# TAP: Efficient Derivation of Tensor Parallel Plans for Large Neural Networks

NUS National University of Singapore

Ziji Shi[1,2], Le Jiang[2], Jie Zhang[2], Xianyan Jia[2], Yong Li[2], Chencan Wu[2], Jialin Li[1], Wei Lin[2]

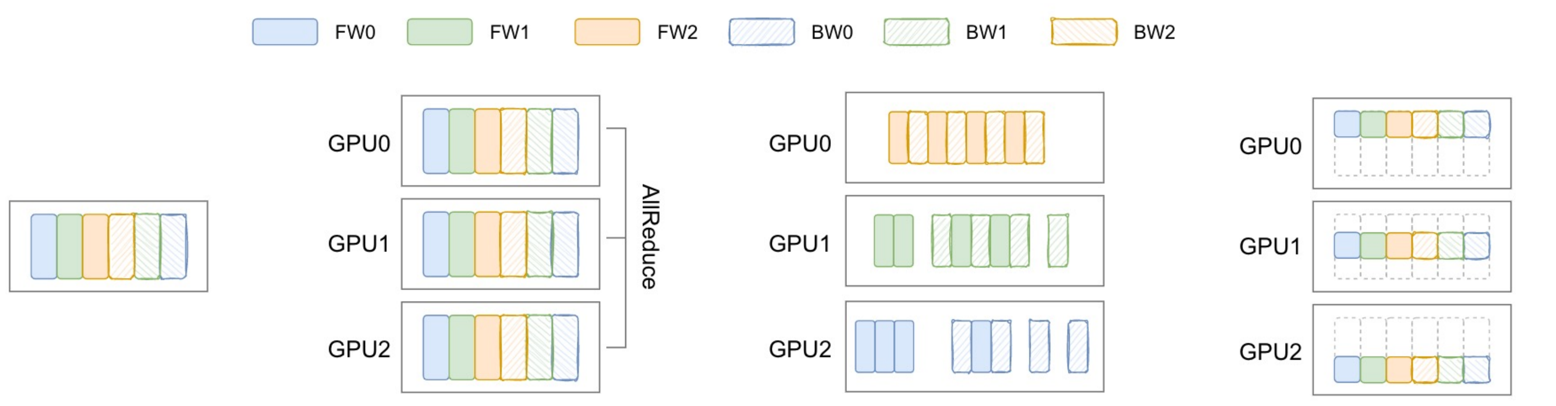
[1]National University of Singapore, [2]Alibaba Group

Alibaba Cloud

## Challenge: Memory Wall and Tensor Parallelism

### a. Memory wall for large neural networks training

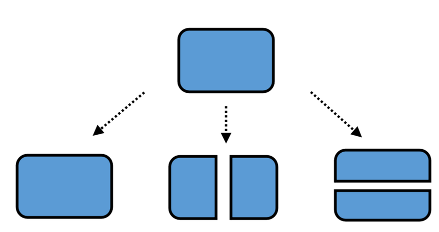
- In the last decade, the model size has increased ~240X every two years, while the GPU memory has only doubled within the same amount of time.
- The memory wall problem has drawn much attention on model parallelism, where the model weight are sharded.




(a) Local Model (b) Data Parallelism (c) Pipeline Parallelism (d) Tensor Model Parallelism

### b. Tensor parallelism

- Split the tensor and distribute on different devices
- A more general approach for training large models
  - When a gigantic layer cannot be fitted into accelerator memory
  - When pipeline parallelism cannot work well due to imbalanced pipeline
- Problem
  - The size of decision space is too large
    - Each 2D tensor has 3 possible choices, thus brute force method is $O(3^N)$



    - A neural network can have thousands to millions of tensors, taking several months to find the best plan
  - Expert annotation can be difficult, stiff, and error-prone
    - Requires deep understanding on both system and neural network
    - Annotations need to be updated when system/model changes
    - Incorrect annotations may result in training halt or even failure

