Survey on Distributed Neural Networks and GPU Programming Frameworks

Razvan Florian Vasile
Department of Computer Science
University of Bologna
razvanflorian.vasile@studio.unibo.it

Abstract

This paper presents a systematic literature review of distributed and parallel techniques for running deep neural networks on multiple machines and GPUs. It is composed of three parts: 1) a review of the available libraries that enable distributed training across GPU clusters, 2) a review of the most popular frameworks that facilitate parallelizing the training process on GPUs, and 3) a practical section that demonstrates a proof-of-concept implementation of the training process using common frameworks, namely PyTorch DDP and cuDNN.

The review synthesizes research from the past decade, examining various approaches to distributed training, their effectiveness, and implementation challenges. The work is aimed at students and practitioners, with the goal to provide an introduction to the topic and help frame a general idea of the most common libraries in each domain.

The distributed experiments use data parallelism to accelerate the training process, while the GPU experiments use cuDNN, cuBLAS and manual kernel implementations to train a small network. The effectiveness of each approach is demonstrated and to aid reproduction and experimentation Docker environments are provided. This allows to simulate a multi-GPU setup on a single Nvidia GPU, promoting ease of use by not relying on cloud services. The repository is available at https://github.com/atomwalk12/deep-bridge-survey.

1. Introduction

Deep Neural Networks. DNNs have powered a wide range of applications in areas such as computer vision, natural language processing, audio processing, graph knowledge representation and time series analysis [86]. One of the key driving factors that enable the application of deep learning models in such a wide range of domains, while still ensuring impressive performance, is the ability to scale raw computational power and datasets to impressive amounts [8, 26, 54]. Distributed Neural Networks (DNNs) provide techniques that enable scaling both these factors, effectively enabling models to learn complex relationships and excel in a wide range of domains.

GPU Programming. At the heart of distributed training stand frameworks that enable efficient calculations of matrix multiplication operations in parallel through the use of specialized GPUs. At the dawn of the machine learning revolution, AlexNet [53] (also called ConvNet), showed that deep learning models can effectively outperform human-level performance on the ImageNet dataset. They used two GTX 580 GPUs to train a network over a period of five to six days. The network achieved top-1 error rate of 37.5%, outperforming previous state-of-the-art results at the time, while their success sparked significant interest in the community and spurred a wave of research.

Importance of the Topic. It follows that the effectiveness of the deep neural networks is influenced by breakthroughs in both domains, which significantly influence both research and industry applications [16, 88]. To help create a more unified view of each area, in this study I will highlight the most significant frameworks that helped shape the landscape of each domain, identifying trends over a 13 year timeline. Morever, to ease-out the introduction and provide a baseline for getting started, two end-to-end experiments are implemented. These experiments will hopefully facilitate learning by reducing the steep learning curve through concrete examples, effectively providing a more intuitive understanding of the topics.

Paper Organization. The remainder of this paper is organized as follows: Section 2 introduces similar work already present in the literature, Section 3 presents this study's systematic review protocol documenting the search process, Section 4 describes the methodology used to identify key information from the selected papers, Section 5 presents the identification of the relationships between the two topics, Section 6 briefly presents the experiments used to evaluate the selected frameworks, Section 7 synthesizes the findings to answer the research questions, and finally Section 8 concludes by highlighting future work.

Table 1. A summary of existing surveys on the two topics.

Survey Name	Survey Description
Deep Learning frameworks for large-scale data mining [62]	This study emphasizes the connection between deep learning and massively parallelism to efficiently handle Big Data computations.
From Distributed Machine to Distributed Deep Learning [26]	This study emphasizes the connections between distributed machine learning and traditional machine learning methods.
Hitchhiker's Guide On Distributed Training of Deep NNs [16]	This study focuses on the relevant implementation details for training distributed neural networks. It details how data and model parallelism work.
Survey on Distributed Deep Learning Frameworks for Big Data [9]	The review compares distributed deep learning approaches based on parallelism techniques, hardware support and framework compatibility.

2. Related Work

For documenting the review process, this study follows primarily the guidelines laid out in [83], however advice for conducting the review is synthesized from a wider range of related articles [13, 14, 27, 50].

A summary of existing work in the field is summarized in Table 1. Our study differs from previous surveys in the following aspects:

First, concerning DNNs, related work focuses on techniques and algorithms for training models across multiple machines [9, 16, 26], where themes such as data and model parallelization techniques and communication protocols are explored. As a result, existing literature tackles architectural patterns and design choices, none focusing explicitly on providing a broad review of the available frameworks.

Another relevant article [62] aims to provide an overview of massive parallel frameworks available for deep learning. It describes specialized tools for hardware accelerators (GPUs, FPGAs, TPUs), however does not focus on auxiliary libraries for linear algebra, numerical computing and GPU communication. These libraries are relevant since they are actively used by other higher level frameworks which provide optimized primitives (i.e. cuDNN, MIOpen, oneDNN, etc.).

Secondly, although existing repositories do provide examples on how to use the CUDA library [21], as well as DDP [68], none provide end-to-end implementations that would allow the community to build non-trivial applications. This study aims to answer both of these concerns.

3. Research method

Multiple studies emphasize that a literature survey should be both transparent and replicable [28, 83], as this can ensure that reviewer bias is minimized. Generally, a literature survey is conducted by a group of people working together to assess the literature and involves multiple iterations. In this paper, steps have been taken to mitigate reviewer bias, nonetheless it is acknowledged that it is not possible to eliminate it completely. To ensure a fair comparison, the process is documented and the resulting artifacts are shown.

The main workflow is displayed in Figure 1 where the key phases are annotated as follows: getting started (M.1, M.2), planning the review (M.3, M.4), conducting the review (M.5, M.6), and reporting the results (M.7, M.8).

M.1 – The need for a survey A comprehensive survey bridging general frameworks and their practical application to these two topics is currently absent from the literature, as demonstrated in Section 2. This survey addresses this gap by not only discussing theoretical frameworks but also providing practical, end-to-end implementations, making it a valuable resource for both students and practitioners.

Learning outcomes. Conducting a review is a good way of learning about the two topics. By following a systematic approach, it will become easier to find relevant frameworks and research publications in the future, which is a skill that can be applied to other domains as well.

