Unity

Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization

Unger et al., 2022

Jan Pytel
University of Cambridge
November 19, 2025

Introduction & Motivation

- Unity's Goal: optimise DNN training
- Two main areas of DNN optimisation
  - algebraic transformations
  - parallelisation
  - (can be done separately with existing systems)
- Unity
  - first system that optimises these **jointly**
  - key novel concepts
    - *parallel computation graph* IR
      - (expresses **computation**, **parallelisation**, **communication** of distributed DNN training)
    - generation of optimisations in form of **graph substitutions**
    - hierarchical **search algorithm** to find optimal combination of substitutions

Algebraic Transformations & Parallelisation

- Algebraic Transformations
  - exploit **operator identities** to optimise underlying computations
  - examples
    - operator fusion (merging two operators into a single equivalent)
    - operator reordering (when associativity or commutativity allows)
- Parallelisation
  - distribute operators over multiple devices
  - partition-n-reduce
    - split operator’s input across devices
    - compute partial results independently on each device (same operator)
    - reduce partial results
  - parallelism dimensions
    - data, model, spatial, reduction, pipeline, …

Parallelism Dimensions

- Data
  - replicate the model onto devices
  - split training data into subsets and distribute across devices
- Model
  - split model into disjoint sub-models
  - train each sub-model on a separate device
- Spatial
  - divide spatial dimensions (e.g. height, width) of a tensor into partitions
  - assign each partition to a specific device
- Reduction
  - exploit linearity of tensor algebra operations
    - (e.g. split matrices across columns or rows for matrix multiplication)
- Pipeline
  - divide model into stages and run on different devices in a pipelined sequence

Joint vs Sequential Optimisation

- Previous automated optimisation systems
  - algebraic transformations: MetaFlow, TASO, PET
  - parallelism: FlexFlow, automap, Tofu, Whale
- Can apply these sequentially
  - algebraic transformations (1st)
  - parallelism (2nd)
  - e.g. TASO + FlexFlow
- Issue: can miss significant optimisation opportunities

