The Limits of Compiler Optimizations in the Face of Deep Learning's Rapid Growth - A Survey

Sidharrth Nagappan
University of Cambridge
sn666@cam.ac.uk

Abstract

This survey studies the evolution of deep learning compiler optimisations from traditional rule-based approaches to advanced search-based methods and intelligent reinforcement learning-driven strategies. It also explores limitations, themes and open problems in the face of the growing complexity of neural network architectures and concludes with directions for future work to make this research area more robust. [1]

1 Introduction

The growing size and computational complexity of deep neural networks (DNNs), edge-computing and the production of custom hardware has made optimising deep learning models non-trivial for efficient deployments. Typically, a deep learning compiler converts a programmatically defined neural network from a high-level framework like PyTorch [15] or TensorFlow [2] into a Directed Acyclic Graph (DAG), representative of the flow of data through layers of the DNN. This graph is then converted to Intermediate Representations (IR) at multiple levels of granularity, that enable different hardware providers to read from. Optimisation is done at every level of granularity: from the macroscopic computation graph, through each IR stage, and all the way down to the hardware layer. Compilation infrastructure like Google's MLIR bridge high-level and low-level IRs in an approach to sharing optimisations between different hardware backends [12]. At the highest level, hardware-independent optimisations such as operator fusion are applied, before lowering (*translating*) to hardware-specific code (e.g. PTX for NVIDIA GPUs) [1, 8].

While lower-level optimisations are relatively niche, recent academic research has focused on *intelligently* exploring graph and operator substitutions to improve performance before the hardware specialisation phase. This survey will particularly focus on **high-level graph optimisations**, a rudimentary example of which is shown in Figure 1. Specifically, I will

1. Briefly explain rule-based graph substitutions, before deeply examining super-optimisation, tensor derivations and the recent advent of reinforcement learning in this domain.
2. Outline open problems in the literature that make this domain particularly challenging, and future work.

2 Evolution of Optimisations

A loose grouping of optimisation strategies is available in Table 1.

[1] Word count excluding abstract is 1918 (computed using *texcount*).

