

Neural Algorithmic Reasoning + A Generalist Neural Algorithmic Learner

Presented by
Lee Guan Bo Ambrose



UNIVERSITY OF
WATERLOO

DAVID R. CHERITON SCHOOL
OF COMPUTER SCIENCE



SCHOOL OF
**COMPUTING &
DATA SCIENCE**
The University of Hong Kong

Neural Algorithmic Reasoning

Petar Veličković and Charles Blundell

DeepMind

A Generalist Neural Algorithmic Learner

Borja Ibarz*
DeepMind

Vitaly Kurin[†]
University of Oxford

George Papamakarios
DeepMind

Kyriacos Nikiforou
DeepMind

Mehdi Bennani
DeepMind

Róbert Csordás[†]
IDSIA, USI Lugano

Andrew Dudzik
DeepMind

Matko Bošnjak
DeepMind

Alex Vitvitskyi
DeepMind

Yulia Rubanova
DeepMind

Andreea Deac[†]
Mila, Université de Montréal

Beatrice Bevilacqua[†]
Purdue University

Yaroslav Ganin
DeepMind

Charles Blundell
DeepMind

Petar Veličković*
DeepMind

Part 1: Introduction

Algorithm vs Deep Learning

Deep Learning: learn from data and are then deployed to make predictions or decisions

Challenge: generalisation

Algorithm: specifying steps, a precondition and a postcondition

Challenge: flexibility to the problem tackled

Algorithm example

Exp: what kind of input it expects (e.g., a finite list of integers allocated in memory it can modify) and then the postcondition might state that after execution, the input memory location contains the same integers but in ascending order

2 approaches of combining them

- Deep reinforcement learning to use existing, known algorithms as fixed external tools
- Teach deep neural networks to imitate the workings of an existing algorithm
 - Producing the same output
 - In the strongest case by replicating the same intermediate steps
 - Potentially learn how to replicate multiple algorithms
 - Potentially learn new variants or algorithm

Bottleneck of algorithm

Converting problem from natural space to abstract space (partially observable)

- Exp: railway junctions being nodes in a graph, and edges between them endowed with scalar capacities, specifying the limits of traffic flow along edges