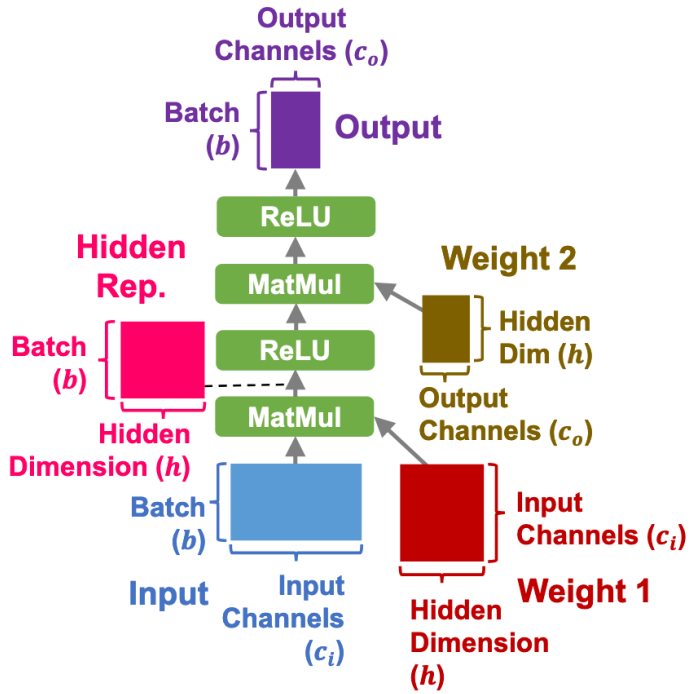
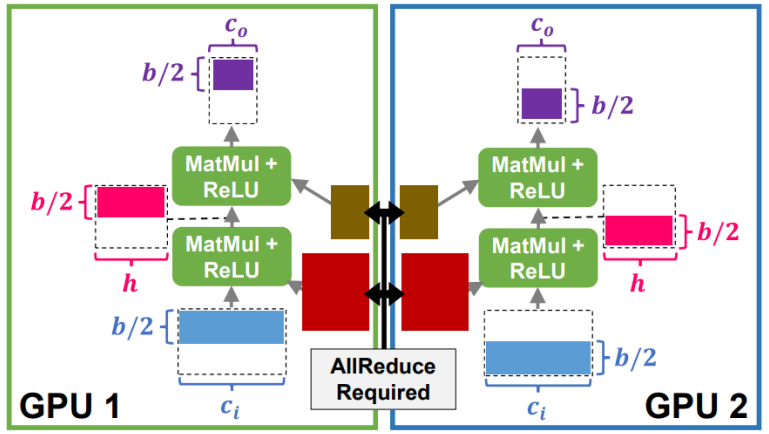
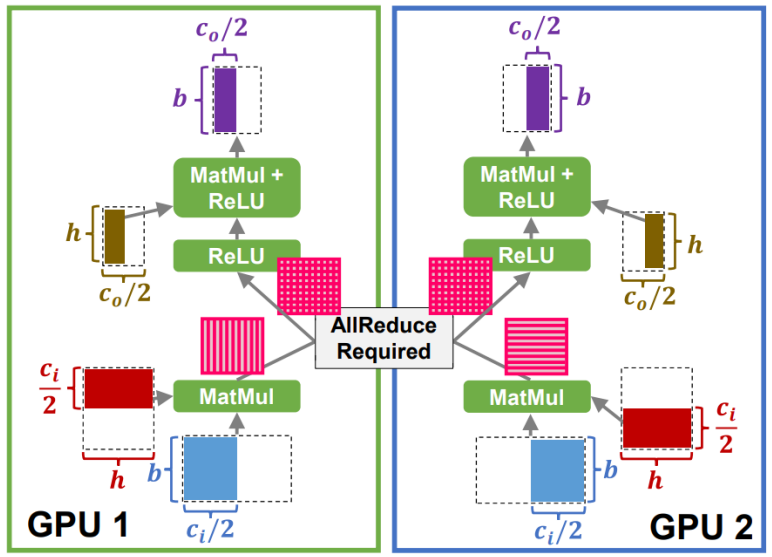


Figure 1: Computation graph for a 2-layer MLP.

- communication cost
  - a) $2(c_i h + h c_0)$
  - b) 4bh
- example: MNIST image classification
  - (b) reduces communication cost by 6x



(a) Sequential optimization.



(b) Joint optimization.

Figure 2: Comparing joint and sequential optimizations.

Graph Representation: Prior Work

- Represent DNN as *computation graph*
    - node: mathematical tensor operation
    - edge: tensor passed between operations
- Algebraic transformation – iteratively applied graph substitutions
- Parallelisation – parallelism annotations for each node
- Limitations
    - restricted joint optimisation
    - communication costs of parallelism not expressed

Unity's Parallel Computation Graph (PCG)

- Unity: both algebraic transformations and parallelisation as **graph substitutions**
- Needs a **unified graph representation**
  - to express computation, parallelism and communication
- PCG
  - graph
    - nodes: also capture parallelisation through *parallelization operators*
    - edges: also capture distributed movement of tensor data
  - tensor representation
    - Unity models tensors as sets of data dimensions
    - each dimension:
      - size
      - degree (number of partitions the tensor has been divided into along that dimension)
        - encodes parallelism, not just shape
    - each tensor
      - replica dimension (number of replicas of that tensor's data)
  - *machine mapping:* maps operators onto devices to be executed on