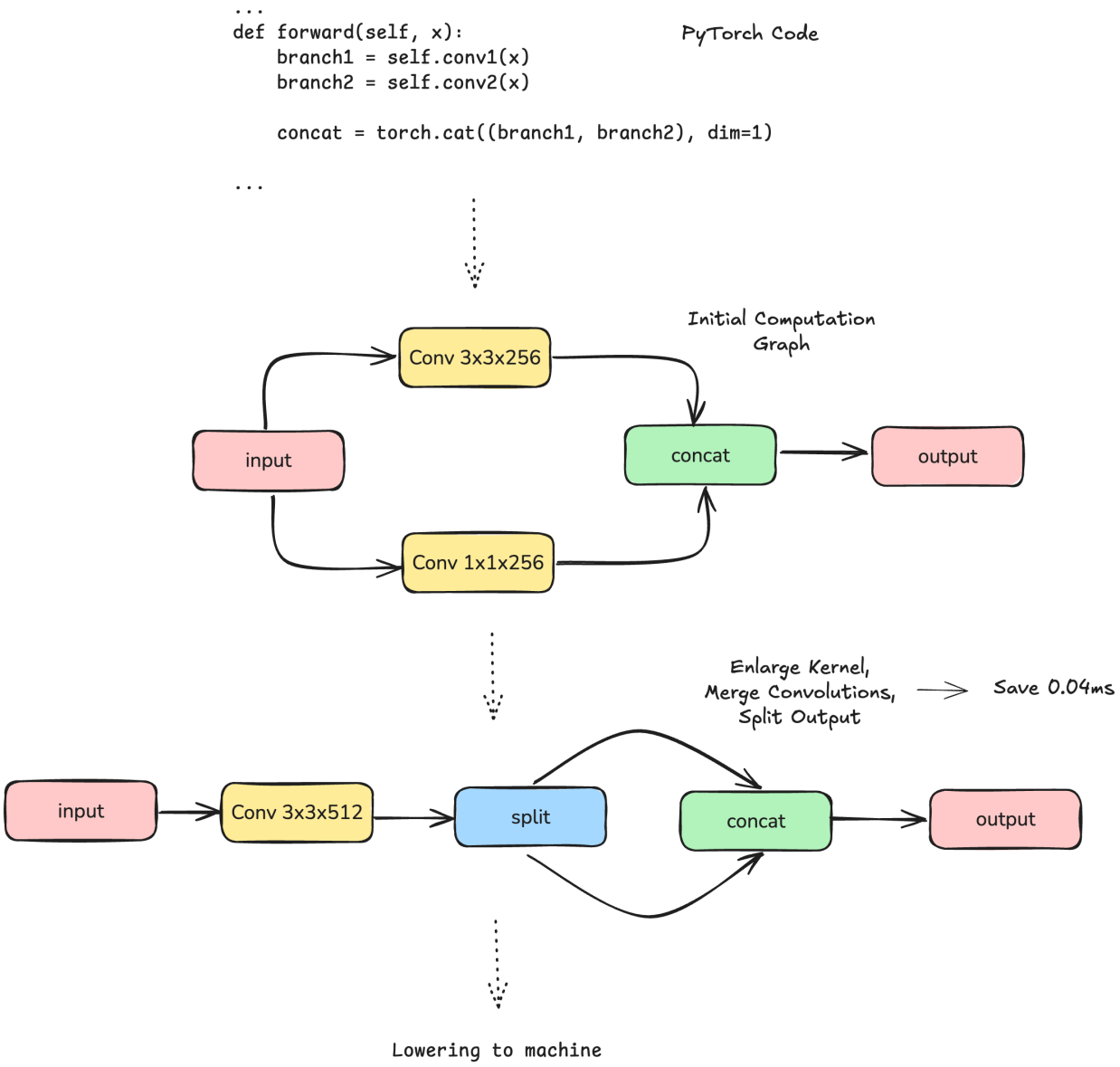

Figure 1: A Simple Convolution Re-Write by Merging Operations, my abstraction of an example from [4]

Strategy	Year	Cost Model	Search Algorithm
Rule-Based			
TensorRT [13]	2016	-	
XLA [2]	2017	-	
Super-optimisation			
TASO [10]	2019	$\sum$ operator execution times	Backtracking Search
PET [16]	2021	$\sum$ operator execution times	Adaptive Depth First Search
TENSAT [17]	2021	$\sum$ operator execution times	Integer Linear Programming (ILP)
Derivation-Based			
EinnEt [19]	2023	Arithmetic Intensity (FLOPs)	Distance from hardware supported operators
Reinforcement Learning			
RLFlow [14]	2022	Weighted Reward of Graph Runtime + Memory Access (based on TASO)	Model-Based RL *World Model (MDN-RNN)*
X-RLFlow [7]	2023	End-to-end Inference Latency	Model-Free RL *PPO*

Table 1: Loose Grouping of Optimisation Strategies

2.1 Rule-Based Optimisation

Traditional DNN frameworks such as TensorRT [13], XLA [2], and MetaFlow [9] employ pre-defined rules and fixed cost-based heuristics to simplify the final computation graph. Transformations such as constant folding (evaluating constant values in graphs at compile time), operator fusion (combining multiple operators to avoid storing intermediate results) and layout optimisation (to improve cache utilisation) are common. These approaches are fundamental, and dependable and have been used in traditional compilers long before the growth of deep learning. While straightforward to implement, these approaches rely on manually written collections of transformation rules. For instance, TVM has 15000 lines of code and TensorFlow has 53000 lines for template optimisations, so the depth of optimisations is directly proportional to the coverage of the compiler's rule-sets. In the face of sophisticated and evolving DNN architectures, these rule repositories become harder to maintain and eventually fail to produce sufficiently effective optimisations.

