**Motivation**

- Pre-training a deep cost model offline requires a comprehensive dataset.
- Traditional learning makes the search very time-consuming.
- Existing tree-based models are insufficient for performance evaluation.
- Transferable knowledge is difficult to acquire across different platforms.

**Problem Formulation**

We describe a DNN model as a computation graph and then define some important terminologies. $G$ is partitioned into a set of subgraphs $S$ based on the graph-level optimizer. Each search task is extracted from an independent subgraph $S_i$ on a specific hardware platform $H$. Thus, we define search task $Q$ as follows:

$$Q(S_i) = \{Q^1(S_i), Q^2(S_i), \ldots, Q^n(S_i)\}$$

where $n$ is the number of subgraphs in $G$. Note that each subgraph $S_i$ contains a computation-intensive operator $\sigma$ and $\sigma \in S_i$. Therefore, we use $Q(S_i)$ to represent the $i$-th search task in $G$. Each subgraph $S_i$ has its own search space, which is determined by the input and output shapes, data precisions, memory layout, and the hardware platform. The search space is usually large enough to cover almost all kinds of tensor candidates.

A tensor program, denoted by $p$, represents an implementation of the subgraph using low-level primitives that are dependent on the hardware platform. Each tensor program can be considered as a candidate in the search space. We define the hierarchical search space $\phi_{ij}$, which decouples high-level structures from low-level details $\phi_i$, allowing for the efficient exploration of potential tensor candidates during the tuning process.

Here, we can transform a tuning problem into an optimization problem that explores the potential tensor programs in a hierarchical search space.

Given code generation function $\Theta$, high-level structure generation parameters $\phi_i$, low-level detail sampling parameters $\phi_{ij}$, computation-intensive operator $\sigma$, and operator setting $k$ (e.g., kernel size), our goal is to use $\phi_{ij}$ to build a hierarchical search space and generate tensor programs $p$ to achieve the optimal prediction score $y^*$ on a specific hardware platform $H$.

$$\phi_{ij} = \arg\max_{\phi} \quad p = f(g(\Theta, \phi_i, \phi_{ij}, k))$$

The cost model $f$ predicts score $y$ of the tensor program $p$. The accuracy of the cost model $f$ is crucial in finding ideal optimization configuration.

**Hierarchical Features**

- Coarse-grained operator embedding features: 10 dimension.
- Fine-grained statement features: 164 dimension.

**Model Architecture**

- Kernel embedding layer: extract a compact feature representation.
- Computation layer: captures essential information from the innermost non-loop computation statements.
- Regression layer: make the final prediction.

**Transfer Learning**

- Source domain: collected from T4 dataset with offline.
- Target domain: collected from 3090/2808 Ti with online.
- Cost model: XGBoost, LSTM, ATFormer.

**Experimental Results**

<table>
<thead>
<tr>
<th>Method</th>
<th>XGBoost</th>
<th>LSTM</th>
<th>ATFormer</th>
</tr>
</thead>
<tbody>
<tr>
<td>offline</td>
<td>0.79</td>
<td>0.72</td>
<td>0.74</td>
</tr>
<tr>
<td>online</td>
<td>0.78</td>
<td>0.73</td>
<td>0.74</td>
</tr>
</tbody>
</table>

**Conclusions**

- A novel and effective design for optimizing tensor programs.
- Self-attention blocks are utilized to explore global dependencies.
- Further analysis and performance improvement on Tensor Cores.
- Transfer learning from GPUs to CPUs.