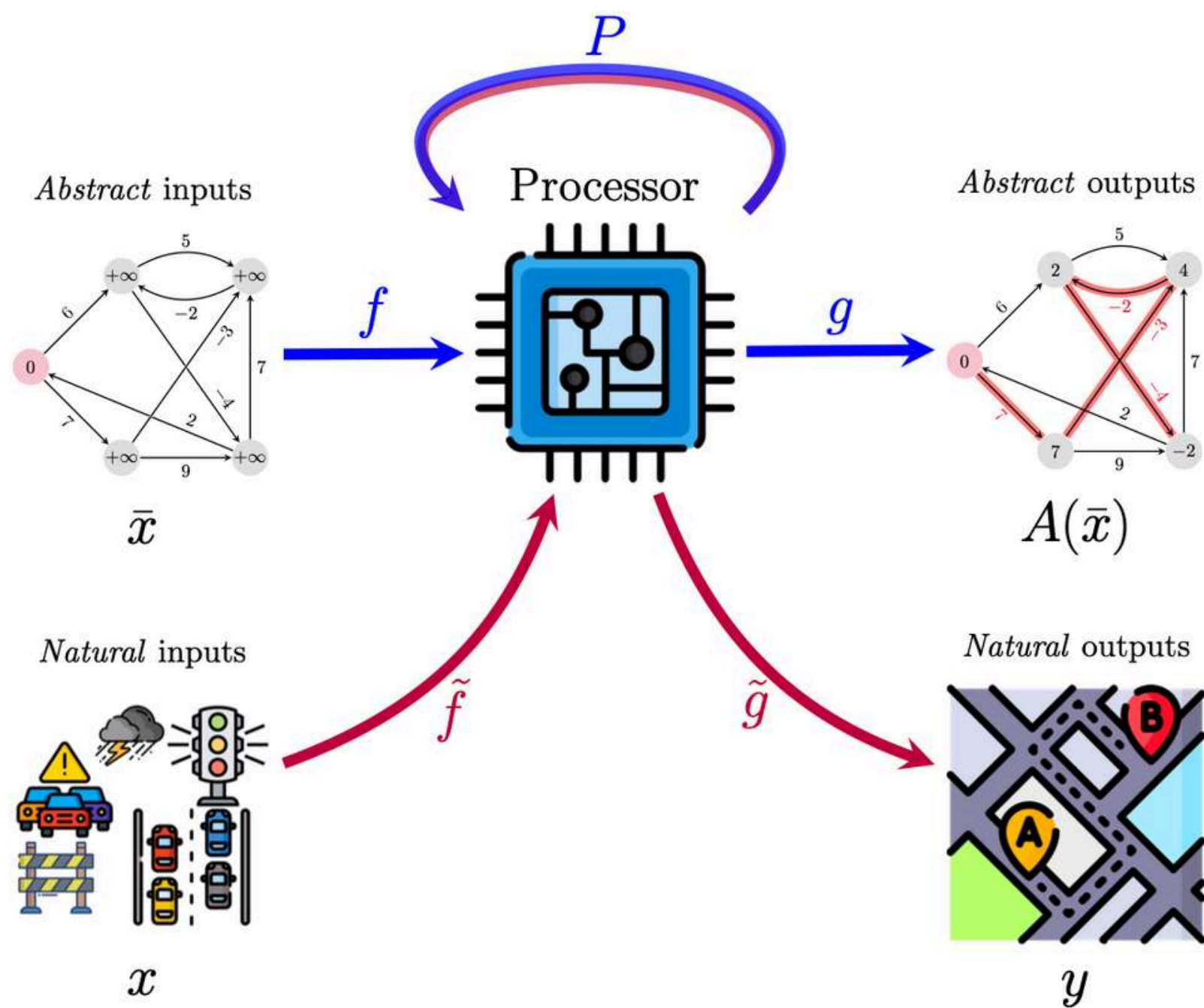
Abstraction lead to information loss

Replacing the algorithm with neural network

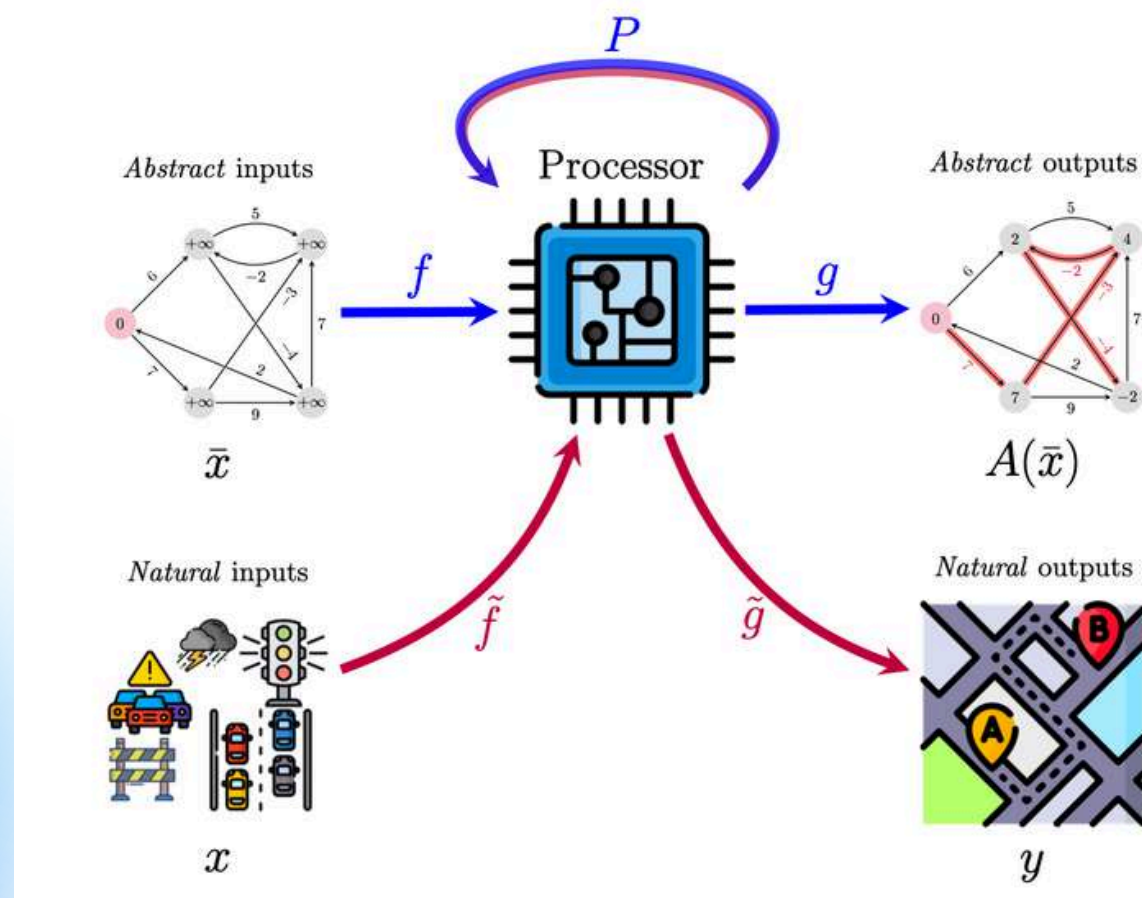
Train a processor network that

- Aligned with the computations of the target algorithm
- Operates by matrix multiplications, hence natively admits useful gradients
- Operates over high-dimensional latent spaces, hence is not vulnerable to bottleneck phenomena and may be more data-efficient.