2.2 Super-optimisation and Graph Substitutions

Built on a starter set of known optimisations, the paradigm of "super-optimisation" traverses a search space of graph substitutions, aiming to replace subgraphs with functionally equivalent, but computationally cheaper alternatives. The search process is governed by a straightforward cost model, that is the sum of individual operator execution times, originally computed by profiling each operator on every permutation of hardware configurations. This paradigm is formally proven NP-hard, because of the interplay between different substitutions and the sheer combinatorial volume of substitution sequences; I refer the reader to [4] for the mathematical proof, where the authors reduce ordered substitutions to a max-flow problem.

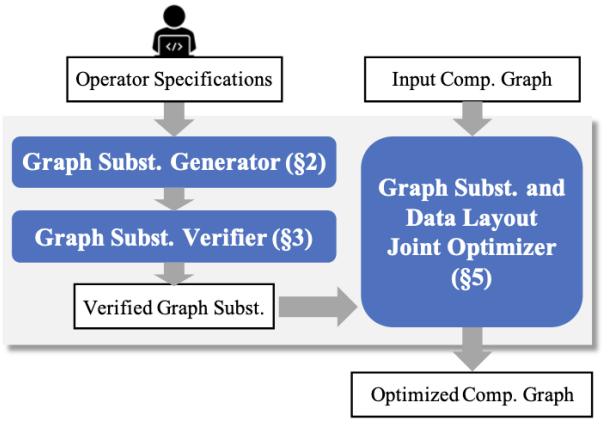

Figure 2: TASO's fundamental super-optimisation system design [10]

TASO (depicted in Figure 2) is the first to coin the term super-optimisation in this space, generating a space of potential sub-graph substitutions based on prior knowledge, and using the cost model above to rank, execute and explore optimal substitutions using a backtracking search [10]. Though not stated, I estimate the time complexity of the search to be $O(n^b)$, where b is the branching factor and d is the depth of the search tree. It employs formal verification methods to ensure that the generated rewrites are mathematically sound. While TASO strictly emphasised "full" functional equivalence in its optimisations, PET included partially equivalent transformations in its search space and empirically proved its utility, allowing potential substitutions that TASO would have ruled out [16]. To get around the discrepancies that partial equivalence would introduce, correction kernels are later employed to restore full functional equivalence. Since the ordeal of graph-rewriting is permutation-sensitive (as described above), where the order in which re-writes are applied affects the final performance, TENSAT then advocated for the use of equality saturation that applies e-graphs to simultaneously explore every possible re-write in a tree-like structure [17], with this approach further improved using Monte Carlo Tree Search in 2024 [6].

As seen above, after TASO established a common baseline for super-optimisation literature, later work used these foundations to either expand the search space or find more computationally optimal search strategies. Though the search space increases, TENSAT's optimisation time is fastest due to ILP's speed in smaller search spaces. Additionally, the complexity of understanding the whole "partial equivalence" paradigm saw later work hesitating to compare their methods against PET [7]; this comparability problem is discussed in greater detail in the last section. In terms of the global optimality of their final graphs, the four papers do not test their algorithms on the same DNNs, so it is unclear which method is collectively superior.

2.3 Derivation-Based Optimisations

Preceding work employed predefined operators and their transformations as a starting point, placing a constraint on the size of the optimisation space. EinnEt, depicted at a high-level in Figure 3, opted for a more granular representation of the problem by decomposing computation graphs into tensor algebra (TA) , creating an almost infinite search space of possible derivations. TA can be considered as a form of IR that is more atomic than computation graphs, but higher than machine code. Via standard mathematical derivations such as summation splitting and boundary tightening, algebraic expressions are simplified to more efficient, but functionally equivalent alternatives,

