EinNet: Optimizing Tensor Programs with Derivation-Based Transformations

Liyan Zheng, Haojie Wang, Jidong Zhai, Muyan Hu, Zixuan Ma, Tuowei Wang, and Shuhong Huang, *Tsinghua University;* Xupeng Miao, *Carnegie Mellon University;* Shizhi Tang and Kezhao Huang, *Tsinghua University;* Zhihao Jia, *Carnegie Mellon University*

Stefan Milosevic (sm2731)

Abstract and Introduction

- The paper introduces EINNET (Efficient Inference Network), a derivation-based optimizer for tensor programs, which are at the core of DNN computations.

- Traditional optimization techniques have limitations because they rely on a fixed set of predefined tensor operators, leading to restricted optimization possibilities.

- EINNET expands this by using general tensor algebra expressions, enabling a much larger optimization space and automatically creating new operators required by transformations

Background and Motivation

- The current tensor program optimization works at two levels: operator and graph. Operator-level optimization focuses on performance tuning for specific tensor operators, while graph-level optimization reorganizes DNN computations for efficiency.

- However, both approaches are constrained to predefined operator representable (POR) transformations, which EINNET aims to transcend by exploring general tensor algebra transformations

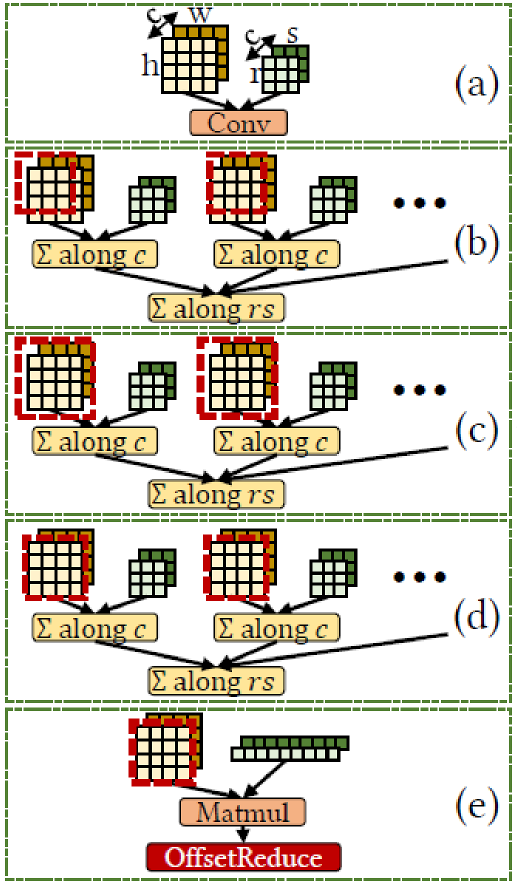
Key Contributions

- EINNET is distinguished by revealing operator computation semantics and applying derivation rules to tensor algebra expressions, allowing for the reorganization of computation into arbitrary tensor expressions.

- This system can potentially introduce novel program transformations and optimize beyond the capabilities of existing frameworks

Addressing Optimization Challenges

- EINNET tackles three main challenges: discovering transformations between general expressions, converting expressions back to executable kernels (expression instantiation), and efficiently finding optimizing transformations in the vast space of general tensor algebra transformations



$$\mathcal{E}_1 \quad \mathop{\mathrm{L}}_{hwf}\mathop{\Sigma}_{crs} A[h+r,w+s,c]K[r,s,f,c]$$

Summation splitting

$$\mathcal{E}_2 = \mathop{\mathrm{L}}_{hwf}\mathop{\Sigma}_{rs}\left\{\mathop{\mathrm{L}}_{rshwf}\mathop{\Sigma}_{c} A[h+r,w+s,c]K[r,s,f,c]\right\}[r,s,h,w,f]$$

Variable substitution: $t_1 = h+r, t_2 = w+s$

$$\mathcal{E}_3 = \mathop{\mathrm{L}}_{hwf}\mathop{\Sigma}_{rs}\left\{\mathop{\mathrm{L}}_{rshwf}\left\{\mathop{\mathrm{L}}_{rsf}\mathop{\mathrm{L}}_{t_1=r}^{H+r}\mathop{\mathrm{L}}_{t_2=s}^{W+s}\mathop{\Sigma}_{c} A[t_1,t_2,c]K[r,s,f,c]\right\}[r,s,f,h+r,w+s]\right\}[r,s,h,w,f]$$

Boundary relaxing

$$\mathcal{E}_4 = \mathop{\mathrm{L}}_{hwf}\mathop{\Sigma}_{rs}\left\{\mathop{\mathrm{L}}_{rshwf}\left\{\mathop{\mathrm{L}}_{rsf}\mathop{\mathrm{L}}_{t_1=-1}^{H+1}\mathop{\mathrm{L}}_{t_2=-1}^{W+1}\mathop{\Sigma}_{c} A[t_1,t_2,c]K[r,s,f,c]\right\}[r,s,f,h+r,w+s]\right\}[r,s,h,w,f]$$

Traversal merging

$$\mathcal{E}_5 = \mathop{\mathrm{L}}_{hwf}\mathop{\Sigma}_{rs}\left\{\mathop{\mathrm{L}}_{rsf}\mathop{\mathrm{L}}_{t_1=-1}^{H+1}\mathop{\mathrm{L}}_{t_2=-1}^{W+1}\mathop{\Sigma}_{c} A[t_1,t_2,c]K[r,s,f,c]\right\}[r,s,f,h+r,w+s]$$