M.2 – Getting started The main goal is to analyze parallelization frameworks in DNNs and GPU programming. This entails two distinct constraints: (i) consider only frameworks that introduce GPU programming and DNN frameworks and (ii) include primary studies.

3.1. M.3 – Selection of Relevant Studies

This step identified relevant papers and is composed of two distinct processes: (i) S.1. literature survey on DNN frameworks and (ii) S.2. literature survey on GPU programming libraries.

3.1.1. S.1 – Literature Survey on DNN libraries

The steps involved are shown in Figure 2.

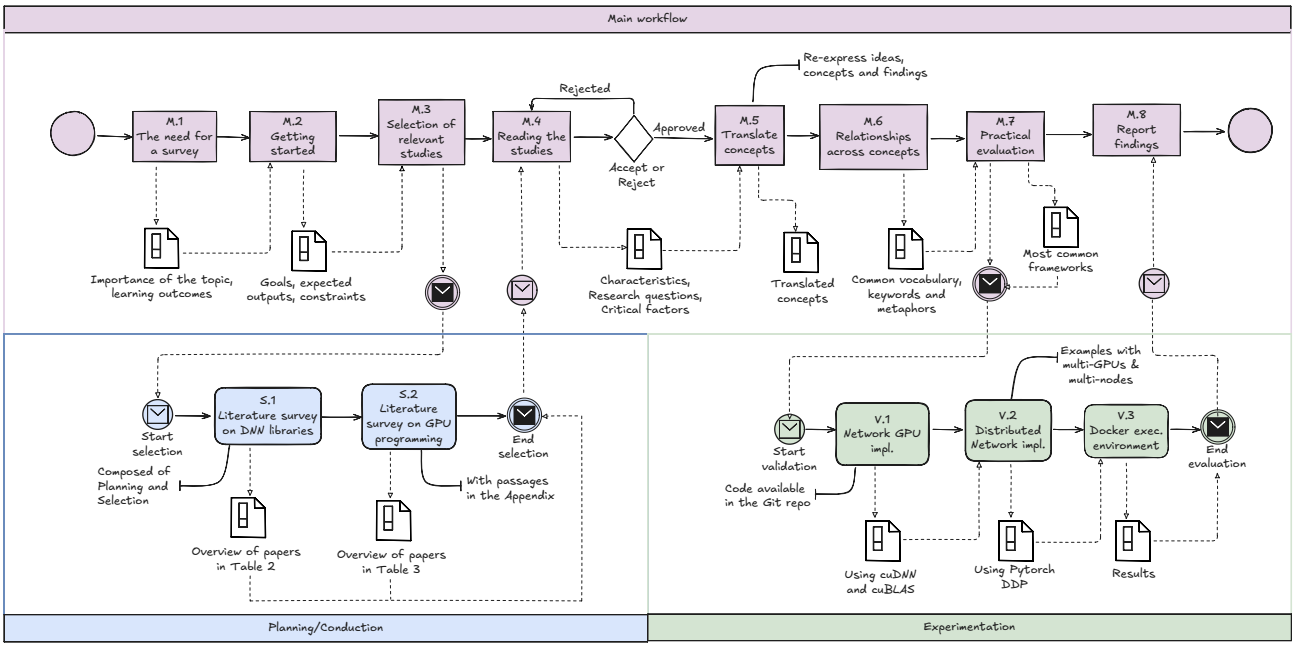


Figure 1. The workflow is divided into three main phases: main workflow (top), studies selection (bottom left), and evaluation (bottom right). Dashed lines indicate documentation and communication flows.

S.1.1 – Problem definition. In order to identify relevant studies, I conducted a secondary study[1]. This decision was made in the problem definition step (S.1.1). To ensure a methodical approach, this section defines a research protocol which formally defines the key attributes of the search process. This basically entails defining the research questions and expanding on the search strategy.

S.1.2 – Research questions. After an initial literature review, the research questions (RQ) are defined using the guidelines from [51] and [83]. Specifically, the PICOC (Population, Intervention, Comparison, Outcome, Context) criterion is used to write the questions into a format that ensures them to be specific, measurable and well-defined.

- **RQ$_1$:** What are the most commonly cited frameworks for distributed neural network training, and how do their communities vary in size?
  Rationale: By identifying the most common frameworks, we can trace the years in which they were published and form a unified timeline of the evolution of the field.
- **RQ$_2$:** What are the most frequently cited frameworks for GPU programming, and how do their communities differ in size?
  Rationale: By identifying the most common frameworks, we can identify which are the gaps in the literature they cover and which are the most promising areas for future research.
- **RQ$_3$:** Which are the overlaps and shared limitations of these areas?
  Rationale: By identifying the overlaps, this can lead to a more comprehensive understanding of the field.
- **RQ$_4$:** How can these technologies be applied in practice?
  Rationale: This can yield hands-on experience on the topic which is helpful for practical applications.

S.1.3 – Search Strategy. The search strategy represents a systematic approach for identifying relevant studies that adequately answer the research questions.

Databases. The process involves a manual search of three citation databases – Scopus, Semantic Scholar and arXiv[2] – that include conference proceedings and journal papers, considering three metadata fields (title, abstract, and keywords).

Inclusion/Exclusion Criteria. There were defined three inclusion criteria (IC) and three exclusion criteria (EC). In particular, I decided to select only primary studies, however secondary studies were mentioned in Section 2. The identification of secondary studies is useful as it allows to synthesize evidence and can make it possible to access primary studies:

[1] A secondary study synthesizes primary papers to provide a comprehensive overview. This contrasts with tertiary studies which analyze secondary studies.

[2] Other relevant databases that could have been used include: IEEE Xplore, ACM Digital Library and Science Direct.

