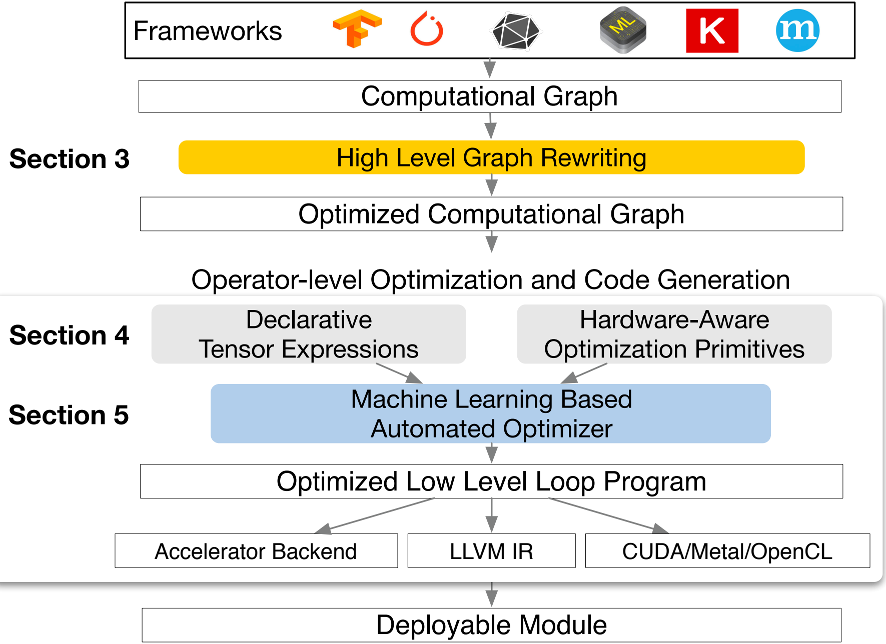
Original Compute Graph



Vertical Fusion



Horizontal Fusion


Frameworks
Computational Graph
Section 3
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Section 4
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Section 5
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module

Related Work and Background

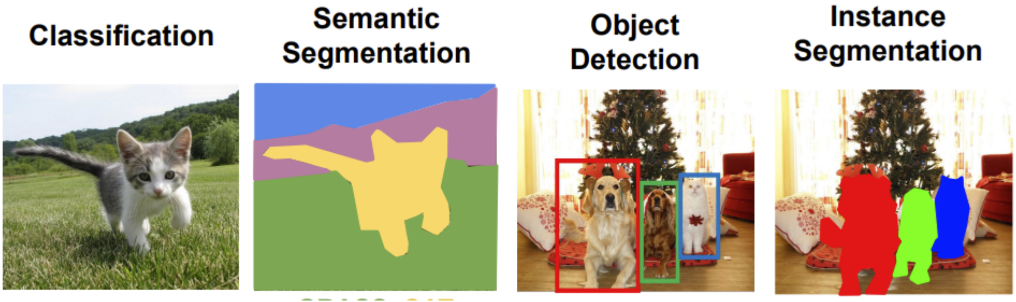
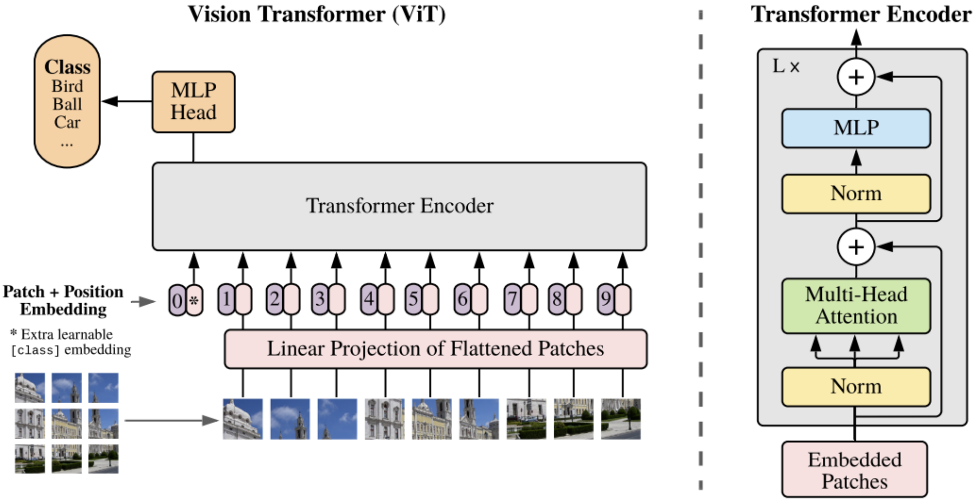
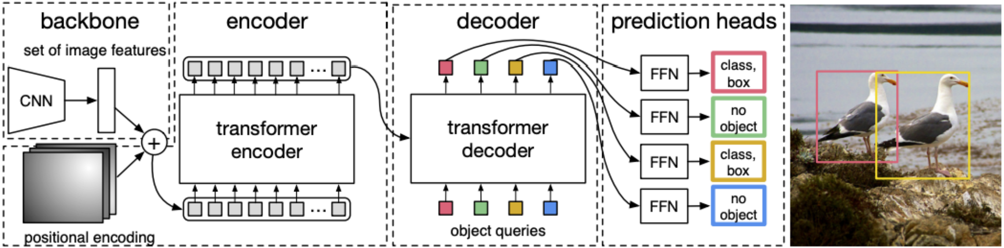