Boundary tightening

$$\mathcal{E}_6 = \mathop{\mathrm{L}}_{hwf}\mathop{\Sigma}_{rs}\left\{\mathop{\mathrm{L}}_{rsf}\mathop{\mathrm{L}}_{t_1=0}^{H}\mathop{\mathrm{L}}_{t_2=0}^{W}\mathop{\Sigma}_{c} A[t_1,t_2,c]K[r,s,f,c]\right\}[r,s,f,h+r,w+s]$$

Operator matching: $T_1 = K(rsf,c)\times A(c,t_1t_2)$

$$\mathcal{E}_7 = \mathop{\mathrm{L}}_{hwf}\mathop{\Sigma}_{rs} T_1[r,s,f,h+r,w+s]$$

eOperator generation: T_2 *is computed by a memory bound operator*

$$\mathcal{E}_8 = T_2$$

Figure 5: The derivation process of the example in Figure 3(b), which transforms `Conv` with `Matmul` and eOperators

Methodology

- The paper details the derivation rules EINNET employs to transform tensor programs into optimized forms
- These rules encompass intra-expression derivation
- The approach combines traversal and summation notations, along with scope-based transformations to optimize computation

Optimization

- EINNET optimizes tensor programs: i.e.) transforming convolution operations into matrix multiplications and fusing multiple operators into a single one for efficiency

$$\mathop{\mathrm{L}}_{c=0}^{C}\mathop{\mathrm{L}}_{r=0}^{R}\sum_{k_0=0}^{K}\sum_{k_1=0}^{K} A[c,k_0]B[k_0,k_1]C[k_1,r] \qquad \text{(a)}$$

$$=\mathop{\mathrm{L}}_{c=0}^{C}\mathop{\mathrm{L}}_{r=0}^{R}\sum_{k_0=0}^{K}\boxed{\left\{\mathop{\mathrm{L}}_{c'=0}^{C}\mathop{\mathrm{L}}_{k_2=0}^{K}\sum_{k_1=0}^{K} A[c',k_1]B[k_1,k_2]\right\}}[c,k_0]C[k_0,r] \qquad \text{(b)}$$

Traversal notation Summation notation Scope

Figure 4: A tensor algebra expression example for two matrix multiplications $A \times B \times C$. The red box highlights a *scope* that instantiates the intermediate result of $A \times B$.

Implementation and Evaluation

- EINNET has been implemented with over 23,000 lines of code in C++ and Python and has shown significant performance improvements over existing optimizers, with speedups of up to 2.72× on certain hardware

Overview of EINNET

- The optimizer → an input tensor program into subprograms, translates these into tensor algebra expressions, applies derivation rules, and generates optimized subprograms.

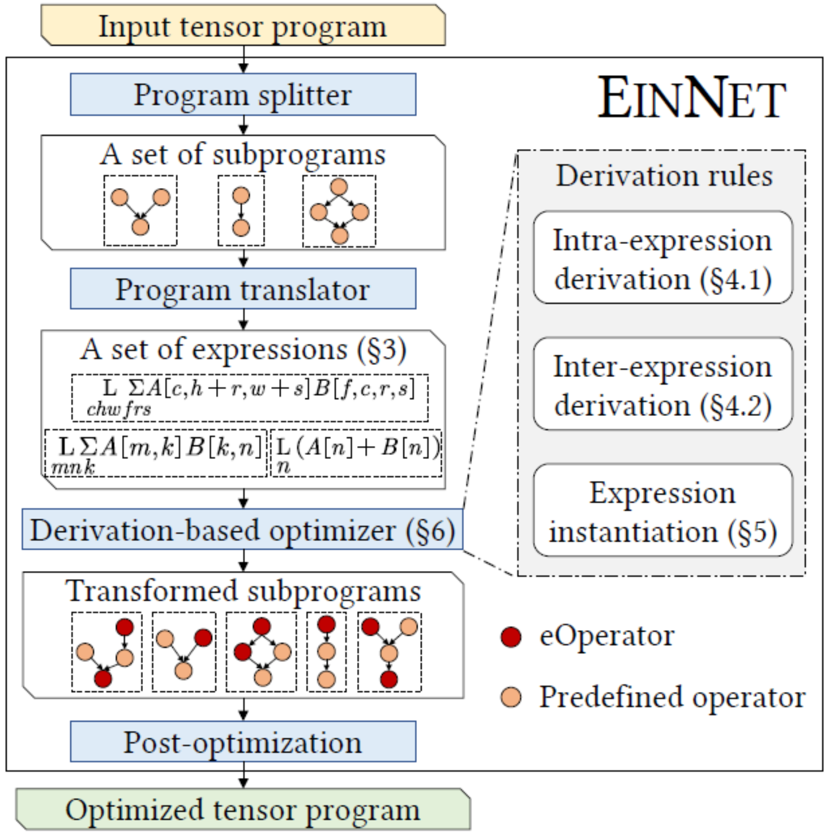


Figure 2: EINNET overview

Conclusion and Final Thoughts

- **Innovation in Optimization**: EINNET represents a significant advancement in tensor program optimization, pushing beyond the constraints of predefined operator representable (POR) transformations by leveraging general tensor algebra expressions.
- **Methodological Breakthrough**: Introduces a derivation-based mechanism that transforms tensor programs into optimized forms, utilizing a set of sophisticated derivation rules that ensure functional equivalence while enhancing performance.
- **Performance Enhancement**: Demonstrated substantial performance improvements over existing optimization frameworks, achieving up to 2.72× speedup
- **Practical Impact**: EINNET's capabilities can be applied to a variety of real-world applications that utilize DNNs, potentially leading to efficiency gains in critical areas such as autonomous driving, speech recognition, etc.
- **Future Directions**:
  - **Expand Optimization Techniques**
  - **Broader Hardware Compatibility**
  - **Integration with Existing Frameworks**