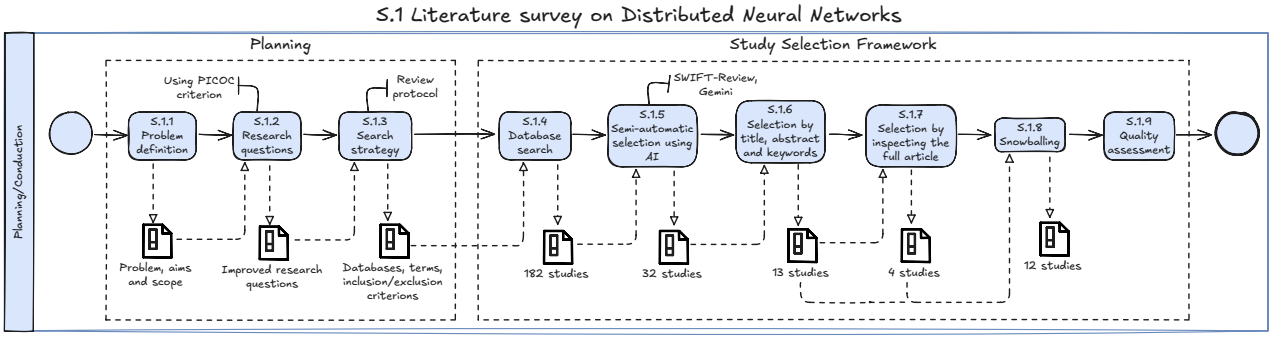


Figure 2. The diagram shows the series of steps carried out in the planning and study selection phases for the DNNs survey.

.

- **IC$_1$**: Study is a primary study.
- **IC$_2$**: Study addresses distributed frameworks in DL.
- **IC$_3$**: Study introduces a library or a framework.

- **EC$_1$**: Study does not discuss implementation details.
- **EC$_2$**: Study is not a primary study.
- **EC$_3$**: Study is not written in English.

Search terms. In Step S.1.3, I inquired about the literature using the following search string:

("machine learning" OR "deep learning") AND ("Data parallelism" OR "model parallelism") AND ("framework" OR "implementation")

Publication year criteria. For the DDL task, papers were considered between the time period 2015-2024. The start year (2015) was chosen due to being the year where there was a shift towards resource conservation, which resulted in a focus on parallelization techniques [8]. This is the year by which the effectiveness of deep learning algorithms was more widely recognized and more research was published that focused on scalability.

S1.4 - S1.5 – Semi-Automatic selection. After an initial database search (Step S1.4), 182 studies were retrieved. Subsequently, as part of S.1.5, two methods were applied for reducing the number of studies to a more manageable set. Below, I describe each approach, including my evaluation regarding their effectiveness.

Swift-Review. As suggested by [11], I applied Swift-Review [40], a machine learning classifier to filter out irrelevant studies (Step S.1.5). The technique is semi-automatic with a focus on screening and extraction of relevant information. Prior to initiating the classification, I had to manually identify 10 positive and 10 negative examples. These acted as input seed samples for the classifier, which helped pinpoint other useful material. Then the classifier identified other relevant papers by screening the title, abstract and keywords of each study. All papers with a confidence score above 0.5 were selected.

Classification using Gemini. The downside of the previous approach is that it still involves a lot of work to manually identify key studies. To address this, [11] suggests many emerging tools that use Large Language Models (LLMs) to automatically classify relevant material. However, one important factor that limits many tools' usability and effectiveness is the context window being relatively small to handle a large corpus of text. Nonetheless, Gemini [35] is an effective tool for the initial search, as its massive context window – over 2M tokens – allows it to handle a large corpus of text. One inconvenience is that Gemini does not provide citations when using the web interface [37]. However, NotebookLM [38] – which builds on Gemini – solves the issue by providing references to portions of the text that are relevant to the query. This initial screening is an iterative process that involves reading the abstracts and keywords of each study to ascertain about the reliability of each response.

Results. Swift-Review is a more traditional method while NotebookLM is a more recent tool. The former used to be a viable screening tool before the advent of LLMs. However, Gemini being a much larger model, provides a faster and more reliable screening process by giving the user the ability to actively check the results.

S.1.6 - S.1.8 – Manual selection. Step S.1.6 involved a manual inspection over the the title, abstract and keywords. This resulted in a total of 13 studies that appeared to be relevant. After checking their full text, this number was reduced to 4. I performed backward snowballing [43] by revisiting the references of the 4 studies, as well as checking available preprint articles and finally identified other 8 studies (Step 1.8). Hence, a total of 12 studies (4 + 8) were selected. A

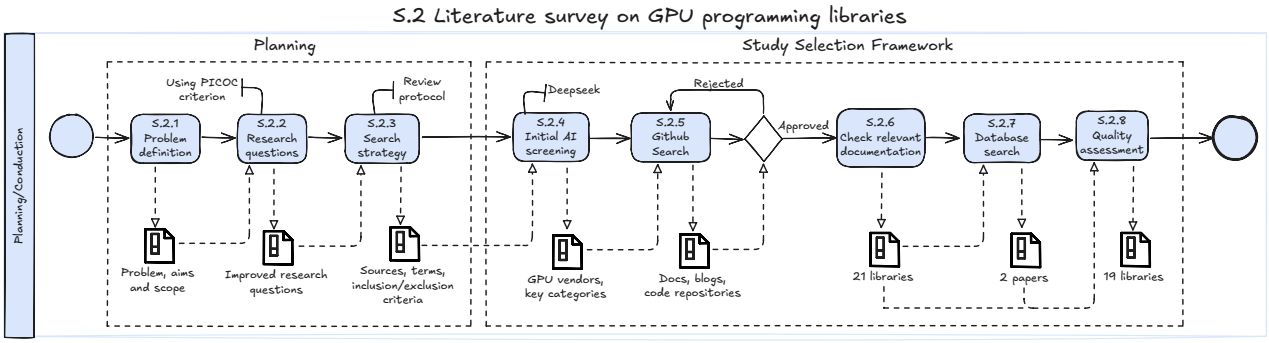


Figure 3. The diagram shows the series of steps carried out in the planning and study selection phases for the GPU programming survey.

.

subsequent 5 frameworks were found with no accompanying papers by reviewing the references. All 17 libraries are listed in Table 2. The results are classified based on the parallelization techniques they support, namely data, model, hybrid and pipeline parallelism.

S.1.9 – Quality assessment. To evaluate the quality of these studies (Step S.1.9), I adapted the quality appraisal instrument suggested by [89], considering 2 main aspects: report and relevance. Regarding report, I checked whether the studies clearly tackled the problem, research questions and inclusion/exclusion criteria defined in Steps S.1.1, S.1.2 and S.1.3 respectively. Concerning relevance, I verified whether the studies presented relevant information to ensure their value for students and practitioners. Finally, all 17 studies identified in the previous steps passed the quality checks.