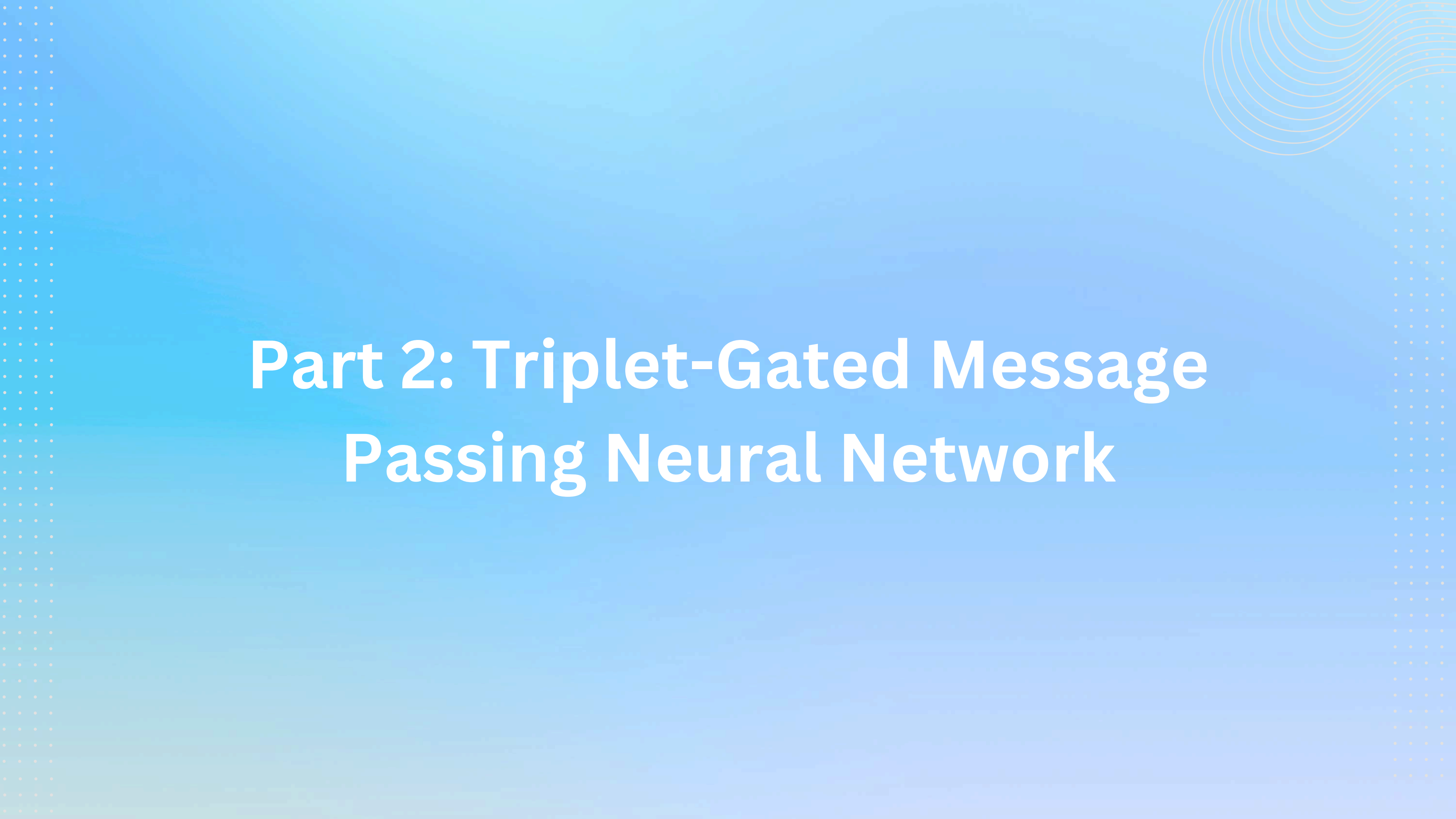
Visualisation



Visualisation



1. Learn an algorithmic reasoner for A , by learning to execute it on synthetically generated inputs, \bar{x} . This yields functions f, P, g such that $g(P(f(\bar{x}))) \approx A(\bar{x})$. f and g are encoder/decoder functions, designed to carry data to and from the latent space of P (the processor network).
2. Set up appropriate encoder and decoder neural networks, \tilde{f} and \tilde{g} , to process raw data and produce desirable outputs. The encoder should produce embeddings that correspond to the input dimension of P , while the decoder should operate over input embeddings that correspond to the output dimension of P .
3. Swap out f and g for \tilde{f} and \tilde{g} , and learn their parameters by gradient descent on any differentiable loss function that compares $\tilde{g}(P(\tilde{f}(x)))$ to ground-truth outputs, y . The parameters of P should be kept *frozen*.