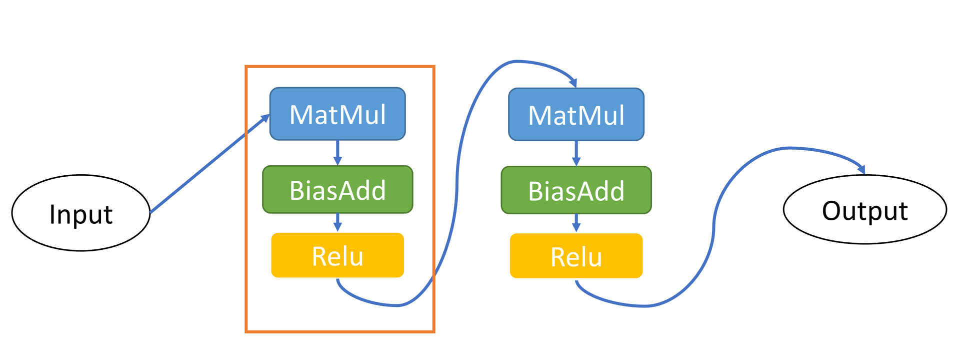
## Approach: Tensor Auto Parallelism (TAP)

### a. Related work

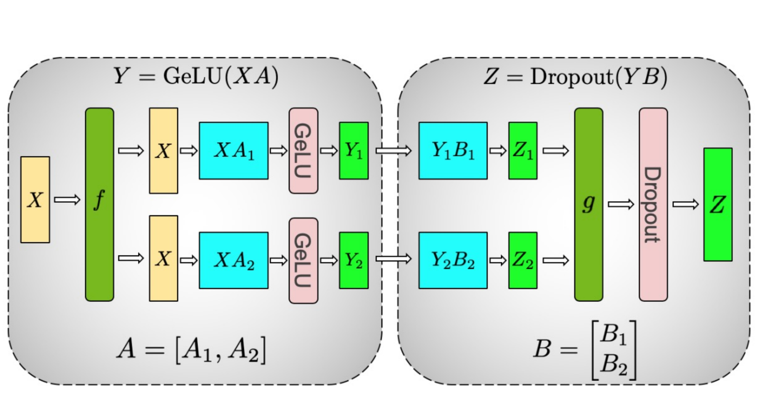
- Expert-annotation driven
  - MeshTensorFlow(NeurIPS'18), GSPMD (2021), Whale (ATC'22)…
- Automatic Parallelism
  - FlexFlow(ICML'18), Tofu(EuroSys'18), Unity(OSDI'22), Alpa(OSDI'22)…

### b. Key observations

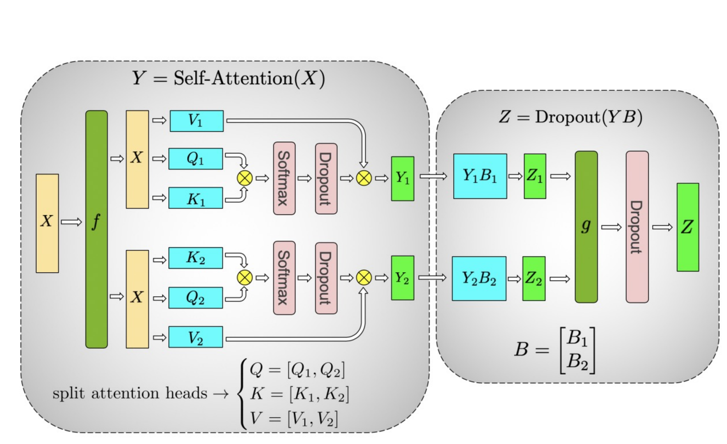
- A neural network can be represented as a directed acyclic graph, within which only contains a limited set of unique subgraphs.
  - Layers (eg. dense, self-attention)
  - Composite operators (eg. softmax)




- The parallel strategy for similar layers are similar
  - For instance, Megatron-LM adopts a similar strategy for the Transformer layers