3.1.2. S.2 – Survey on GPU programming libraries

This step involved identifying popular frameworks that facilitate programming on the GPU. The workflow that was followed is shown in Figure 3.

S.2.1 – Problem definition. It was not possible to perform a systematic review in the traditional sense due to the nature of the available libraries. The available frameworks are frequently proprietary and are rarely accompanied by academic papers. Also, the libraries are implementation-focused with documentation and tutorials being the main source of information. As a result, I had to widen out the type of articles to include in the review by referencing useful tutorials and key documentation pages.

S.2.2 – Research Questions. The relevant research questions from Section 3.1.1 are RQ_2, RQ_3 and RQ_4:

S.2.3 – Search strategy. To find relevant materials (Github repositories, documentation pages and tutorials), Advanced Github Search is used by searching for keywords related to the main GPU manufacturers: AMD, Intel and NVIDIA.

Inclusion/Exclusion criteria. The following criteria were identified to guide the selection process:

- **IC_1**: The material addresses GPU programming.
- **IC_2**: The material is official documentation/repository.
- **IC_3**: The material is a tutorial.
- **IC_4**: The material introduces a library or framework.

- **EC_1**: The material is not a primary source.
- **EC_2**: The material is not written in English.

Search terms. The search terms are similar to the following ones:

user:ROCm user:oneAPI-SRC user:NVIDIA

Publication year criteria. The GPU programming search was restricted to resources published between 2012-2024. The start year (2012) was chosen as the baseline due to being the year when AlexNet [53] was published. This paper revolutionized research in neural networks by allowing advanced AI models to be trained on GPUs.

S.2.4 - S.2.8 – Study Selection Framework. At this step, I used DeepSeek [24] to identify relevant GPU programming keywords and categories (Step S.2.4). This led to identifying [29], which offers an excellent introduction to the topic covering general aspects of GPU programming as well as specific frameworks.

Given the relevant keywords, I followed an iterative approach using search engines, Github repositories and documentation pages to identify relevant material (Step S.2.5). This lead to identifying 21 libraries (S.2.6). Step S.2.7 individuated 2 accompanying papers [18, 65]. The quality assessment step (S.2.8) ensured that the libraries are relevant to training neural networks and the resulting papers

Table 2. A list of papers on Distributed Neural Networks

#	Ref.	Title	Type	Year	Citations	Stars
DNN1	[2]	TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems	Data	2016	9998	187k [1]
DNN2	[17]	MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems	Hybrid	2015	2214	20.8k [60]
DNN3	[41]	GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism	Pipeline	2018	1446	2.8k [39]
DNN4	[45]	BytePS: A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters	Data	2020	338	3.7k [15]
DNN5	[55]	GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding	Model	2020	931	2.8k [39]
DNN6	[56]	PyTorch Distributed: Experiences on Accelerating Data Parallel Training	Hybrid	2020	175	86.1k [69]
DNN7	[57]	Colossal AI: A Unified Deep Learning System for Large-Scale Parallel Training	Hybrid	2023	118	39k [79]
DNN8	[59]	Ray: A distributed Framework for Emerging AI Applications	Hybrid	2018	1108	35k [71]
DNN9	[70]	DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters	Hybrid	2020	1059	36.3k [58]
DNN10	[75]	Horovod: fast and easy distributed deep learning in TensorFlow	Data	2018	1152	14.3k [82]
DNN11	[78]	Megatron-LM: Training Multi-Billion Parameter Models for Natural Language Processing	Hybrid	2020	1578	11.2k [22]
DNN12	[87]	HuggingFace's Transformers: State-of-the-art Natural Language Processing	Data	2020	1444	8.2k [30]
DNN13	[3]	Pytorch Lightning: The lightweight PyTorch wrapper for high-performance AI research. Scale your models, not the boilerplate.	Data	2019	N/A	28.8k [33]
DNN14	[31]	FairScale: A general purpose modular PyTorch library for high performance and large scale training	Hybrid	2021	N/A	3.2k [32]
DNN15	[76]	Amazon SageMaker Platform	Data	2017	N/A	10.3k [77]
DNN16	[74]	Microsoft AzureML Platform	Data	2021	N/A	1.8k [7]
DNN17	[19]	Google Vertex AI Platform	Data	2021	N/A	178 [67]
DNN18	[34]	Jax: Compiling machine learning programs via high-level tracing	Hybrid	2018	N/A	31k [12]

are shown in Table 3. The categories that were included belong to the main GPU manufacturers: AMD, Intel and NVIDIA. Across each category, libraries for algebraic operations (matrix multiplications) include [64, 73, 85], which are used as building blocks for more complex deep learning primitives [6, 18, 66]. Libraries that bridge the gap between GPU programming and DNNs include communication protocols [61, 72, 84]. These frameworks are used in distributed training to synchronize calculations by providing a low-level APIs that manages cross-GPU communication. Moreover, cross-platform and high-level libraries enhance usability and promote wider community adoption. The column concerning the NN libraries [20, 36, 44, 53] represent pioneering frameworks that implement GPU acceleration prior to the advent of optimized frameworks such as cuDNN.

4. M.4 – Reading the studies

Figure 4 shows a timeline of the evolution of the field, suggesting a wide range of available DNN frameworks, intertwined with pioneering GPU programming libraries. Moreover, Figure 5 shows a taxonomy of the libraries in a unified, hierarchical view.