Part 2: Triplet-Gated Message Passing Neural Network

Overview

Related-works:

- Specialist models
- Nearly-identical control flow backbone for each algorithm
- Algorithmic alignment, causality and self-supervised learning

Objective: Train a generalist neural algorithmic learner that capable of dealing with OOD

Main contributions

- Present a series of improvements to the training, optimisation, input representations, and GNN architectures
- Results: Improve the benchmark by 20%

Tasks

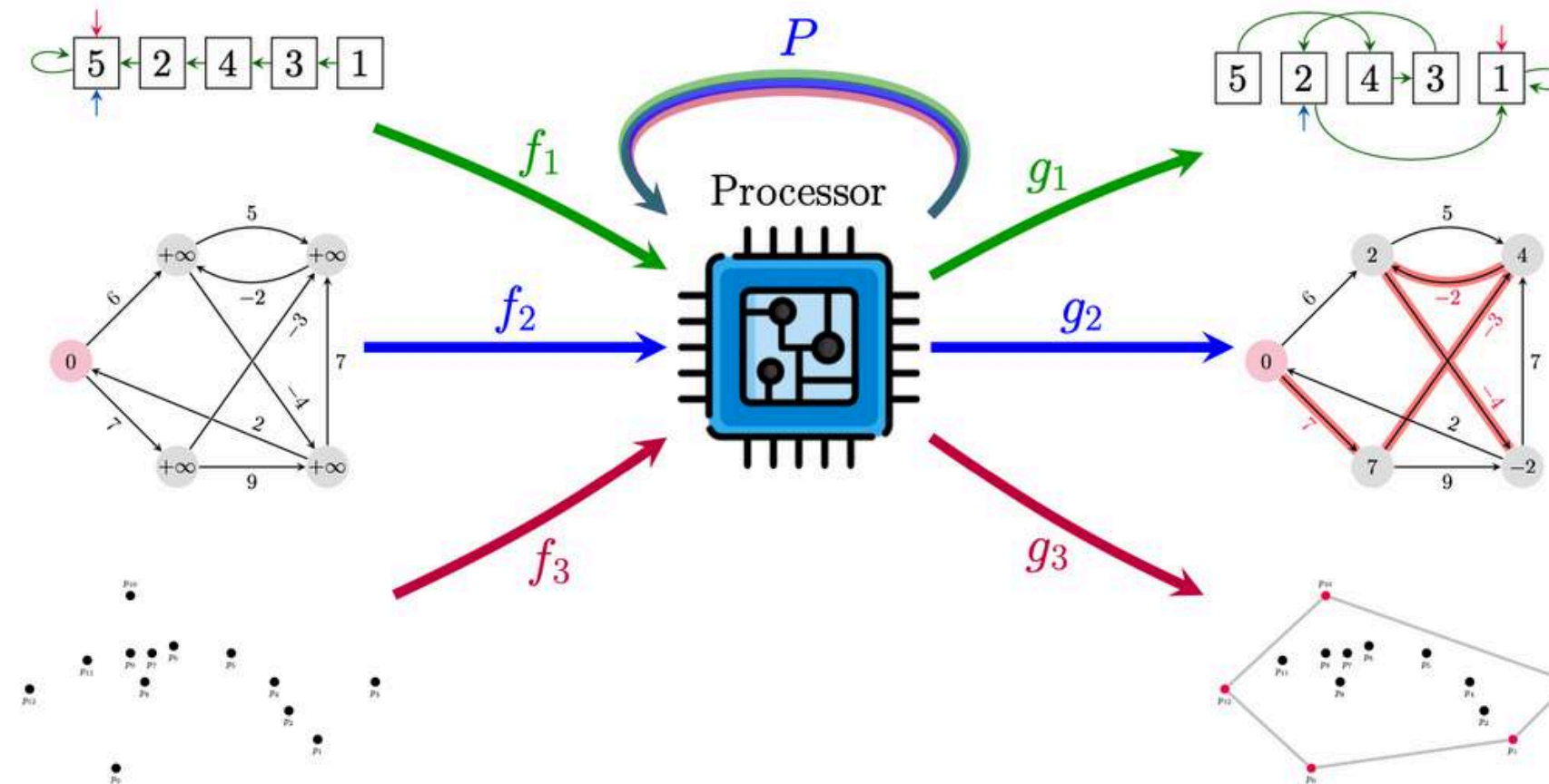


Figure 1: Our generalist neural algorithmic learner is a *single* processor GNN P , with a single set of weights, capable of solving several algorithmic tasks, τ , in a shared latent space (each of which is attached to P with simple encoders/decoders f_τ and g_τ). Among others, our processor network is capable of sorting (**top**), shortest path-finding (**middle**), and convex hull finding (**bottom**).

Type of tasks: spanning sorting, searching, greedy algorithms, dynamic programming, graph algorithms, string algorithms and geometric algorithms

Dataset

Dataset: CLRS-50

- collection of thirty classical algorithmic tasks spanning the before categories, along with a unified representational interface
- each task is specified by a number of inputs, hints (time series) and outputs located in either the nodes, the edges, or the graph itself
- five types of features: scalar, categorical, mask, mask_one and pointer, with their own encoding and decoding strategies and loss functions

Base Model

- Encode-process-decode paradigm
 - Task-based encoder and decoder
- Linear encoder for input and hint:
 - Embed them into high dimensional vectors and add them together
 - At the end of time t :
single set of embeddings $\{\mathbf{x}_i^{(t)}, \mathbf{e}_{ij}^{(t)}, \mathbf{g}^{(t)}\}$, shapes $n \times h$, $n \times n \times h$, and h ,
in the nodes, edges and graph, respectively.
 - independent of number and type of inputs and hints
 - allow sharing of latent space

