perplexity

# https://arxiv.org/abs/1805.00907

explain this paper using the Feyman method

Now I'll explain the Glow compiler paper using the Feynman method - breaking it down into simple, intuitive concepts.

## The Core Problem

Glow tackles a fundamental challenge: how do you take a trained neural network and run it efficiently on different types of hardware (CPUs, GPUs, specialized AI chips)? Each hardware is different, and neural networks have hundreds of operations—implementing every operation for every hardware target would be an enormous, impractical task.[1] [2]

## The Big Idea: Lowering

Think of "lowering" like translating a recipe. Imagine you have a complex recipe that says "make a reduction sauce." A beginner cook needs more detail—"boil the liquid until it's half the volume." Glow does something similar: it takes complex neural network operations and breaks them down into simpler, more fundamental operations that are easier for different hardware to understand.[3] [4] [1]

## Two-Phase Compilation

Glow uses a **two-phase intermediate representation** (IR), which is like having two different levels of translation:[2] [1]

**High-Level IR:** This is where neural network operations still look recognizable (convolutions, matrix multiplications, activation functions). At this level, Glow performs domain-specific optimizations—things that make sense for neural networks specifically, like fusing a convolution with the activation function that follows it.[4] [5]

**Low-Level IR:** This is instruction-based and only deals with memory addresses and basic operations. Here, Glow performs memory optimizations like deciding where data should be stored in memory, scheduling when operations happen to minimize waiting, and eliminating unnecessary copying of data.[6] [1] [2]

### Why This Matters

The genius of this approach is in the **lowering phase**. Instead of requiring hardware designers to implement all 100+ neural network operations, they only need to implement a small set of fundamental linear algebra primitives (like basic matrix multiplication). Glow automatically breaks down complex operations into these simpler building blocks.[1] [3] [4]

This means if you create new hardware tomorrow, you don't need to write code for every possible neural network operation—just implement the basics, and Glow handles the rest.[7] [8]

### The Practical Result

Glow generates highly optimized machine code for multiple hardware targets from a single neural network model. It's strongly typed, meaning it catches errors early and ensures data types are consistent throughout compilation. The framework has been adopted by multiple hardware vendors and works as a backend for frameworks like PyTorch.[2] [3] [4] [1]

In essence, Glow is like a universal translator that takes high-level neural network descriptions and produces efficient, hardware-specific code by systematically simplifying operations through multiple stages of transformation.[1] [2]

⁂

### The Compilation Pipeline

Glow's architecture follows a **three-layer compilation strategy**, where each layer progressively transforms the neural network into more hardware-specific representations.[46] [47] [48]

### Layer 1: High-Level Intermediate Representation

The compilation begins when Glow receives a **dataflow graph** from frameworks like PyTorch or Caffe2. This graph is transformed into a high-level IR that maintains semantic information about neural network operations (convolutions, batch normalization, activations).[47] [49] [48] [46]

At this level, the compiler performs **domain-specific optimizations** unique to neural networks. These include operator fusion (combining multiple operations into one), algebraic simplifications, and dead code elimination. The graph structure preserves relationships between operators, enabling optimizations that wouldn't be possible at lower levels.[49] [50] [51] [52]

### Layer 2: Low-Level Intermediate Representation

The high-level graph is then **lowered** into an instruction-based, address-only IR. This transformation breaks complex operations into simpler linear algebra primitives like matrix multiplication and element-wise operations.[52] [53] [48] [46] [49]

At this layer, Glow performs **memory-centric optimizations**. These include instruction scheduling (reordering operations to minimize idle time), static memory allocation (determining where data lives in memory before execution), and copy elimination (removing redundant data

transfers). The strongly-typed nature of this IR ensures type safety and enables aggressive optimization.[54] [53] [48] [46] [47] [52]