4.1. S.1 – Distributed Neural Networks

In order to more easily extract useful information from the studies, I identified four key criteria which aim to facilitate answering the research questions defined in Section 3.1.1:

- C1: Key Motivating Factors
- C2: Critical Factors and Guidelines
- C3: Practical Evaluation Scenarios
- C4: Tool Limitations and Challenges

The studies' findings are summarized below, with sup-

Table 3. A list of libraries and frameworks for GPU programming

Category	ID	Library/Framework	Vendor	Type	Ref.
NVIDIA	NV1	cuBLAS: GPU-accelerated BLAS library for linear algebra	NVIDIA	Core	[64]
	NV2	cuDNN: Optimized deep neural network primitives	NVIDIA	Core	[18]
	NV4	NCCL: Multi-GPU/multi-node communication	NVIDIA	Core	[61]
	NV5	CUTLASS: Optimized C++ templates for matrix multiplication	NVIDIA	Core	[80]
AMD	AMD1	rocBLAS: AMD's BLAS implementation	AMD	Core	[73]
	AMD2	MIOpen: Deep learning primitives	AMD	Core	[6]
	AMD3	HIP: Portable API for CUDA-like code	Multiple	Framework	[5]
	AMD4	RCCL: Multi-GPU communication	AMD	Core	[72]
Intel	INT1	oneMKL: Math Kernel Library	Intel	Core	[85]
	INT2	oneDNN: Deep learning primitives	Intel	Core	[66]
	INT3	oneCCL: Collective communication library	Intel	Core	[84]
Cross-Platform	CP1	OpenCL: Open standard for heterogeneous computing	Multiple	Framework	[48]
	CP2	SYCL: C++-based multi-device programming	Multiple	Framework	[49]
	CP3	Kokkos: Performance-portable C++ framework	Multiple	Framework	[81]
High-Level	HL1	CUDA.jl: Julia GPU package for NVIDIA	NVIDIA	Language	[47]
	HL2	AMDGPU.jl: Julia GPU package for AMD	AMD	Language	[46]
	HL3	oneAPI.jl: Julia GPU package for Intel	Intel	Language	[10]
	HL4	CuPy: NumPy-compatible GPU arrays for NVIDIA/AMD	Multiple	Language	[23, 65]
	HL5	Numba: JIT compiler for GPU acceleration (NVIDIA only)	NVIDIA	Language	[63]
Tutorials	T1	CUDA Toolkit Samples	NVIDIA	Tutorial	[21]
	T2	AMD Lab Notes	AMD	Tutorial	[4]
	T3	Intel Compute Samples	Intel	Tutorial	[42]
NN Libraries	NN1	Caffe: Convolutional Architecture for Fast Feature Embedding	BVLC	Library	[44]
	NN2	Cuda-Convnet: ImageNet Classification with Deep Convolutional Neural Networks	AlexNet	Library	[52, 53]
	NN3	Pylearn2: a machine learning research library	Pylearn2	Library	[36]
	NN4	Torch7: A Matlab-like Environment for Machine Learning	Torch7	Library	[20]

porting artifacts and passages displayed in Table 7.

Motivating Factors. The motivating factors for training DNNs include the pursuit of better performance through more efficient training and increased usability in practical scenarios.

Scaling. It has been widely thought that by scaling the architectures to a large number of parameters and leveraging larger datasets, the evaluation accuracy on many benchmarks would be improved (D102, D103, D105, D111), idea which is supported by multiple deep learning frameworks. However, it has been recently demonstrated by Deepseek R1 [25] that scaling up the computational power is not the only possible method of progress. As a result, future research is likely to focus not only on distributed training, but also on innovating existing architectures. Nonetheless, through scaling, many applications become viable in industrial settings (D101, D105, D106), which otherwise would be infeasible to train due to the large required training time.

Raw performance. Apart from scalability, execution speed is also a key motivating factor (D103, D105). Some studies have shown that increasing the depth of the network overcomes performance bottlenecks (D103), ensuring applicability in a wide range of domains.

Resource utilization. Yet another important role plays resource utilization which has influence over both scalability and raw performance. This typically involves optimizing practical scenarios where clusters utilize heterogeneous hardware (D104).

Accessability. Beyond performance and scale, ease of

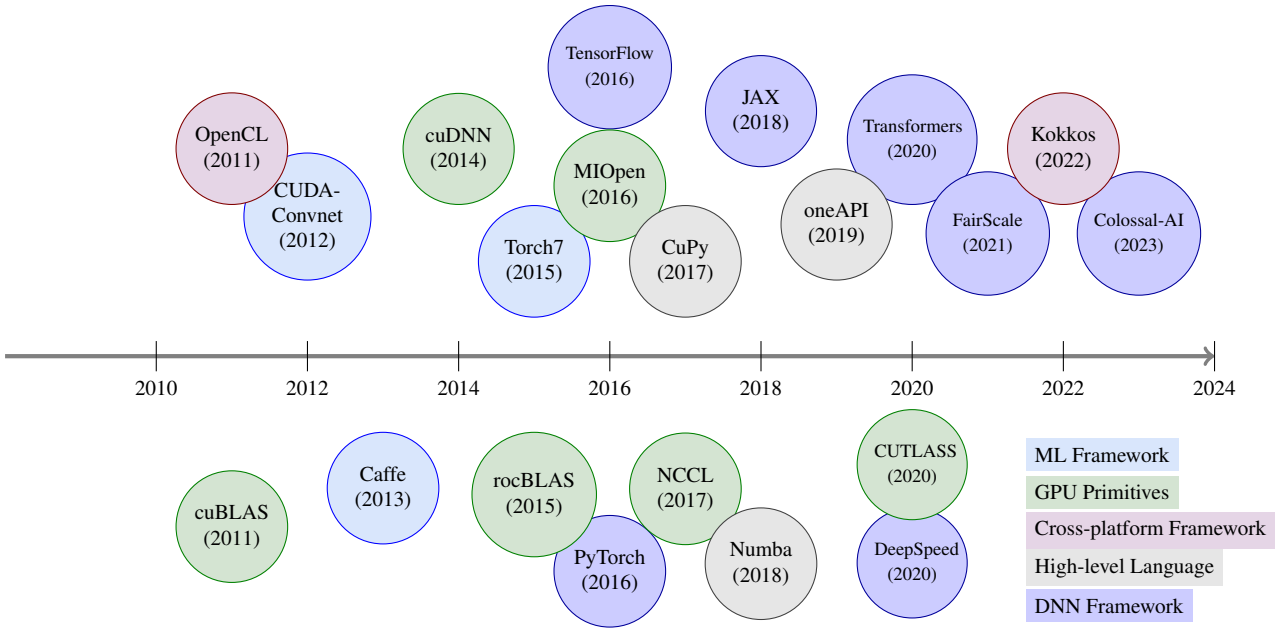


Figure 4. Timeline of Major GPU Programming Libraries and Frameworks

programming and accessability to a wide range of uses also are important. This involves simplifying workflows and enabling access to advanced techniques to a wider audience (D110, D111, D112). However, the need to evolve existing frameworks and support an increasing number of applications as user requirements change are constraints that make usability challenging (D106).