Base Model

Processor. The embeddings are fed into a *processor* P , a GNN that performs one step of computation. The processor transforms the input node, edge and graph embeddings into *processed* node embeddings, $\mathbf{h}_i^{(t)}$. Additionally, the processor uses the processed node embeddings from the previous step, $\mathbf{h}_i^{(t-1)}$, as inputs. Importantly, the same processor model can operate on graphs of *any* size. We leverage the message-passing neural network [35, MPNN], using the max aggregation and passing messages over a *fully-connected graph*, as our base model. The MPNN computes processed embeddings as follows:

$$\mathbf{z}^{(t)} = \mathbf{x}_i^{(t)} \parallel \mathbf{h}_i^{(t-1)} \quad \mathbf{m}_i^{(t)} = \max_{1 \leq j \leq n} f_m \left(\mathbf{z}_i^{(t)}, \mathbf{z}_j^{(t)}, \mathbf{e}_{ij}^{(t)}, \mathbf{g}^{(t)} \right) \quad \mathbf{h}_i^{(t)} = f_r \left(\mathbf{z}_i^{(t)}, \mathbf{m}_i^{(t)} \right) \quad (1)$$

starting from $\mathbf{h}^{(0)} = \mathbf{0}$. Here \parallel denotes concatenation, $f_m : \mathbb{R}^{2h} \times \mathbb{R}^{2h} \times \mathbb{R}^h \times \mathbb{R}^h \rightarrow \mathbb{R}^h$ is the *message function* (for which we use a three-layer MLP with ReLU activations), and $f_r : \mathbb{R}^{2h} \times \mathbb{R}^h \rightarrow \mathbb{R}^h$ is the *readout function* (for which we use a linear layer with ReLU activation). The use of the max aggregator is well-motivated by prior work [5, 9], and we use the fully connected graph—letting the neighbours j range over all nodes ($1 \leq j \leq n$)—in order to allow the model to overcome situations where the input graph structure may be suboptimal. Layer normalisation [36] is applied to $\mathbf{h}_i^{(t)}$ before using them further. Further details on the MPNN processor may be found in Veličković et al. [5].

Base Model

- Task-based decoder
 - rely on linear decoder for each hint and output
 - with mechanism to calculate pairwise node similarities
- Loss
 - hint prediction losses are averaged across hints and time
 - output loss is averaged across outputs
 - hint loss and output loss are added together
- Train on samples with size $n \leq 16$
- Evaluate on in-distribution and OOD ($n > 16$)

Model Improvements – Dataset & Training

- Remove teacher forcing
 - In base model, hint are provided with probability 0.5 during training but not provided during evaluation
 - without teacher forcing, losses tended to grow and destabilise the training
- Augment training data
 - Used the on-line samplers in CLRS to generate new training examples on the fly, rather than using a fixed dataset
 - Trained on examples of mixed sizes, $n \leq 16$, rather than only 16
 - Vary the data
 - For graph algorithm: Varied the connectivity probability p of the input graphs
 - For string matching algorithm: varied the length of the pattern to be matched

Model Improvements – Dataset & Training

- Improving training stability with encoder initialisation and gradient clipping
 - Xavier initialisation and LeCun initialisation

Model Improvements – Encoder & Decoder

- Randomised position scalar
 - originally uniquely indexes for each nodes: values linearly spaced between 0 and 1 along the node
 - replaced them with random values, uniformly sampled in $[0,1]$, sorted to match the initial order implied by the linearly spaced values.

Model Improvements – Processor Networks

- Gating mechanisms
 - sometimes nodes are kept unchanged during intermediate steps
 - harm the results in multi-task experiment

per-node gating vector:

$$\mathbf{g}_i^{(t)} = f_g \left(\mathbf{z}_i^{(t)}, \mathbf{m}_i^{(t)} \right)$$

processed gated embeddings:

$$\hat{\mathbf{h}}_i^{(t)} = \mathbf{g}_i^{(t)} \odot \mathbf{h}_i^{(t)} + (1 - \mathbf{g}_i^{(t)}) \odot \mathbf{h}_i^{(t-1)}$$

Model Improvements – Processor Networks

- Triplet reasoning
 - message passing towards edges

$$\mathbf{t}_{ijk} = \psi_t(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k, \mathbf{e}_{ij}, \mathbf{e}_{ik}, \mathbf{e}_{kj}, \mathbf{g})$$

/

triplet message function

$$\mathbf{h}_{ij} = \phi_t(\max_k \mathbf{t}_{ijk})$$

/

edge readout function

Results

Table 1: Single-task OOD micro-F₁ score of previous SOTA Memnet, MPNN and PGN [5] and our best model Triplet-GMPNN with all our improvements, after 10,000 training steps.