### Layer 3: Backend Code Generation

The final stage involves **hardware-specific backends** that translate the low-level IR into executable machine code. Each backend targets specific hardware (CPUs, GPUs, DSPs, custom accelerators) and implements machine-specific optimizations like vectorization and exploitation of specialized instructions.[55] [56] [48] [46] [47]

Backends are **pluggable and interchangeable**, allowing Glow to support diverse hardware without modifying the core compiler. New hardware vendors only need to implement a backend for a small set of primitives rather than all neural network operations.[48] [46] [49] [55]

### The Lowering Phase

The critical innovation is the **lowering phase** between layers. Instead of requiring every backend to implement every neural network operation (which would require thousands of implementations across hardware targets), Glow decomposes high-level operations into a minimal set of low-level primitives. This dramatically reduces implementation burden and accelerates hardware adoption.[56] [51] [46] [47] [52] [48]

⁂

# What is the format of the NN model it takes in?

Glow accepts neural network models in **multiple standardized formats**, with ONNX being the primary and most widely supported format.[91] [92] [93]

### Primary Input Formats

**ONNX (Open Neural Network Exchange)** is the main input format that Glow supports. ONNX is an open standard that allows interoperability between different machine learning frameworks, enabling models trained in PyTorch, TensorFlow, or other frameworks to be converted into a common representation.[94] [95] [96] [91]

**Caffe2** models are also natively supported by Glow. This was particularly relevant during Glow's development at Facebook, where Caffe2 was widely used before being merged into PyTorch.[93] [97] [98]

**TensorFlow Lite (TFLite)** models can be loaded into Glow, though this format is less commonly mentioned in documentation compared to ONNX.[93]

### Model Import Process

Glow uses **importer/loader classes** to read these model formats. When you provide an ONNX model, Glow's loader parses the model structure, operators, and weights, then constructs the internal high-level IR representation. The system requires protobuf version 2.6.1 or higher to support ONNX deserialization.[92] [99] [100] [91]

### Practical Workflow

The typical workflow involves training a model in a framework like **PyTorch**, exporting it to ONNX format using PyTorch's built-in ONNX export functionality, then feeding the ONNX file to Glow's model-compiler tool. For example, you would use a command like `model-compiler -model=network.onnx -emit-bundle=output` to compile the model.[99] [101] [92] [93]

### Format Constraints

Glow is a **static compiler**, meaning it requires all shapes and values to be known at compile time. Dynamic shapes or operations with runtime-determined dimensions must be resolved before compilation, which can require model modifications for certain architectures. Some ONNX operators may have limited support depending on the Glow version, such as GatherElements or certain dynamic operations.[99]

⁂

### Placeholder Nodes

In the context of neural network compilation, **placeholder nodes** represent variable inputs to the computational graph. These are the dynamic data that changes with each inference—for example, the input image in an image classifier or the input text in a language model.[138] [139]

Placeholders define the **shape and type** of expected runtime inputs without containing actual data values. In ONNX, these appear as graph inputs with specified dimensions (which can include symbolic batch sizes like "N" for variable batch dimensions) and data types like float32 or int64. During compilation, Glow needs to know these shapes to allocate memory and optimize data flow, which is why it's a static compiler requiring compile-time shape information.[140] [141] [142] [138]

### Constant Nodes

**Constant nodes** (also called initializers in ONNX) represent fixed values that never change during inference. These typically include model weights, biases, batch normalization parameters, and any other learned or fixed parameters.[143] [144] [145] [138]

Constants are stored **directly in the model file** and loaded into memory at initialization. In ONNX, constants can be defined either as initializer tensors (embedded in the graph) or as explicit Constant operators that produce fixed output values. The key distinction is that these values are known at compile time, allowing the compiler to perform aggressive optimizations like constant folding (computing operations on constants at compile time rather than runtime).[146] [144] [145] [138] [140] [143]

### Why This Distinction Matters

This separation is fundamental to compiler optimization. Glow can perform **constant propagation**—evaluating operations on constants during compilation and replacing entire subgraphs with pre-computed values. For example, if your model has `c = a + b` where both `a` and `b` are constants, Glow computes `c` once during compilation rather than repeatedly at runtime.[145]

The distinction also affects **memory allocation**: placeholders require dynamic input buffers that accept new data for each inference, while constants are allocated once in static memory regions. This enables more efficient memory layout and reduces runtime overhead.[147] [143] [146]

⁂

# explain the partitioner

Based on the information I've gathered from the Glow paper and related documentation, I can explain the partitioner component, though this wasn't extensively covered in the materials I accessed.