PCG: Parallelisation Operators

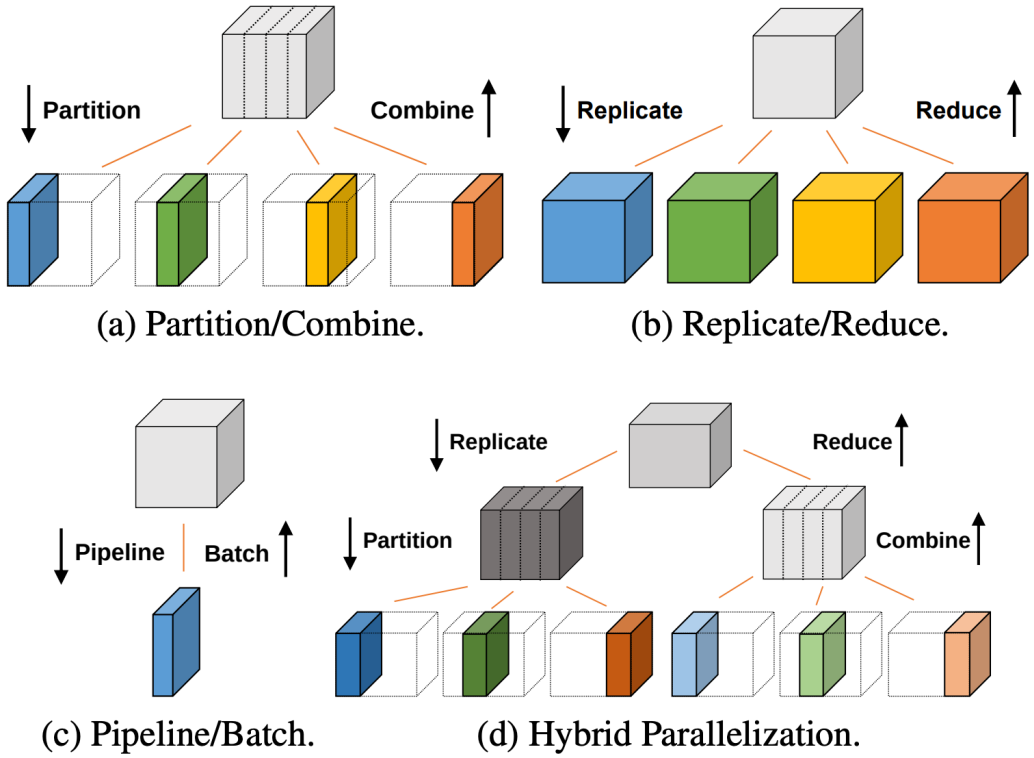


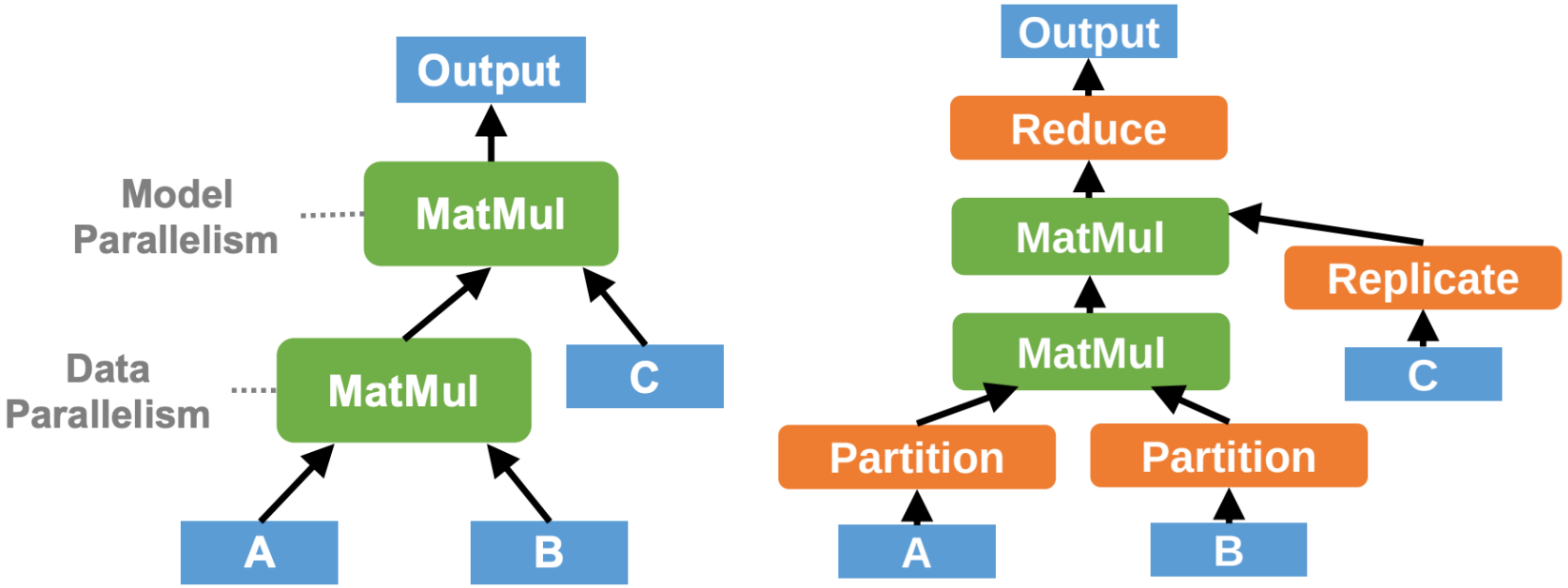
Figure 8: Parallelization operators in Unity.