Image Recognition in Computer Vision Tasks (CS231n)

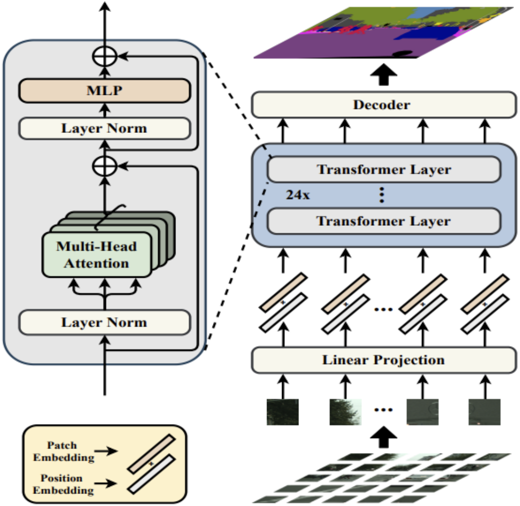


The architecture of Vision-Trasnformer (ViT, ICLR 2021)

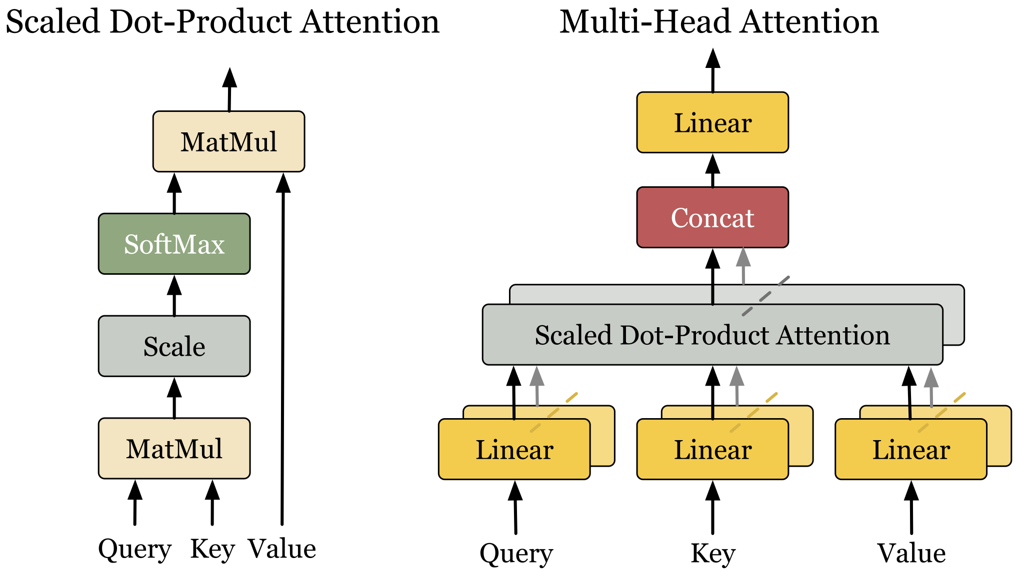


The architecture of DEtection-TRansformer (DETR, ECCV 2020)

Semanic Segmentation

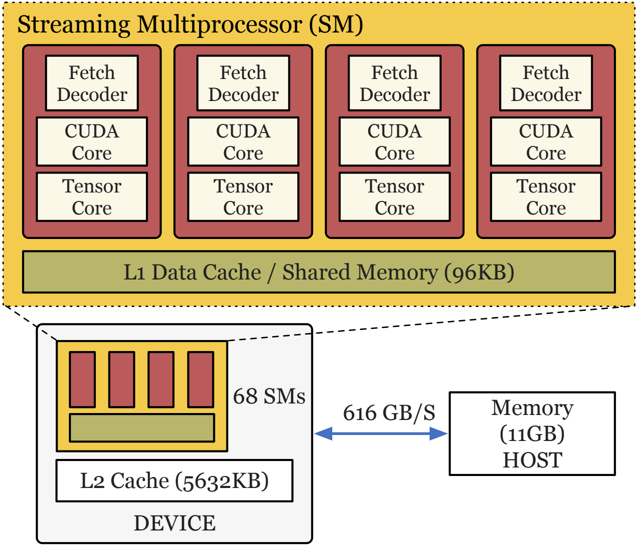


The architecture of SEgmentation-TRansformer (SETR, CVPR 2021)



Scaled Dot-Product and Multi-Head Attention (MHA).