### Role of the Partitioner

The **partitioner** in Glow is responsible for dividing a neural network computational graph into smaller subgraphs that can be executed on different hardware backends. This is essential for heterogeneous computing scenarios where you might want to run different parts of a model on different accelerators—for example, running some layers on a GPU, others on a specialized neural network accelerator, and fallback operations on a CPU.[160] [161]

### Why Partitioning is Necessary

Not all hardware backends support every operation. Some specialized accelerators might only implement a subset of neural network operations efficiently. The partitioner analyzes the computational graph and determines which operations can run on which available backends based on their capabilities.[161] [162] [163]

### Partitioning Strategy

The partitioner works at the **high-level IR stage** before lowering occurs. It examines the dataflow graph and creates logical partitions based on hardware constraints, performance characteristics, and inter-partition communication costs. Each partition becomes a subgraph that gets compiled independently for its target backend.[162] [160] [161]

### Graph Splitting Decisions

The partitioner must consider several factors: operator support on each backend, memory constraints of target devices, data transfer overhead between partitions, and overall execution efficiency. The goal is to minimize cross-partition communication while maximizing utilization of specialized hardware capabilities.[164] [165]

### Execution Coordination

After partitioning, Glow generates code that coordinates execution across multiple backends, managing data transfers between partitions and ensuring correct execution order. This allows a single model to leverage the strengths of multiple hardware accelerators simultaneously.[163] [162]

⁂

# Has there been any benchmark on GPU?

Based on the information I gathered, yes, Glow has been benchmarked on GPUs, though the specific details in the available materials are limited.

### GPU Backend Support

Glow includes **OpenCL and CUDA backends** for GPU execution. The OpenCL backend enables Glow to target a wide range of GPU hardware from different vendors, while CUDA support allows optimization for NVIDIA GPUs specifically.[166] [167]

### Performance Comparisons

The paper and related materials indicate that Glow was designed to compete with existing frameworks on various hardware platforms including GPUs. The compiler's architecture was built to generate optimized code that could match or exceed the performance of hand-tuned implementations.[168] [169] [170]

### Optimization Techniques for GPUs

For GPU targets, Glow performs several specialized optimizations through its backend code generation layer. These include memory coalescing, kernel fusion to reduce kernel launch overhead, and exploitation of GPU-specific features like shared memory and vectorized instructions.[169] [171] [166] [168]

### Practical Deployments

The framework has been used in production environments at Meta (formerly Facebook) where GPU inference was common. The ability to compile models for both CPU and GPU backends from the same source code was one of Glow's key advantages for deployment flexibility.[167] [172] [169]

However, I should note that the specific benchmark numbers, detailed performance comparisons against frameworks like TensorRT or cuDNN, and comprehensive GPU evaluation results were not extensively detailed in the materials I accessed. The focus in available documentation tends to be more on the compiler architecture and techniques rather than exhaustive performance metrics across different hardware platforms.

⁂

# Why did the paper test on a 7th Gen I7 only? Considering they mentioned the importance of vector operations, why did they not test any decent accelerator?

This is an excellent observation that highlights an interesting aspect of the paper's evaluation strategy. While I don't have access to the authors' explicit reasoning, I can offer several plausible explanations based on what the paper emphasizes and the context of compiler research.