To facilitate broader adoption and flexibility, many frameworks provide cross-platform and cross-framework support (D109, D112). This allows researchers to leverage existing knowledge when transitioning to new frameworks.

Critical Factors. Along the previously mentioned factors – raw speed, scalability and usability – there is also the requirement to support a wide range of architectural choices in distributed environments (D203). This is a critical factor that requires extensive expertise about the underlying hardware, and since there are also other constraints to strive for such as performance and scalability, it becomes challenging to find a balance about what to strive for. In fact, many frameworks make design choices that favor one property over another (D206, D207, D209, D212) and it is necessary to be aware about what each library excels at.

To be more specific, Figure 6a shows popular frameworks that are categorized by the type of parallelism they support, namely data, model, hybrid and pipeline parallelism. Below is a short description about what each category means:

- **Data Parallelism:** The dataset is divided across multiple nodes, with each node training a complete copy of the model on its portion of data. Gradients from all nodes are then combined to update the model parameters. This approach can be implemented either synchronously (all nodes wait for each other) or asynchronously (nodes work independently).
- **Model Parallelism:** The neural network model itself is divided across different nodes, with each node responsible for computing a specific portion of the model architecture. This strategy is particularly useful when the model is too large to fit on a single machine.
- **Pipeline Parallelism:** The training process is divided into sequential stages, where the output of one stage becomes the input for the next. This allows different parts of the model to train simultaneously while maintaining dependencies.
- **Hybrid Parallelism:** This approach combines multiple parallelization strategies to optimize training efficiency. For example, model parallelism might be used to distribute a large model across GPUs, while data parallelism is applied to each model segment.

The choice of the specific technique depends on factors including model architecture, available hardware, and training requirements. For a comprehensive review of these techniques and their implementations, the reader is referred to [16].

Evaluation Scenarios. Evaluation is done by measuring accuracy across a wide range of tasks, including image classification, vision and Natural Language Processing (D303, D305, D306, D308, D311). In order to provide a more fair assessment, many libraries are evaluated in real-world

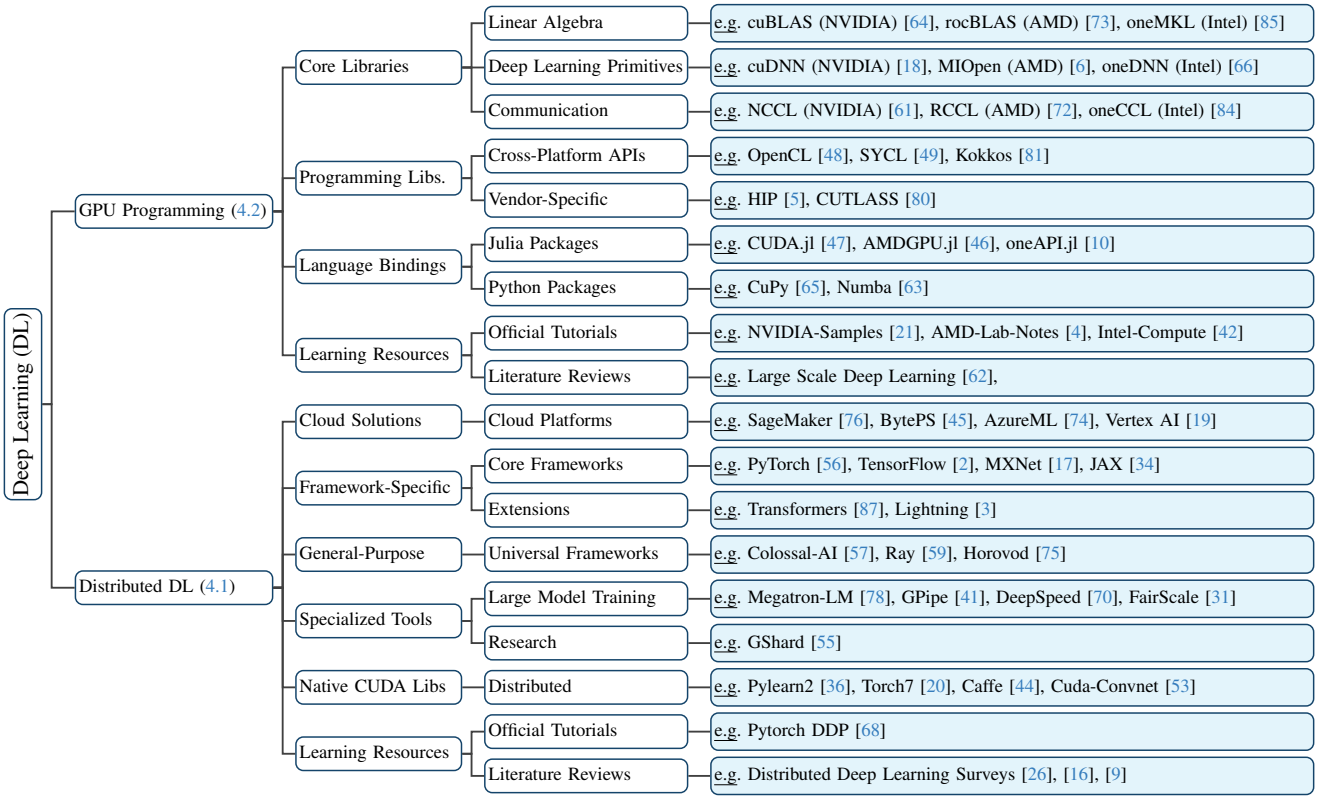


Figure 5. Unified taxonomy of the libraries and frameworks discussed in this paper.

scenarios (D301). Some algorithms have also focused on cross-framework evaluation to ensure broader applicability (D304).

Limitations and Challenges. The main challenges faced by DNNs include communication overhead and resource under-utilization (D401, D403, D404, D405, D407, D410). The lack of standardized tools and frameworks lead to difficulties regarding programming complexity and ease of use (D402, D403, D408).