Hierarchy of 2080 Ti GPUs



A streaming multiprocessor and the memory architecture of GeForce RTX 2080 Ti GPU.

Problem Formulation

Our Definition

1. Computate Graph:
   A transformer model is defined by a computation graph $G = (V, E)$, where V is the set of vertices and E is the edge set. Each vertex can represent an operator such as GEMM and softmax operation in the computation graph. Each edge $(u, v) \in E$ is to describe the dependencies between node u and v.
2. Operator Pattern
   - injective
   - reduction
   - complex-out-fusable
   - element-wise
   - opaque
3. Fusion Strategy and Schedule:
   We define a schedule S of a computation graph G as follow:

$$S = \{(V_1, F_1), (V_2, F_2), \ldots, (V_k, F_k)\}, \tag{1}$$

where V_i represents a group of operators in the i-th phase and F_i is a pair to describe the fusion relationship between two nodes. Finally, computation graph can be executed under the schedule S from the first phase (V_1, F_1) to the last phase (V_k, F_k) consecutively.

Our Problem

- Given a computation graph G and fusion schedule S on GPU, our goal is to search for a schedule S^*:

$$S^* = \operatorname*{argmin}_S \; Cost(G, S), \tag{2}$$

  where $Cost$ is the latency of executing G according to the schedule S.

- Multi-Head Attention Function:

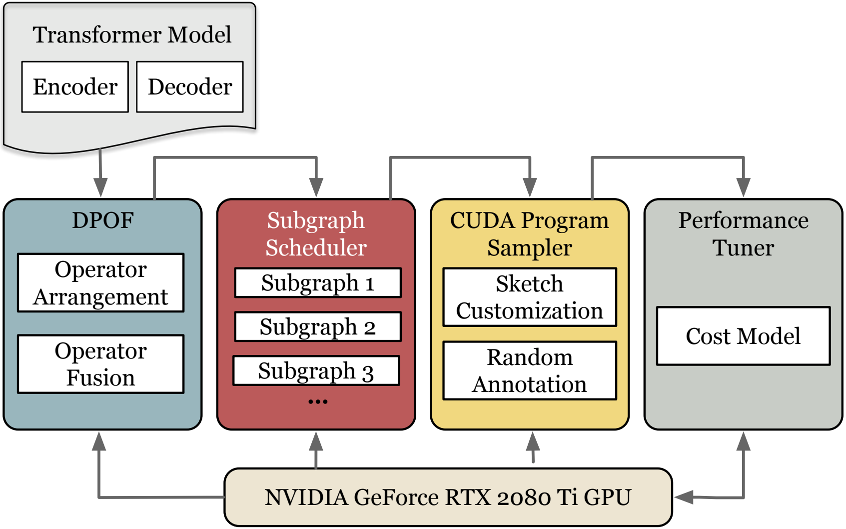
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \tag{4}$$

- Transformers have lots of softmax operators in Multi-Head Attention and can be fused with batch matrix multiplication operators

Overview of our system

The Proposed System

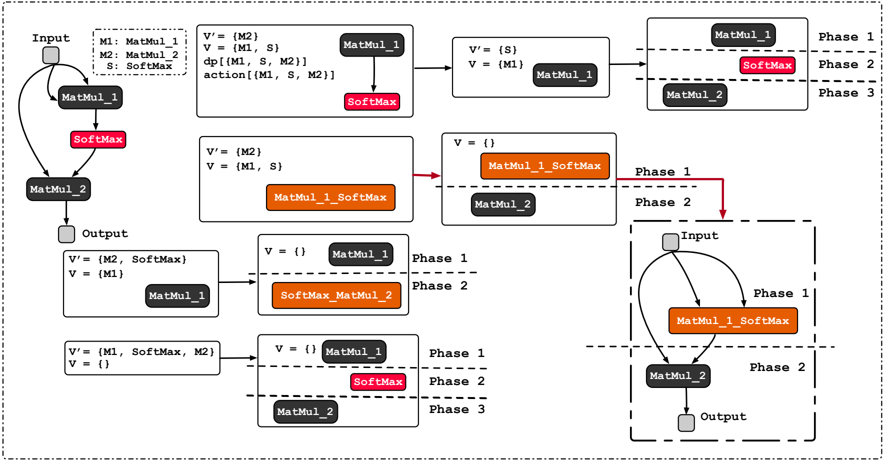


The arrows show the flow of the optimized subgraphs from transformer model and tensor programs generation on GPU platform.