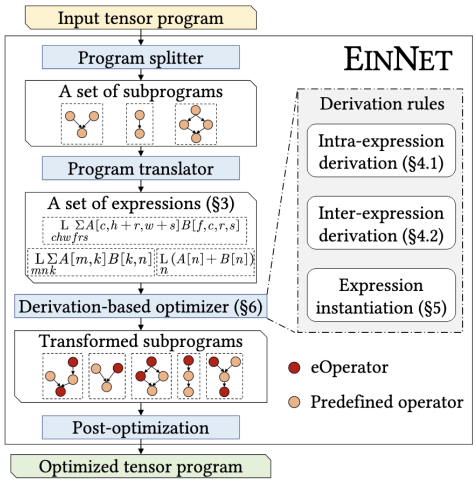

Figure 3: EinnEt at a high-level from [19]

similar to simplifying a secondary school algebraic equation. The vast search space is guided by expression distance, to either map the simplified form back to operators supported by existing kernels (e.g. TVM), or create new *eOperators* that represent novel, discovered optimisations tailored for a network. Limited newer optimisers have adopted this foundational TA paradigm, but it is nevertheless promising.

Proof : Optimising a Convolution

A convolution is atomically represented in tensor algebra (A input tensor and K kernel):

$$\mathcal{E}_1 = L \sum_{hwf} \sum_{crs} A[h+r, w+s, c] K[r, s, f, c]$$

summation is first split ($\sum_c$) to create re-usable intermediate tensors, reducing redundancy:

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

Refer to [19] for a full derivation.

The authors also highlight EinnEt's generalisation ability to unseen structures by using Longformer's sophisticated attention mechanism [3] as a case study. However, a larger search space induces significant computational overhead, and the results ambiguously quote search times of up to two hours, and write them off as one-off executions.

2.4 Advent of Reinforcement Learning

Reinforcement Learning (RL) has been making strides in system optimisation, offering an alternative to greedy search strategies. If a sequential decision-making problem (like graph optimisation) can be modelled as a Markov Decision Process, then RL can be applied. RLFlow used a World Model [5] to train an agent that performs simulated subgraph transformations on a graph whose

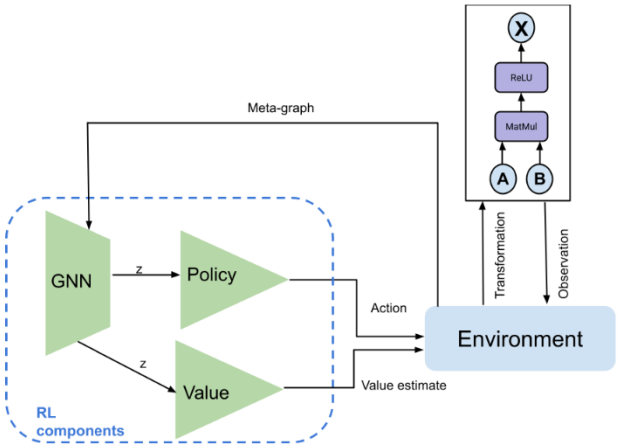

Figure 4: X-RLFlow Component Structure

representation is embedded via variational auto-encoders. Since the computation graph is in fact a "graph", Google used Graph Neural Networks (GNNs) for topological encoding [18], enabling granular node-level optimisations such as operator fusion, scheduling and device placement [20]. X-RLFlow then applied GNNs at a more macroscopic level for super-optimisation to select sequential graph-level re-writes using a policy network trained via Proximal Policy Optimisation, a model-free RL method (depicted in Figure 4).

A key advantage of RL is the ability to work with delayed rewards; in graph optimisation, the true impact of a transformation may not be apparent until the final product comes to fruition. This focus on the graph structure also enables learned representations to generalize across varying tensor shapes (such as in language models) without retraining, since the underlying topology remains unchanged. While greedy search algorithms (e.g. TASO's backtracking) prioritise immediate reductions in cost, RLFLow and X-RLFlow empirically locate more globally optimal solutions. This, however, comes at the cost of long training times and the computational expense of having to train on new DNNs. Also, performances of these RL-methods have only been benchmarked against TASO, without direct comparisons between the RL approaches themselves.