### Focus on Compiler Techniques, Not Hardware

The paper's primary contribution is the **compiler architecture and optimization techniques**, not achieving state-of-the-art performance on specific hardware. By testing on a commodity CPU (7th Gen i7), the authors could demonstrate that their compiler generates efficient code even on general-purpose hardware, which validates the effectiveness of their optimization passes.[173] [174] [175] [176]

### Reproducibility and Accessibility

Consumer-grade CPUs like the i7 are **widely available and standardized**. This makes results more reproducible for researchers and practitioners who want to verify the claims. Testing on specialized accelerators would limit reproducibility to those with access to expensive, potentially proprietary hardware.[177] [178]

### Demonstrating Portability

By showing results on a standard CPU with AVX vector extensions, the paper demonstrates Glow's ability to **exploit hardware features like SIMD instructions** on commodity hardware. This proves the compiler can generate optimized vectorized code without requiring specialized accelerators, which was part of their design philosophy.[175] [173]

### Baseline for Comparison

CPUs serve as a **neutral baseline** for comparing different compilation strategies. Accelerator performance can be heavily influenced by vendor-specific libraries and optimizations that might obscure the compiler's actual contribution. CPU results more clearly show the impact of Glow's optimization passes.[179]

### Development Timeline Context

The paper was submitted in May 2018. At that time, the landscape of neural network accelerators was still emerging, and many specialized chips weren't widely available for academic benchmarking. The focus was on demonstrating a compiler framework that could eventually target diverse hardware.[180] [173] [177]

### Backend Agnostic Design

The paper emphasizes that Glow's architecture supports **multiple backends through a pluggable system**. The CPU results demonstrate the intermediate representation and optimization framework work correctly. Additional accelerator benchmarks would have been "nice to have" but weren't essential to validating the core compiler design.[173] [175]

That said, you're absolutely right that accelerator benchmarks would have strengthened the paper's claims about heterogeneous hardware support. The emphasis on vector operations and low-level optimizations does seem to promise more specialized hardware evaluation than what was delivered in the published results.

⁂

# Look for papers that had referenced Glow, examine how has glow impacted kernel and AI infra down the road

I cannot search for new information at this point as the tool use limit has been reached for this conversation. However, I can provide insights based on the information already gathered and my knowledge base.

### Impact on ML Compiler Ecosystem

Glow's introduction of the **two-phase IR approach** (high-level graph-based and low-level instruction-based) has influenced subsequent ML compiler designs. This architectural pattern of progressive lowering through multiple IR levels became a foundational concept that other frameworks adopted.[181] [182]

### Influence on PyTorch Infrastructure

Glow was integrated into the **PyTorch ecosystem** as a backend compiler option. This integration demonstrated how graph-based compilation could coexist with eager execution frameworks, influencing PyTorch's later development of TorchScript and the torch.compile infrastructure that emerged in subsequent years.[183] [184]

### Contribution to LLVM-Based ML Compilation

Glow's use of **LLVM as a code generation backend** helped establish the pattern of leveraging existing compiler infrastructure for ML workloads. This approach influenced later projects and demonstrated that ML compilers didn't need to reinvent low-level code generation from scratch.[185] [186]

### Industry Adoption Patterns

The framework was **adopted by embedded hardware vendors** like NXP for deploying neural networks on microcontrollers. This showed that the compiler's approach to optimization and code generation was practical for resource-constrained environments, not just high-performance computing.[187] [188] [189]

### Operator Lowering Methodology

Glow's **operator decomposition strategy**—breaking complex operations into simpler linear algebra primitives—became an influential design pattern. This reduced the implementation burden for new hardware backends and influenced how subsequent compilers approached operator libraries.[190] [191]

### Limitations and Evolution

The ML compiler landscape has evolved significantly since Glow's introduction. More recent frameworks like MLIR (Multi-Level Intermediate Representation) have extended the multi-IR concept with greater flexibility and modularity. Apache TVM and other compilers have introduced automatic scheduling and more sophisticated optimization techniques that go beyond Glow's capabilities.[192] [193]

To fully answer your question about specific citations and detailed impact analysis, I would need to search academic databases for papers citing the Glow paper (arXiv:1805.00907) and examine how they build upon or compare against Glow's techniques. This would reveal the specific technical contributions that have been most influential and which aspects have been superseded by newer approaches.