Alg. Type	Memnet [5]	MPNN [5]	PGN [5]	Triplet-GMPNN (ours)
Div. & C.	13.05% \pm 0.14	20.30% \pm 0.85	65.23% \pm 4.44	76.36% \pm 1.34
DP	67.94% \pm 8.20	65.10% \pm 6.44	70.58% \pm 6.48	81.99% \pm 4.98
Geometry	45.14% \pm 11.95	73.11% \pm 17.19	61.19% \pm 7.01	94.09% \pm 2.30
Graphs	24.12% \pm 5.30	62.79% \pm 8.75	60.25% \pm 8.42	81.41% \pm 6.21
Greedy	53.42% \pm 20.82	82.39% \pm 3.01	75.84% \pm 6.59	91.21% \pm 2.95
Search	34.35% \pm 21.67	41.20% \pm 19.87	56.11% \pm 21.56	58.61% \pm 24.34
Sorting	71.53% \pm 1.41	11.83% \pm 2.78	15.45% \pm 8.46	60.37% \pm 12.16
Strings	1.51% \pm 0.46	3.21% \pm 0.94	2.04% \pm 0.20	49.09% \pm 23.49
Overall avg.	38.88%	44.99%	50.84%	74.14%
> 90%	0/30	6/30	3/30	11/30
> 80%	3/30	9/30	7/30	17/30
> 60%	10/30	14/30	15/30	24/30

- Evaluation metric: micro-f1 score
- 20% higher than the next best model

Results

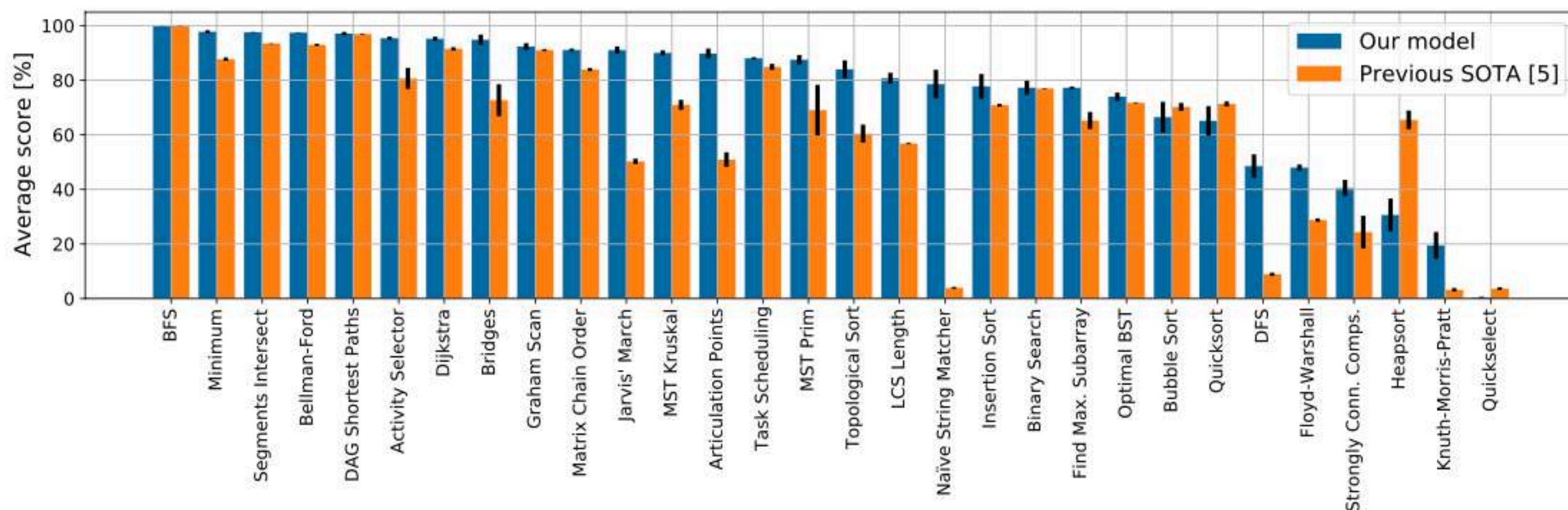
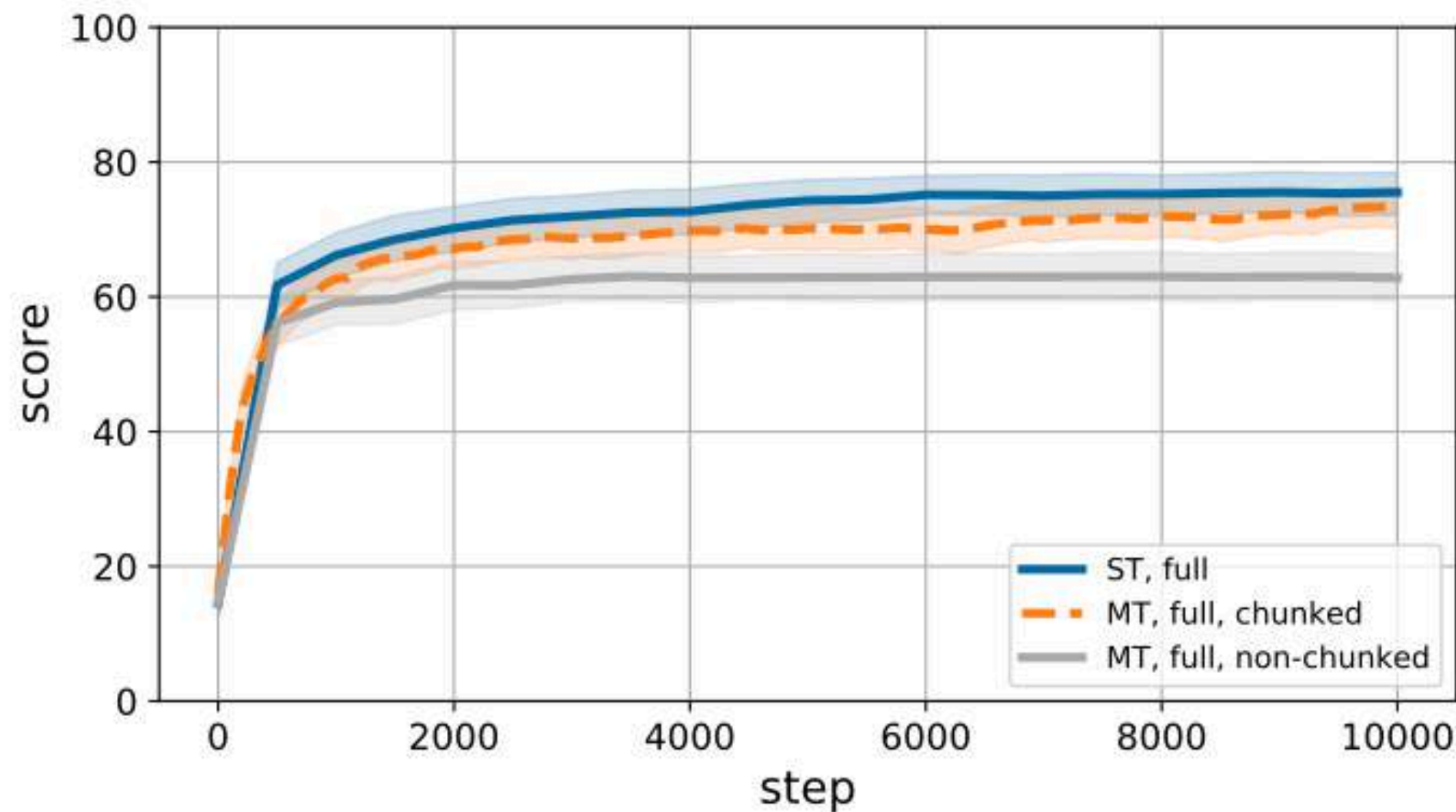