There exist high-level optimization challenges that prevent achieving peak performance due to different architectures and hardware configurations not being extensively supported (D406, D411). Finally, the requirement to manually tune applications to find out optimal parameter configurations points towards the need to create more user-friendly machine learning frameworks (D411).

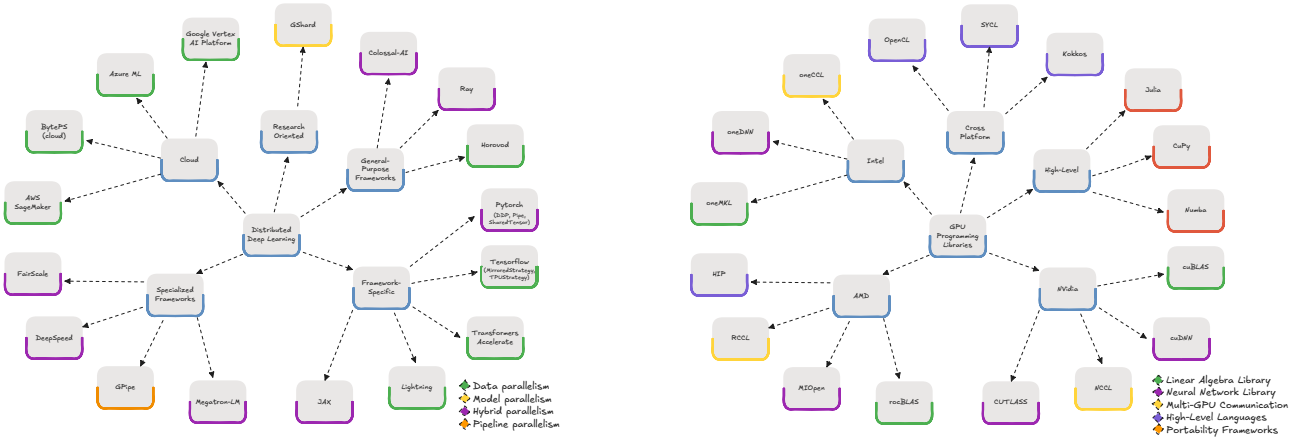
4.2. S.2 – GPU Programming

Figure 6b shows a mindmap of common GPU programming libraries being discussed in this section. Specifically, the various categories are separated by the specific function each provides. Some focus on linear algebra operations (i.e. cuBLAS, rocBLAS, etc) which are primarily used by neural network frameworks (such as cuDNN, CUTLASS, MIOpen, etc.) to build optimized kernels (i.e. convolution operations) w.r.t. the specific GPU architecture. Also, communication libraries provide cross-GPU protocols which are heavily used in distributed training. Other tools such as high-level languages and portability frameworks enhance usability by reducing complexity and allowing for easier integration into existing tools.

Motivation and Critical Factors. Below are summarized the motivating factors for the development of GPU programming libraries, making use of the characteristics displayed in Table 8.

Scalability and performance optimizations. Similarly to DNNs, the development of GPU programming libraries is primarily driven by the need to increase performance (G1013, G1051). This is most often done through focused optimizations involving the learning kernels, which consist in performing large matrix multiplications at a faster speed and reducing the amount of auxiliary memory required. These improvements are intrinsically linked to scalability and the ability to develop larger and more powerful architectures (G1011, G1012).

Compatibility. Another important concern is the integration and compatibility of GPU libraries with existing frameworks (G1013, G1014, G1015, G1062). This is particularly important as seamless compatibility streamlines development and promotes wider adoption. As an example, Caffe [44] particularly emphasizes how time-consuming is



(a) DNN libraries

(b) GPU programming libraries

Figure 6. A mindmap of influential libraries

to develop optimized code for individual architectures. To combat this, frameworks such as CuDNN [18] aim to provide optimized primitives for Nvidia GPUs across multiple GPU architectures.

Usability and programming experience. Another important aspect is usability and its effects on programming productivity (G1031, G1071). Usability emphasizes the creation of tools accessible to a broader audience, leading to increased productivity when building complex applications (G1012, G1016, G1017, G1061). A prime example is CuPy [65], which is a parallel computing framework that provides a NumPy-like interface to GPU-accelerated arrays. It adheres to principles such as separation of concerns and provides support for both declarative and imperative programming styles to create a more user-friendly programming experience (G2012, G2041).

Evaluation. To measure performance, both quantitative and qualitative metrics are used.

Quantitative evaluation. Quantitative metrics assess solutions based on raw performance like convolution speed and raw throughput. This is frequently benchmarked against established baselines (G3011, G3013). As an example, many frameworks assess raw performance by varying mini-batch sizes against popular neural network architectures (G3011).

Additionally, reproducibility across diverse hardware configurations is also an assessment criterion (G3012, G3013, G3061). Without deterministic results, fair performance evaluation is difficult.

Qualitative evaluation. This assessment type is more subjective as it does not involve numerical quantities to base the evaluation on, but rather works by trying to form an intuition about the quality of the results. As an example, AlexNet (G3051) promotes qualitative assessment by visually inspecting the results of the network and basing decisions considering the images' characteristics.

Limitations and Challenges. One significant concern is that the kernel optimizations are time-consuming and require specialized expertise about the dedicated GPU architecture (G4012, G4041). The utilization of standard libraries is is necessary, otherwise the replication of the results would be challenging as execution times can vary significantly across frameworks (G4041).

Algorithmic constraints. Performance bottlenecks are due to matrix operation computations and cross GPU-communication operations. The former is an intrinsic algorithmic limitation (G4061), while the later must be manually managed due to the absence of a standardized API (G4051).

Algorithmic and memory constraints are also present, with some algorithms being limited by high memory usage (G4013), and others requiring specialized implementations for various edge cases (G4013).

Evolving architectures. Lastly, the field also faces ongoing challenges with evolving architectures, both in terms of hardware advancements and new neural network innovations, requiring continuous adaptation as new ideas emerge (G4011).

5. M.5 – Translate concepts

The relationships between these two topics are reviewed below. The *translations* presented in Tables 4, 5, and 6 are deductions derived from the studies' findings. These deductions are based on specific passages detailed in Tables 7 and 8 (Appendix).