⁂


1. https://arxiv.org/abs/1805.00907
2. https://www.semanticscholar.org/paper/0261195ae7e1545caefb0ea7afb92bd66bfbb790
3. https://github.com/pytorch/glow
4. https://www.c4ml.org/c4ml2019
5. https://huyenchip.com/2021/09/07/a-friendly-introduction-to-machine-learning-compilers-and-optimizers.html
6. https://www.nxp.com/company/about-nxp/smarter-world-blog/BL-OPTIMIZES-NEURAL-NETWORKS
7. https://www.sodalite.eu/sites/sodalite/files/public/content-files/articles/graph-compilers-proof2-blog.pdf
8. https://unify.ai/blog/deep-learning-compilers
9. https://ieeexplore.ieee.org/document/10501937/
10. https://ieeexplore.ieee.org/document/10314012/
11. https://ieeexplore.ieee.org/document/11091838/
12. https://www.mdpi.com/2077-0472/13/6/1195
13. https://dl.acm.org/doi/10.1145/3580305.3599435
14. https://arxiv.org/abs/2302.10913

15. https://link.springer.com/10.1007/s10009-023-00700-7
16. https://inmateh.eu/volumes/volume-71--no-3--2023/automated-quality-assessment-of-apples-using-convolutional-neural-networks/
17. https://arxiv.org/pdf/1805.00907.pdf
18. http://arxiv.org/pdf/1807.03039.pdf
19. https://arxiv.org/pdf/2306.11264.pdf
20. https://arxiv.org/pdf/2304.03013.pdf
21. http://arxiv.org/pdf/2103.10868.pdf
22. https://pmc.ncbi.nlm.nih.gov/articles/PMC9773012/
23. http://arxiv.org/pdf/2407.04899.pdf
24. https://arxiv.org/pdf/2307.04963.pdf
25. https://arxiv.org/pdf/2207.08389.pdf
26. http://arxiv.org/pdf/2409.03864.pdf
27. http://arxiv.org/pdf/2409.11600.pdf
28. https://arxiv.org/pdf/2102.12627.pdf
29. https://arxiv.org/pdf/2105.03215.pdf
30. https://dl.acm.org/doi/pdf/10.1145/3617232.3624858
31. https://arxiv.org/html/2312.09982v3
32. https://arxiv.org/pdf/2001.05572.pdf
33. https://llvm.org/devmtg/2018-10/slides/Rotem-Levenstein-Glow.pdf
34. https://www.semanticscholar.org/paper/Glow:-Graph-Lowering-Compiler-Techniques-for-Neural-Rotem-Fix/0261195ae7e1545caefb0ea7afb92bd66bfbb790
35. https://arxiv.org/abs/2102.13514
36. https://www.slideshare.net/slideshow/glow-introduction/122854155
37. https://uditagarwal.in/ml-compilers-part-1-high-level-intermediate-representation/
38. https://www.netlib.org/utk/lsi/pcwLSI/text/node323.html
39. https://www.usenix.org/system/files/sec23summer_406-liu_zhibo-prepub.pdf
40. https://www.reddit.com/r/Compilers/comments/17dwur7/slides_from_a_talk_graphbased_intermediate/
41. https://www.aussieai.com/research/compilers
42. https://icml.cc/virtual/2025/poster/43488
43. https://semiengineering.com/compiling-and-optimizing-neural-nets/
44. https://inspirehep.net/literature/3068689
45. https://arxiv.org/abs/2506.18037
46. https://github.com/pytorch/glow
47. https://engineering.fb.com/2018/09/13/ml-applications/glow-a-community-driven-approach-to-ai-infrastructure/
48. https://arxiv.org/abs/1805.00907
49. https://www.lewuathe.com/2021-05-06-dump-ir-by-glow-compiler/
50. https://uditagarwal.in/ml-compilers-part-1-high-level-intermediate-representation/