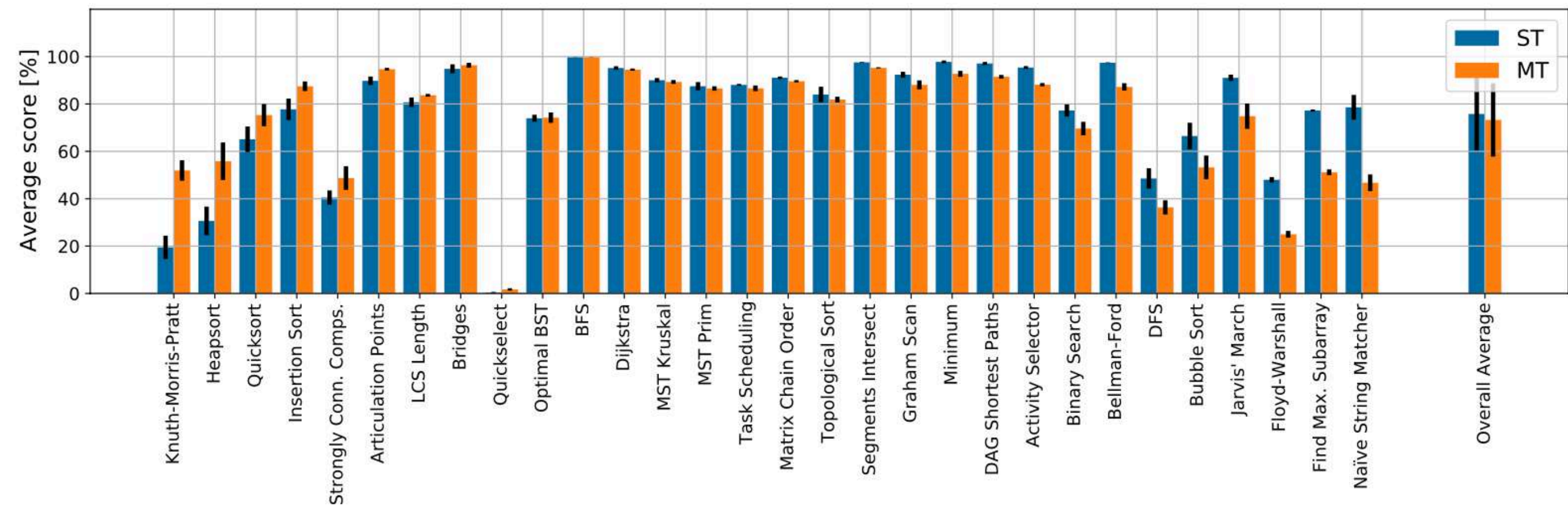
Figure 2: The OOD performance in single-task experiments before and after the improvements presented in this paper, sorted in descending order of current performance. Error bars represent standard error of the mean across seeds (3 seeds for previous SOTA experiments, 10 seeds for current). The previous SOTA values are the best of MPNN, PGN and Memnet models (see Table 2).