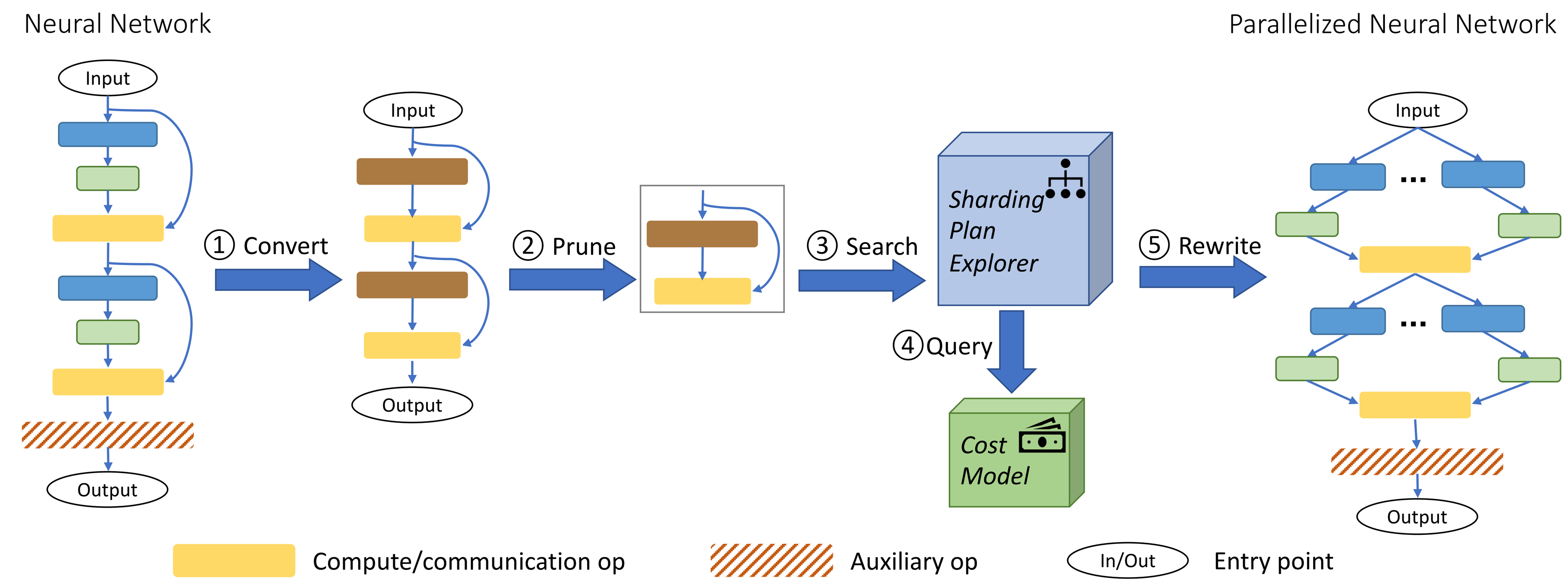

(a) MLP




(b) Self-Attention

Credit: *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism* (2019)

## Design Overview




### a. Workflow

- Convert the original graph to a coarser graph representation
  - Group operators serving a similar functionality
  - Blue and yellow nodes were fused into a brown node
- Prune: Discover unique subgraphs by gradually expanding the smaller subgraphs

**Algorithm 1** Graph Pruning

```
procedure PRUNEGRAPH(modelDef, minDuplicate)
    nodeTree ← ∅
    maxDepth ← modelDef.depth
    for all depth ∈ maxDepth ··· 1 do
        nodeTree[depth] ← longestCommonPrefix(modelDef.nodes.name)
        opCount = findSimilarBlk(nodeTree[depth])
        if opCount ≥ minDuplicate then
            subgraphs.append(nodeTree[depth])
        else
            break
        end if
    end for
    return subgraphs
end procedure
```