a) *Partition*: *split* a dimension into equal-sized partitions; *Combine*: inverse op

b) *Replicate/Reduce*: copy/sum tensors

c) *Pipeline*: split a dimension into equal-sized partitions and process one at a time; *Batch*: aggregate tensors across iterations

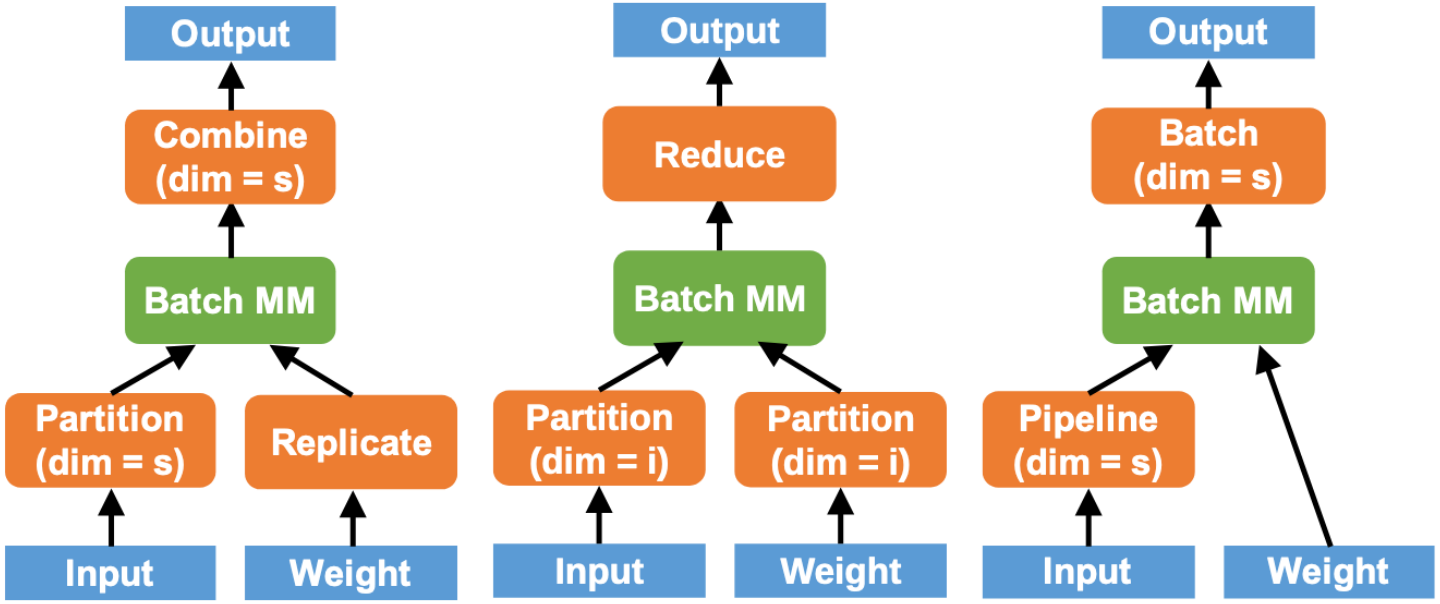
Computation Graph vs PCG: Example



(a) Computation graph.

(b) Parallel computation graph.

Parallelisation Strategies with PCG

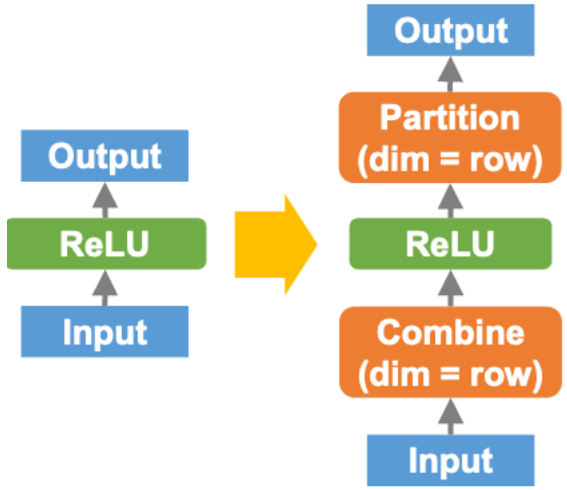


(a) Data/Sample. (b) Reduction. (c) Pipeline.