Our tensor generation framework is composed of four important modules

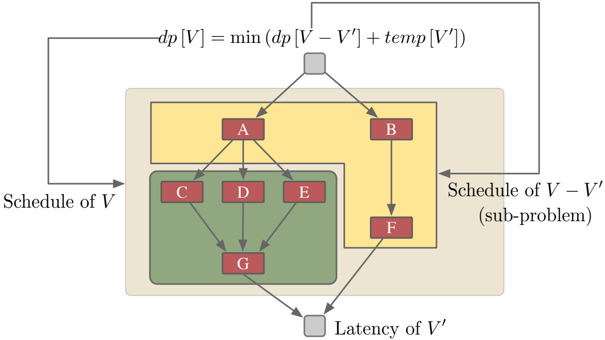
1. Dynamic Programming-based Operator Fusion (DPOF)
2. Subgraph Scheduler
3. CUDA Program sampler
4. Performance Tuner

- Input: A transformer-based model without any operator fusion
- Output: Operators with new tags
- Function: A DPOF that finds an optimized operator fusion schedule for the transformer model



The structure of DPOF

1. Operator Arragenement
   - topological sort to get operators
   - queue to store operators
   - compute-type, no placeholder-type operators
   - size of queue = maximum number of queue
2. Operator Fusion



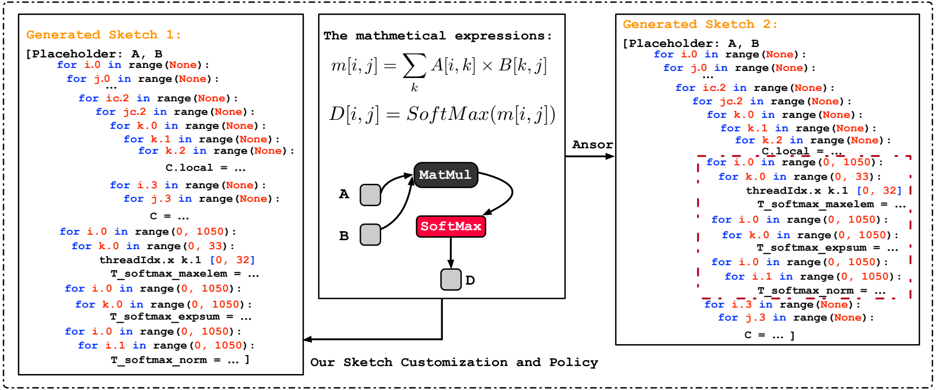
Subgraph Scheduler

- Input: A transformer-based model with operator fusion
- Output: Lots of subgraphs decomposed by compute-intensive operators
- Function: A subgraph scheduler that allocates time resources for optimizing multiple subgraphs generated by the DPOF

CUDA Program sampler

- Input: Subgraph with fused operators
- Output: CUDA kernel code for these operators
- Function: A program sampler that delineates a large search space and randomly samples various programs from it

1. Sketch Generation
2. Annotation



Performance Tuner

- Input: Sampled CUDA kernel codes
- Output: The performance of the generated code
- Function: A performance tuner that trains a cost model to measure the performance of sampled tensor programs

Evaluation Results

Experimental Setup

1. Image Recognition Models
   - DETR for Object Detection
   - SETR for Semantic Segmentation
   - ViT for Image Classification
2. WorkFlow
   - TensorRT: PyTorch → ONNX → ONNX-Simplifier →TensorRT Engine
   - AutoGTCO: PyTorch → TorchScript → Relay → Code Generation
3. WorkLoads
   - Batch Size=1