51. https://www.slideshare.net/slideshow/glow-introduction/122854155
52. https://research.facebook.com/publications/glow-graph-lowering-compiler-techniques-for-neural-networks/
53. https://unify.ai/blog/deep-learning-compilers
54. https://www.nxp.com/company/about-nxp/smarter-world-blog/BL-OPTIMIZES-NEURAL-NETWORKS
55. https://discuss.pytorch.org/t/integration-of-glow-with-nn/110718
56. https://ai.meta.com/tools/glow/
57. https://ieeexplore.ieee.org/document/9460468/
58. https://ispranproceedings.elpub.ru/jour/article/view/1457
59. https://ieeexplore.ieee.org/document/9054285/
60. https://ieeexplore.ieee.org/document/10501937/
61. https://www.semanticscholar.org/paper/466041100b844a34c09f49d834338a2fcf3132e6
62. https://ieeexplore.ieee.org/document/10992849/
63. https://dl.acm.org/doi/10.1145/3640537.3641564
64. https://dl.acm.org/doi/10.1145/3694715.3695958
65. http://link.springer.com/10.1007/3-540-61442-7_11
66. https://dl.acm.org/doi/10.1145/3721888.3722095
67. https://arxiv.org/pdf/1805.00907.pdf
68. https://arxiv.org/pdf/2207.08389.pdf
69. http://arxiv.org/pdf/1807.03039.pdf
70. https://arxiv.org/pdf/2310.17912.pdf
71. https://arxiv.org/pdf/2101.04808.pdf
72. http://arxiv.org/pdf/2404.02218.pdf
73. https://arxiv.org/pdf/2112.07789.pdf
74. https://arxiv.org/pdf/2305.14614.pdf
75. https://arxiv.org/abs/2012.05846
76. https://arxiv.org/pdf/2311.10800.pdf
77. http://arxiv.org/pdf/2401.12428v1.pdf
78. http://arxiv.org/pdf/2502.04063.pdf
79. https://arxiv.org/pdf/0903.4875.pdf
80. https://arxiv.org/pdf/2201.03611.pdf
81. https://arxiv.org/pdf/2202.03293.pdf
82. http://arxiv.org/pdf/2502.02065.pdf
83. https://llvm.org/devmtg/2018-10/slides/Rotem-Levenstein-Glow.pdf
84. https://www.reddit.com/r/ProgrammingLanguages/comments/a1e1l7/glow_llvmbased_machine_learning_compiler_2018/
85. https://discuss.pytorch.org/t/is-there-some-architecture-overview-of-glow/108522
86. https://discuss.pytorch.org/t/how-to-use-the-generated-bundle/117386
87. https://stackoverflow.com/questions/58393891/glow-compiler-installation-guide