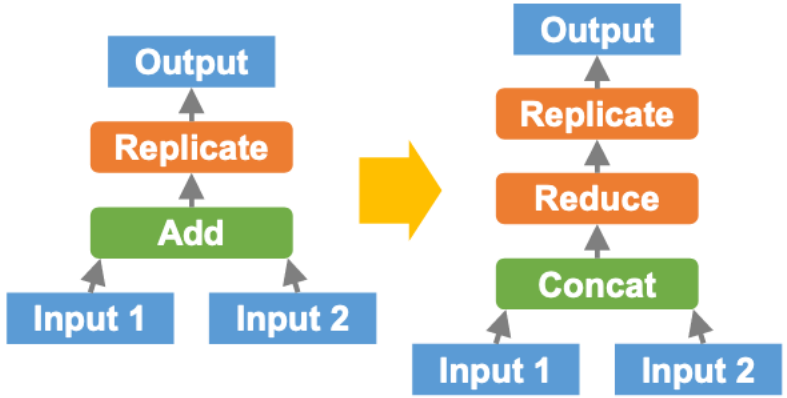
Graph Substitutions

- Goal: represent algebraic transformations and parallelisation
- Challenge
  - need a set of graph substitutions
  - number of substitutions increases exponentially with size
- Unity
  - use compositions of small PCG substitutions
- Substitution generation
  1. enumerate all possible PCGs up to a fixed size
  2. use *fingerprinting* to create an initial set of candidate substitutions
  3. formally verify the substitutions using Z3 (automated theorem prover)