Multi-task experiment

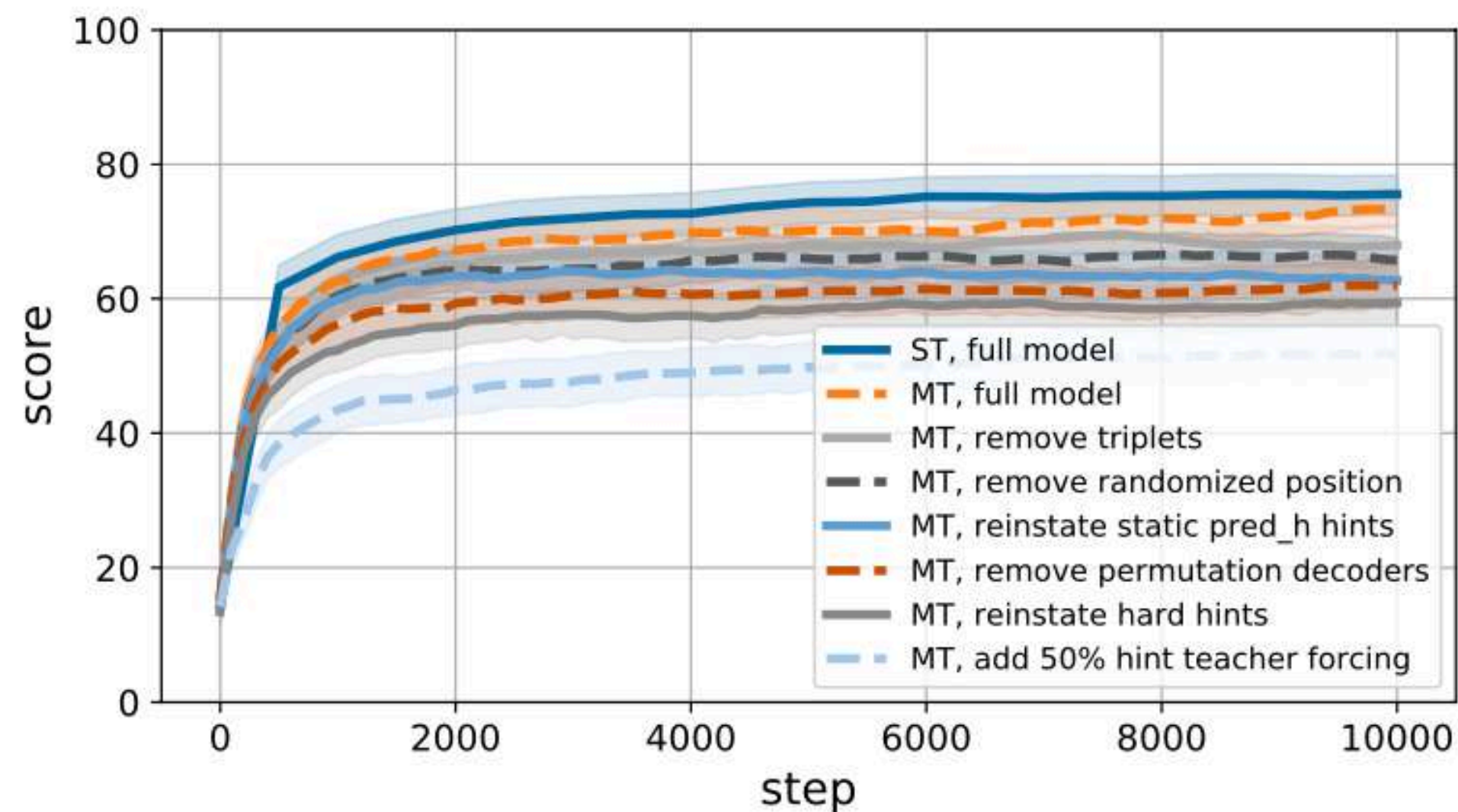
- Chunking
 - to reduce the memory footprint of multi-task training
 - trajectories are split along the time axis for gradient computation (16 steps per chunk)



Results



Results



(b) Cumulative ablation demonstrates the positive effect of model improvements on the final OOD performance.

Conclusion

Summary:

- presented a generalist neural algorithmic learner
- techniques on improving the dataset, optimisation and architectures

Future Work:

- Modification on the GNN architecture, data pipeline and loss functions

THANK YOU FOR LISTENING!

Presented by
Lee Guan Bo Ambrose
BASc (Applied AI)



UNIVERSITY OF
WATERLOO

DAVID R. CHERITON SCHOOL
OF COMPUTER SCIENCE



SCHOOL OF
**COMPUTING &
DATA SCIENCE**
The University of Hong Kong