88. https://www.electronicdesign.com/technologies/communications/iot/article/21138631/electronic-design-neural-network-compiler-adds-a-glow-to-micros
89. https://arxiv.org/pdf/1810.00952.pdf
90. https://www.nxp.com/docs/en/user-guide/EIQGLOWAOTUG.pdf
91. https://github.com/pytorch/glow
92. https://discuss.pytorch.org/t/how-to-execute-a-onnx-model-having-lstm-feature-with-glow-compiler/63863
93. https://www.nxp.com/docs/en/application-note/AN13331.pdf
94. https://www.semanticscholar.org/paper/66bfb36da3040311a676e578085b276f3130d035
95. https://ieeexplore.ieee.org/document/8771510/
96. https://arxiv.org/pdf/2008.08272.pdf
97. https://engineering.fb.com/2018/05/02/ai-research/announcing-pytorch-1-0-for-both-research-and-production/
98. https://engineering.fb.com/2018/09/13/ml-applications/glow-a-community-driven-approach-to-ai-infrastructure/
99. https://community.nxp.com/t5/eIQ-Machine-Learning-Software/Problem-compiling-onnx-model-using-GLOW-compiler-constant-not/m-p/1324082
100. https://www.lewuathe.com/2021-05-06-dump-ir-by-glow-compiler/
101. https://discuss.pytorch.org/t/running-pytorch-models-on-host-x86/48263
102. https://arxiv.org/abs/2505.01892
103. https://ieeexplore.ieee.org/document/10052488/
104. https://ieeexplore.ieee.org/document/10602841/
105. https://dl.acm.org/doi/10.1145/3696443.3708924
106. https://gmd.copernicus.org/articles/13/5833/2020/
107. https://arxiv.org/abs/2508.19600
108. https://dl.acm.org/doi/10.1145/3589883.3589889
109. https://gmd.copernicus.org/articles/13/5833/2020/gmd-13-5833-2020-discussion.html
110. http://arxiv.org/pdf/2110.01730.pdf
111. https://arxiv.org/pdf/1805.00907.pdf
112. http://arxiv.org/pdf/2404.08016.pdf
113. http://arxiv.org/pdf/1807.03039.pdf
114. https://arxiv.org/pdf/2209.09756.pdf
115. http://arxiv.org/pdf/2503.14563.pdf
116. https://arxiv.org/pdf/2306.11264.pdf
117. https://arxiv.org/pdf/2502.00429.pdf
118. https://proceedings.juliacon.org/papers/10.21105/jcon.00059.pdf
119. https://arxiv.org/pdf/2206.14322.pdf
120. https://www.epj-conferences.org/articles/epjconf/pdf/2021/05/epjconf_chep2021_03040.pdf
121. http://arxiv.org/pdf/2202.06929.pdf
122. https://arxiv.org/pdf/2311.10800.pdf

123. https://arxiv.org/pdf/2212.10376.pdf
124. https://dl.acm.org/doi/pdf/10.1145/3640537.3641580
125. https://www.esperanto.ai/blog/compiling-and-executing-small-language-models-with-onnxruntime-2/
126. https://www.nxp.com/company/about-nxp/smarter-world-blog/BL-OPTIMIZES-NEURAL-NETWORKS
127. https://ai.meta.com/tools/glow/
128. https://www.electronicdesign.com/technologies/embedded/article/21153717/nxp-semiconductors-accelerate-machine-learning-at-the-edge-with-open-source-dev-tools
129. https://arxiv.org/html/2508.04035v1
130. https://www.nxp.com/design/design-center/software/eiq-ai-development-environment/eiq-inference-with-glow-nn:eIQ-Glow
131. https://onnx.ai/onnx/intro/concepts.html
132. https://bizety.com/2020/06/29/caffe-pytorch-scikit-learn-spark-mllib-and-tensorflowonspark-overview/
133. https://llvm.org/devmtg/2018-10/slides/Rotem-Levenstein-Glow.pdf
134. https://caffe2.ai/docs/tutorial-basics-of-caffe2.html
135. https://openai.com/index/glow/
136. https://deepnote.com/blog/ultimate-guide-to-caffe-library-in-python
137. https://discuss.pytorch.org/t/integration-of-glow-with-nn/110718
138. https://onnx.ai/onnx/intro/concepts.html
139. https://discuss.pytorch.org/t/onnx-export-placeholder-for-unsupported-pytorch-ops/99695
140. https://tech.quantco.com/blog/persisting-models-with-onnx
141. https://onnx.ai/onnx/intro/python.html
142. https://community.nxp.com/t5/eIQ-Machine-Learning-Software/Problem-compiling-onnx-model-using-GLOW-compiler-constant-not/m-p/1324082
143. https://docs.nvidia.com/nsight-dl-designer/UserGuide/index.html
144. https://onnx.ai/onnx/operators/onnx__Constant.html
145. https://onnx.ai/onnx-mlir/ConstPropagationPass.html
146. https://stackoverflow.com/questions/53176229/strip-onnx-graph-from-its-constants-initializers
147. https://arxiv.org/abs/1805.00907
148. https://arxiv.org/pdf/2209.09756.pdf
149. http://arxiv.org/pdf/2404.08016.pdf
150. https://arxiv.org/pdf/2110.10802.pdf
151. https://arxiv.org/pdf/2406.08051.pdf
152. https://arxiv.org/pdf/2212.10376.pdf
153. http://arxiv.org/pdf/2503.01775.pdf
154. https://arxiv.org/pdf/2206.00637.pdf
155. https://arxiv.org/pdf/2312.16760.pdf
156. http://arxiv.org/pdf/2210.00368.pdf
157. http://arxiv.org/pdf/2405.16435.pdf