Substitution Generation: Examples



spatial parallelism is valid for ReLU



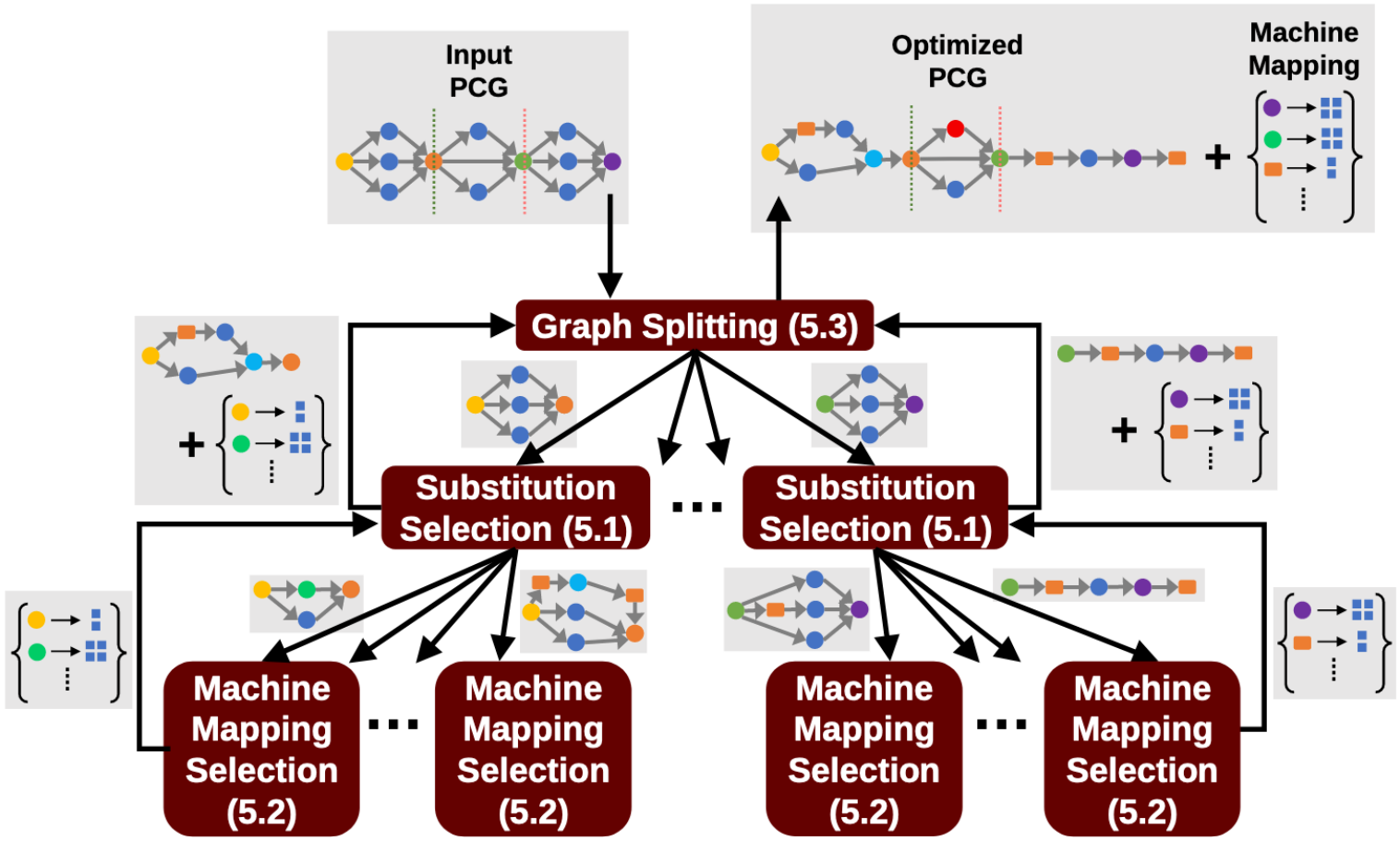
hybrid algebraic-parallel transformation

(Add → Concat + Reduce)

Joint Optimisation

- Input
  - PCG (DNN representation)
  - set of operator-level *machine mappings*
  - set of PCG substitutions
- Output
  - sequence of PCG substitutions & machine mapping for that PCG that minimise per-iteration training time
- Challenge: exponentially larger search space
- Solution: three-level hierarchical search algorithm
  - splits input PCG into subgraphs
  - finds optimum sequence of substitutions for each subgraph
  - combines them

Unity's Hierarchical Search



Evaluation

Task	Architecture	Dataset
Image Classification	ResNeXt-50 [60]	ImageNet [46]
	Inception-v3 [51]	ImageNet [46]
Language Models	BERT-Large [14]	WikiText-2 [35]
Recommendation Systems	DLRM [41]	Criteo Kaggle [4]
	XDL [28]	Criteo Kaggle [4]
Precision Medicine	CANDLE-Uno [3]	Dose response data [1]
Regression	MLP [17]	Synthetic data

Table 1: Overview of the seven DNNs evaluated.