3 Open Problems, Themes & Future Work

Cost Model Sensitivity Many cost models that stem from TASO's preliminary definition of a computation graph's cost [10, 17] operate on the assumption that the sum of all operator runtimes will be the final inference latency. However, the difference is up to 24%, as reported by the authors of X-RLflow [7]. As search algorithms like PET start employing partial equivalence and introducing correction kernels, the final cost model becomes even more complicated because these new operators might offset the initial improvements. Some might argue that computing actual latencies is the generic solution, but this is too computationally expensive when the algorithm needs to compute this cost for every variation of the graph.

DNN Layers Are Increasingly Sophisticated The most significant problem in compiler optimisation is the growing complexity of new DNN layers, such as non-max suppression layers that cannot be expressed in a mathematical form, on which all the prior approaches operate. Most

compilers only focus on a set of known, supported layers; I notice that many only report results on common operations like convolutions. A more optimised convolution is good, but this is already a widely explored and optimised operator and fractional improvements do not mean much. Large organisations have dedicated compiler engineers to figure out how to compress a language model's computation graph, but this is still very manual and none of the mentioned strategies are widely used in production environments. While solutions like EinnEt are a step towards creating new custom optimisations for unseen operations, even this requires an algebraic breakdown, meaning more work is required in this area.

Need for Benchmarking Suite In traditional machine learning literature, there are standard datasets like CIFAR-10 [11] on which newer models benchmark performance. However, there is no such standard of DNNs in the ML Compiler literature, for the algorithms to measure their optimisations against. The common trend is papers cherrypicking architectures on which their algorithm performs favourably, or omitting essential details like exact model sizes (e.g. ResNet-18 and ResNet-50). For instance, EinnNet chose to report optimisations on the Longformer [19], but not for a simpler transformer like BERT, on which both PET and X-RLFlow benchmarked. The problem is exacerbated by the lack of comprehensive code documentation to accompany the papers.

Search Space Traversal From the backtracking search used by TASO, to EinnNet's algebraic search to TENSAT's equality saturation, subsequent methods keep widening the search space. While some form of distance guidance or pruning is applied, optimisations can still take up to several hours for sophisticated networks, and papers like EinnNet are quite ambiguous about how long the search actually takes. The fundamental problem lies in the trade-off between the completeness of a search and the time taken, with the question becoming when is enough and what a global optimum really means.

4 Conclusion

While many of the works expand the search space of potential optimisations, they each introduce their own complexities and overheads, that are critical considerations for actual production deployments. Furthermore, more work is required in realistic cost modelling, decomposing sophisticated DNN layers and establishing a common set of DNNs for fair benchmarking. Many papers argue that their work is orthogonal to their predecessors, and this orthogonality can be exploited by combining complementary techniques, such as RL [7] to guide equality saturation [17, 6], or tensor algebra [19] with partial equivalence [16] to leverage one another's strengths.

Above all, human compiler engineers are still the go-to for system-critical deployments, and we are still far from fully hiding the optimization process from high-level deep learning code [8], meaning scientists should be aware of computational costs when designing DNNs and human expertise should still evaluate any optimisations provided by the strategies in this survey.

References

[1] URL: `https://unify.ai/blog/deep-learning-compilers`.

[2] Martín Abadi et al. "TensorFlow: a system for large-scale machine learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 9781931971331.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: 2004.05150 [cs.CL]. URL: https://arxiv.org/abs/2004.05150.

[4] Jingzhi Fang et al. "Optimizing DNN computation graph using graph substitutions". In: *Proc. VLDB Endow.* 13.12 (July 2020), pp. 2734–2746. ISSN: 2150-8097. DOI: 10.14778/3407790.3407857. URL: https://doi.org/10.14778/3407790.3407857.

[5] David Ha and Jürgen Schmidhuber. "World Models". In: (2018). DOI: 10.5281/ZENODO.1207631. URL: https://zenodo.org/record/1207631.