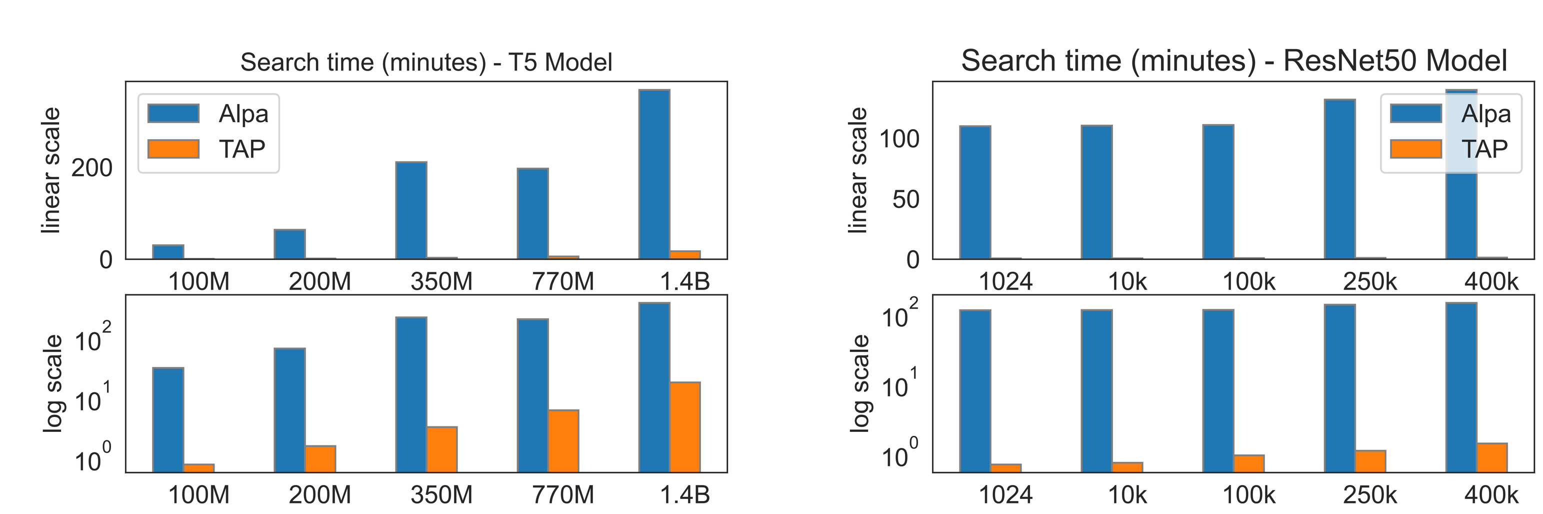
- Search within the subgraphs by enumerating all possible tensor parallel plans
- Query the analytical cost model for all plans and find the one that minimize communication
- Validate and rewrite the plan into original graph format

### b. Usage

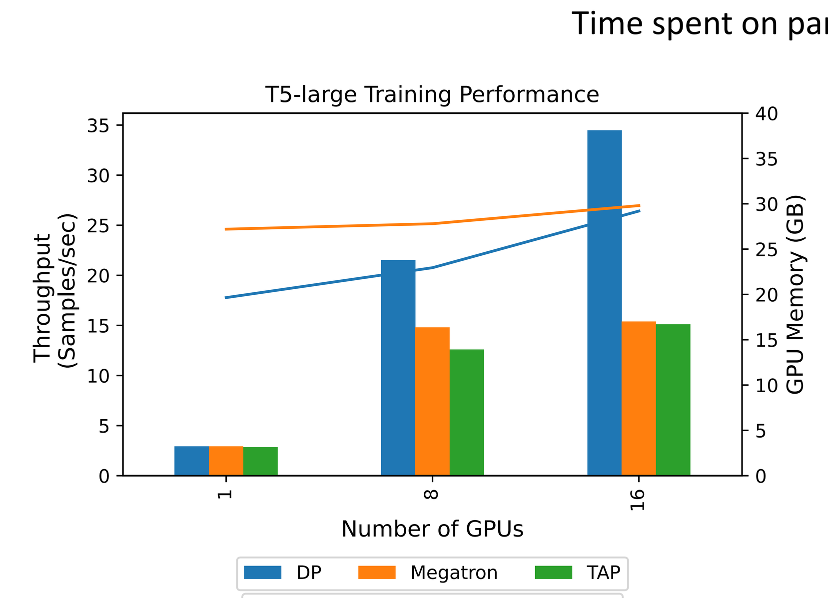
1. Example with TAP on 2 workers each with 8 GPUs

```
import tensor_auto_parallel as tap
mesh = [2, 8]
tap.auto_parallel(tap.split(mesh))
model_def()
```

## Evaluations




Time spent on parallel plan derivation.




Performance comparison with expert-designed Megatron-LM.

- 20-160X faster in plan derivation compared to SoTA Auto Parallel Framework (Alpa)
- Discovered strategy has comparable performance to Megatron
- TAP also discovers new partially sharded plan that perform well when the memory is not so constrained.

## Conclusion

GitHub Repo

Preprint Paper

We present TAP, an automatic parallelism framework that efficiently discovers tensor parallel plans for large models. Leveraging the observation that subgraphs widely exist in neural networks, we design a system that runs at sub-linear end-to-end complexity w.r.t. to model size.