Evaluation

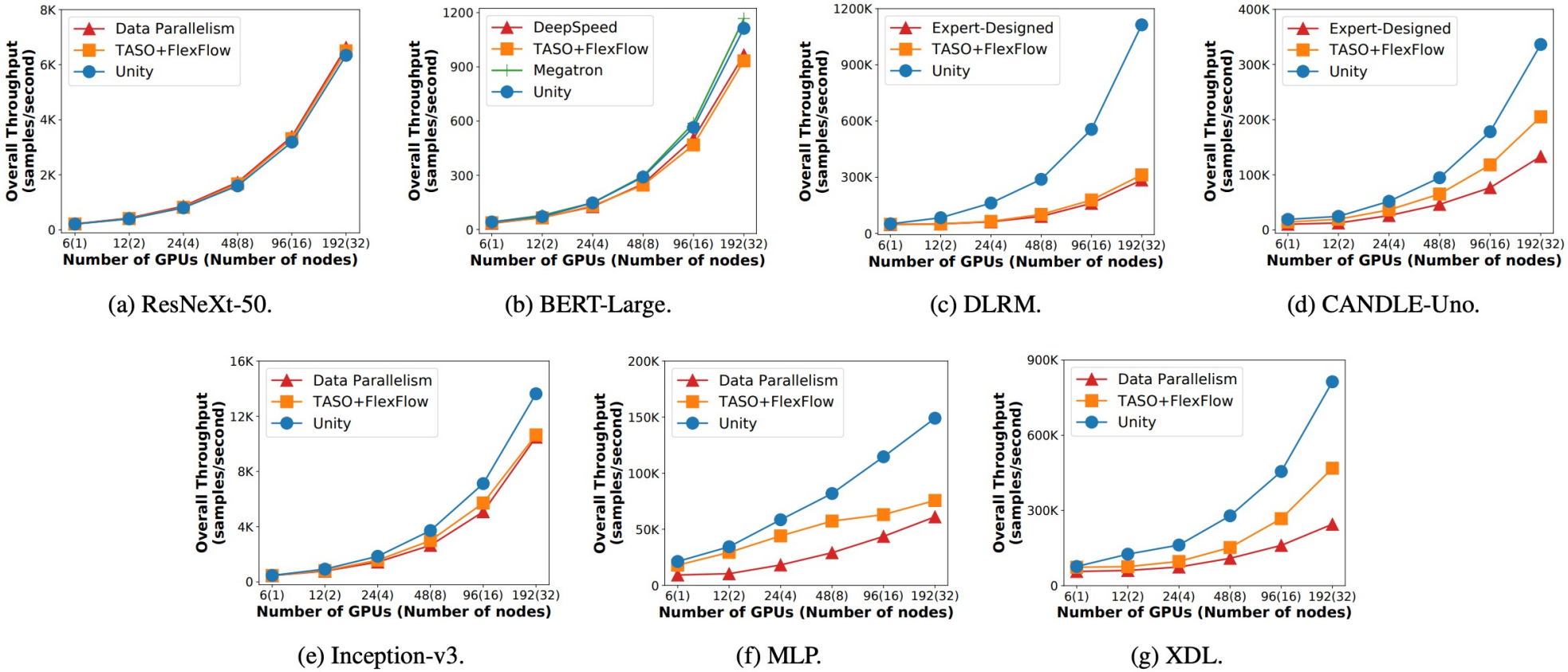
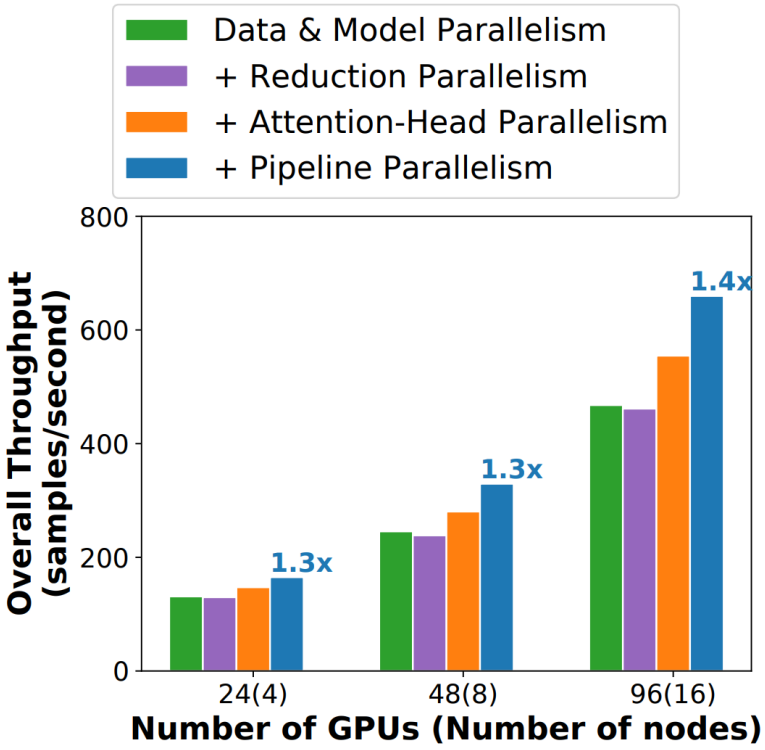
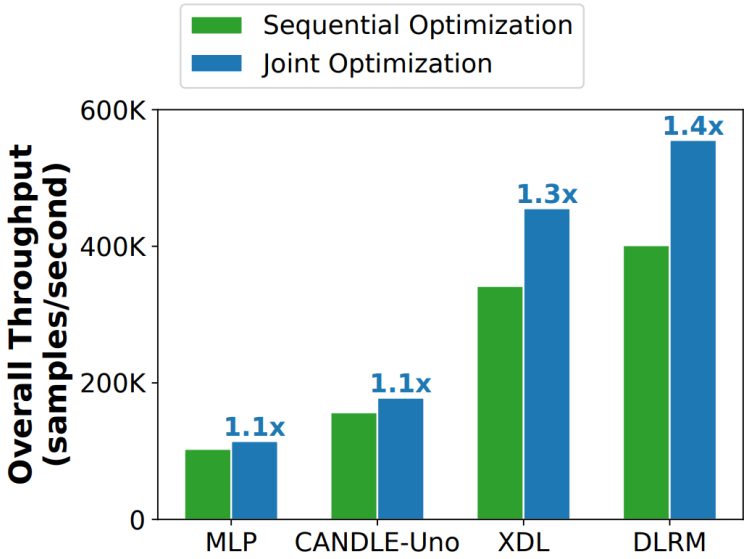


Figure 14: Training throughput comparison among existing frameworks and Unity. The experiments were performed on the Summit supercomputer [2] with 6 GPUs per node. All numbers were measured by averaging 1,000 training iterations.

Evaluation



Impact of Parallelism Dimension



Joint > Sequential Optimisation

Summary & Discussion

- Strengths
  - clear problem and motivation
  - elegant IR
  - correctness guarantees (Z3 verification)
  - good results (performance & generalisation)
  - influential direction
- Limitations
  - limited practicality and integration (custom IR)
  - simplified cost and hardware model
    - assumes homogeneous devices and uniform communication costs
  - no memory awareness
  - search pruning