Architecture of the Model and Configurations

model	ec	dc	width	mlp-dim	nh	input shape	patch	mha input	encoder input	decoder input	Params
DETR-ResNet50-E3	3	6	256	2048	8	[1,3,800,1333]	N/A	query[1050,1,256] key[1050,1,256] value[1050,1,256]	src[1050,1,256]	tgt[100,1,256] mem[1050,1,256]	37.40M
DETR-ResNet50-E6	6	6	256	2048	8	[1,3,800,1333]	N/A	query[1050,1,256] key[1050,1,256] value[1050,1,256]	src[1050,1,256]	tgt[100,1,256] mem[1050,1,256]	41.30M
DETR-ResNet50-E12	12	6	256	2048	8	[1,3,800,1333]	N/A	query[1050,1,256] key[1050,1,256] value[1050,1,256]	src[1050,1,256]	tgt[100,1,256] mem[1050,1,256]	49.20M
SETR-Naive-Base	12	1	768	4096	12	[1,3,384,384]	16	query[576,1,768] key[576,1,768] value[576,1,768]	src[576,1,768]	tgt[576,1,768]	87.69M
SETR-Naive	24	1	1024	4096	16	[1,3,384,384]	16	query[576,1,1024] key[576,1,1024] value[576,1,1024]	src[576,1,1024]	tgt[576,1,1024]	305.67M
SETR-PUP	24	1	1024	4096	16	[1,3,384,384]	16	query[576,1,1024] key[576,1,1024] value[576,1,1024]	src[576,1,1024]	tgt[576,1,1024]	310.57M
ViT-Base-16	12	0	768	3072	12	[1,3,224,224]	16	query[197,1,768] key[197,1,768] value[197,1,768]	src[197,1,768]	N/A	86.00M
ViT-Large-16	24	0	1024	4096	16	[1,3,224,224]	16	query[197,1,1024] key[197,1,1024] value[197,1,1024]	src[197,1,1024]	N/A	307.00M
ViT-Huge-14	32	0	1280	5120	16	[1,3,224,224]	14	query[257,1,1280] key[257,1,1280] value[257,1,1280]	src[257,1,1280]	N/A	632.00M

Experimental Results on E2E Performance

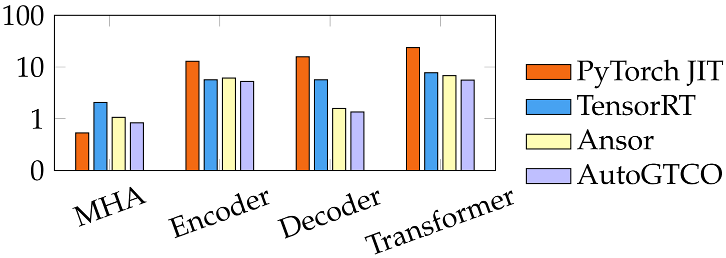
- Baseline: PyTorch JIT, TVM-cuDNN, TensorRT, Ansor
- Pytorch 1.7.1, cuDNN V7.6.5, CUDA 10.0, TensorRT V7.0.0.11, TVM 0.8

Table: End-to-End Exeuction Performance on the Benchmark (ms)

	PyTorch JIT	TVM-CUDA	TVM-cuDNN	TensorRT	Ansor	AutoGTCO
DETR-ResNet50-E3	18.62	54.73	54.43	6.97	5.85	**5.32**
DETR-ResNet50-E6	23.67	93.59	88.25	7.73	6.78	**5.60**
DETR-ResNet50-E12	33.01	171.96	157.97	15.79	14.29	**13.18**
SETR-Naive	68.26	753.25	742.21	33.71	34.22	**28.65**
SETR-Naive-Base	31.06	186.13	187.39	16.97	15.44	**14.21**
SETR-PUP	37.62	199.42	189.21	18.61	17.89	**16.01**
ViT-Base-16	24.92	91.86	96.31	**5.87**	8.57	8.43
ViT-Large-16	52.96	329.74	334.38	18.45	18.99	**18.41**
ViT-Huge-14	76.07	846.87	846.27	34.14	32.53	**29.89**

- Compared with TensorRT: 1.01-1.38× speedup
- Compared with Ansor: 1.01-1.21× speedup

- Baseline: MHA, Encoder, and Decoder of DETR-ResNet-50-E6



The y-axis is the throughput based log 10 and then plus 1.

Compared with:

- PyTorch JIT: 2.47× on Encoder and 11.67× on Decoder
- TensorRT: 2.47× speedup on MHA, 1.08× on Encoder, and 4.19× on Decoder
- Ansor: 1.29× on MHA, 1.17× on Encoder, and 1.17× on Decoder

Conclusions

Conclusions

- Graph-Level optimizaiton designed by human experts miss the potential performance.
- Graph and Tensor Co-Optimize (AutoGTCO):
  - A novel dynamic programming algorithm to explore operator fusion strategies.
  - new sketch generation rules and a search policy for CUDA kernel generation.
- Key Results: 1.01 - 1.38$\times$ speedup on diverse Transformer-based vision models.

THANK YOU!