[6] Jakob Hartmann, Guoliang He, and Eiko Yoneki. "Optimizing Tensor Computation Graphs with Equality Saturation and Monte Carlo Tree Search". In: *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques*. PACT '24. Long Beach, CA, USA: Association for Computing Machinery, 2024, pp. 40–52. ISBN: 9798400706318. DOI: 10.1145/3656019.3689611. URL: https://doi.org/10.1145/3656019.3689611.

[7] Guoliang He, Sean Parker, and Eiko Yoneki. *X-RLflow: Graph Reinforcement Learning for Neural Network Subgraphs Transformation*. 2023. arXiv: 2304.14698 [cs.LG]. URL: https://arxiv.org/abs/2304.14698.

[8] Chip Huyen. *A friendly introduction to machine learning compilers and Optimizers*. Sept. 2021. URL: https://huyenchip.com/2021/09/07/a-friendly-introduction-to-machine-learning-compilers-and-optimizers.html.

[9] Zhihao Jia et al. "Optimizing DNN Computation with Relaxed Graph Substitutions". In: *Proceedings of the 2nd Conference on Systems and Machine Learning (SysML'19)*. 2019.

[10] Zhihao Jia et al. "TASO: optimizing deep learning computation with automatic generation of graph substitutions". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. SOSP '19. Huntsville, Ontario, Canada: Association for Computing Machinery, 2019, pp. 47–62. ISBN: 9781450368735. DOI: 10.1145/3341301.3359630. URL: https://doi.org/10.1145/3341301.3359630.

[11] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009. URL: https://api.semanticscholar.org/CorpusID:18268744.

[12] Chris Lattner et al. "MLIR: scaling compiler infrastructure for domain specific computation". In: *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization*. CGO '21. Virtual Event, Republic of Korea: IEEE Press, 2021, pp. 2–14. ISBN: 9781728186139. DOI: 10.1109/CGO51591.2021.9370308. URL: https://doi.org/10.1109/CGO51591.2021.9370308.

[13] *NVIDIA TensorRT*. https://developer.nvidia.com/tensorrt/. Accessed: Dec 8, 2024.

[14] Sean Parker, Sami Alabed, and Eiko Yoneki. "RLFlow: Optimising Neural Network Subgraph Transformation with World Models". In: *ArXiv* abs/2205.01435 (2022). URL: https://api.semanticscholar.org/CorpusID:248505859.

[15] Adam Paszke et al. "PyTorch: an imperative style, high-performance deep learning library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[16] Haojie Wang et al. "PET: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections". In: *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, July 2021, pp. 37–54. ISBN: 978-1-939133-22-9. URL: https://www.usenix.org/conference/osdi21/presentation/wang.

[17] Yichen Yang et al. "Equality Saturation for Tensor Graph Superoptimization". In: *Proceedings of Machine Learning and Systems*. Ed. by A. Smola, A. Dimakis, and I. Stoica. Vol. 3. 2021, pp. 255–268. URL: https://proceedings.mlsys.org/paper_files/paper/2021/file/65ded5353c5ee48d0b7d48c591b8f430-Paper.pdf.

[18] Haiqi Zhang et al. "Semi-Supervised Classification of Graph Convolutional Networks with Laplacian Rank Constraints". In: *Neural Processing Letters* 54.4 (Aug. 2022), pp. 2645–2656. ISSN: 1573-773X. DOI: 10.1007/s11063-020-10404-7. URL: https://doi.org/10.1007/s11063-020-10404-7.

[19] Liyan Zheng et al. "EINNET: Optimizing Tensor Programs with Derivation-Based Transformations". In: *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, July 2023, pp. 739–755. ISBN: 978-1-939133-34-2. URL: https://www.usenix.org/conference/osdi23/presentation/zheng.

[20] Yanqi Zhou et al. "Transferable graph optimizers for ML compilers". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.