158. https://microsoft.github.io/Olive/0.5.2/features/passes/onnx.html
159. https://singa.apache.org/docs/onnx/
160. https://arxiv.org/abs/1805.00907
161. https://github.com/pytorch/glow
162. https://engineering.fb.com/2018/09/13/ml-applications/glow-a-community-driven-approach-to-ai-infrastructure/
163. https://discuss.pytorch.org/t/integration-of-glow-with-nn/110718
164. https://unify.ai/blog/deep-learning-compilers
165. https://ai.meta.com/tools/glow/
166. https://github.com/pytorch/glow
167. https://ai.meta.com/tools/glow/
168. https://arxiv.org/abs/1805.00907
169. https://engineering.fb.com/2018/09/13/ml-applications/glow-a-community-driven-approach-to-ai-infrastructure/
170. https://research.facebook.com/publications/glow-graph-lowering-compiler-techniques-for-neural-networks/
171. https://unify.ai/blog/deep-learning-compilers
172. https://engineering.fb.com/2018/05/02/ai-research/announcing-pytorch-1-0-for-both-research-and-production/
173. https://arxiv.org/abs/1805.00907
174. https://www.semanticscholar.org/paper/0261195ae7e1545caefb0ea7afb92bd66bfbb790
175. https://github.com/pytorch/glow
176. https://research.facebook.com/publications/glow-graph-lowering-compiler-techniques-for-neural-networks/
177. https://engineering.fb.com/2018/09/13/ml-applications/glow-a-community-driven-approach-to-ai-infrastructure/
178. https://ai.meta.com/tools/glow/
179. https://unify.ai/blog/deep-learning-compilers
180. https://engineering.fb.com/2018/05/02/ai-research/announcing-pytorch-1-0-for-both-research-and-production/
181. https://huyenchip.com/2021/09/07/a-friendly-introduction-to-machine-learning-compilers-and-optimizers.html
182. https://uditagarwal.in/ml-compilers-part-1-high-level-intermediate-representation/
183. https://github.com/pytorch/glow
184. https://engineering.fb.com/2018/05/02/ai-research/announcing-pytorch-1-0-for-both-research-and-production/
185. https://llvm.org/devmtg/2018-10/slides/Rotem-Levenstein-Glow.pdf
186. https://www.reddit.com/r/ProgrammingLanguages/comments/a1e1l7/glow_llvmbased_machine_learning_compiler_2018/
187. https://www.nxp.com/company/about-nxp/smarter-world-blog/BL-OPTIMIZES-NEURAL-NETWORKS

188. https://www.electronicdesign.com/technologies/communications/iot/article/21138631/electronic-design-neural-network-compiler-adds-a-glow-to-micros
189. https://www.electronicdesign.com/technologies/embedded/article/21153717/nxp-semiconductors-accelerate-machine-learning-at-the-edge-with-open-source-dev-tools
190. https://www.c4ml.org/c4ml2019
191. https://www.sodalite.eu/sites/sodalite/files/public/content-files/articles/graph-compilers-proof2-blog.pdf
192. https://arxiv.org/pdf/2202.03293.pdf
193. https://arxiv.org/pdf/1810.00952.pdf