Table 4. Translations of the motivating factors

ID	Distributed Neural Networks	GPU Programming	Translation
MF1	• Google internally requires their deep learning frameworks to be scalable. [D101] • Internally, other organizations (i.e. Facebook) become more and more reliant on neural networks. [D106]	• Optimizing kernels is difficult and time-consuming. [G1011]	**Scalability** • There is a surging need for scalability, likely due to the increasingly abundant data availability which is time consuming to process. • The reliance on neural networks has increased productivity and reduced costs.
MF2	• The trend to scale datasets and computational resources yields increased performance in ImageNet competitions. [D102], [D105], [D103] • The abundance of computation and data are particularly effective in Natural Language Processing (NLP) tasks. [D111]	• Natural parallelizability of deep learning techniques enables training higher capacity networks on larger datasets. [G1012] • Early open-source GPU implementations of CNNs set precedent for code sharing. [G1051]	**Complexity and performance** • Effective training parallelization leads to increased performance. • Larger networks consistently provide better performance, especially in NLP tasks. • Open-source implementations have accelerated progress.
MF3	• The deep learning applications are critical in many domains. [D103], [D105] • Frameworks have been extended to support reinforcement learning [D208]	• Deep learning frameworks (Caffe and PADDLE) rely on GPU programming libraries such as cuDNN. [G1014] • As architectures evolve, underlying code needs to be re-optimized. This is standardized by NVidia as they understand better how the GPU architecture works.[G1013]	**Critical in many domains** • GPU programming libraries are used in a more narrow domain, however DNNs have broader applicability in areas such as reinforcement learning. • Nvidia understands well the GPU architecture and can provide better optimizations in critical areas.
MF4	• Data centers are inherently homogenous. BytePS can leverage spare CPU and bandwidth resources to accelerate distributed training running on GPUs. [D104] • Modern systems can leverage mobile devices, tablets, and thousands of GPU cards. [D201]	• GPU programming libraries expose a C language API to communicate with the host CPU. [G1015]	**Heterogenous hardware** • There is limited support for CPU-GPU interaction in GPU programming libraries. • Heterogeneous hardware plays a more important role in DNNs as the ability to fully utilize available resources is critical.
MF5	• DNNs have powered a wide range of applications including image recognition, language translation, anomaly detection, and more. [D106] • Replication of published results can involve months of work by researchers. [G1041]	• GPU libraries meet user's needs by reducing the need to write custom code, allowing developers to focus on higher-level issues, improved portability. [G1016] • Few toolboxes offer truly off-the-shelf deployment of state-of-the-art models that are computationally efficient. [G1041]	**Requirements and applications** • Both topics aim to make it easier for developers to take advantage of parallel hardware. • GPU programming facilitates the development of new architectures and DNNs scale models to achieve better accuracy. • The need for efficient deployment and replication of research results drives development in both areas.
MF6	• Colossal-AI builds on existing open-source frameworks such as PipeDream, GPipe and Chimera. [D107], [D207]	• CuDNN relies on the CUDA toolkit, specifically the cuBLAS library. [G1016] • CuPy is specifically designed to work with NVidia GPUs. [G1062]	**Leverage existing frameworks** • Since DNN frameworks are generally open-source, this encourages community involvement, which leads to innovation. • GPU programming libraries are proprietary. Nonetheless, internally libraries such as cuDNN rely on the CUDA toolkit.
MF7	• Inter-GPU communication frameworks require minimal code changes and support multiple frontends. [D110], [D112] • Libraries can generally be integrated into existing frontend frameworks (i.e. PyTorch). [D211]	• GPU programming libraries provide lower-level primitives and are generally self-contained. [G1017] • Libraries emphasize compatibility (CuPy with NumPy) or ease of development (Torch7). [G1062], [G1071] • Research-focused libraries prioritize configurability and flexibility. [G1031]	**Cross-framework use** • Open-source DNN libraries promote usability and cross-framework compatibility, fostering innovation. • GPU libraries vary between low-level primitives (cuDNN) and user-friendly interfaces (CuPy, Torch7), though being closed-source limits community-driven innovation. • This highlights the importance of code sharing for research.

Table 5. Translations of the critical factors

ID	Distributed Neural Networks	GPU Programming	Translation
CF1	• Generally DNNs are most flexible and use the most common programming style supported by the host language. [D202], [D205]	• Although most GPU frameworks work using the host language as C++, there exist frontend frameworks that enable users to use Python. [G2061] • Cudnn exposes a C API [G1015] • Torch7 is designed for ease of programming through Lua [G2021] • CuPy implements NumPy-like API [G2061]	**Paradigms, programming ease.** • Imperative and declarative programming styles are both supported. • DNNs use Python as the most popular host language, while GPU frameworks generally work with C++ and CUDA. • There do exist frontend frameworks that enable users to write code in higher-level languages (i.e. Lua, Python), and compatibility libraries exist to improve accessability (NumPy-like API).
CF2	• Distributing neural network layers across multiple GPUs is architecture-specific. [D203] • Scaling is expensive in terms of cost, time and code integration. [D209] • There is a separation of concerns between GPU programming and DNNs, as there is no need for custom C++ code or compiler required to distribute neural networks over cluster nodes. [D211]	• Optimized code using NVIDIA GPUs ensures high performance (freeing up auxiliary memory) [G2011] • CuDNN provides separation of concerns by enabling developers to focus on higher-level optimizations instead of low-level architecture specific code [G2012] • Caffe implements separation of representation and implementation [G2041]	**Scalability, separation of concerns.** • Scalability challenges are related to distributing parts of the network or dataset across multiple nodes. • Frameworks emphasize separation of concerns, allowing developers to focus on higher-level optimizations while library providers handle hardware-specific optimizations.
CF3	• Specialized techniques for distributed training include: bucketing, overlapping communication with computation, etc. [D206] • Megatron-LM extends optimization techniques to the transformer model. [D211]	• Libraries optimize for wide range of use cases [G2011] • Frameworks like Torch7 leverage SSE and support multiple parallelization methods [G2021]	**Performance optimization.** • Both domains focus heavily on performance optimization through various techniques. • DNNs use specialized distributed training techniques, while GPU frameworks optimize for different architectures and use cases through various parallelization methods.
CF4	• There exist algorithms that can optimize network latency. [D210] [D204]	• Multi-GPU training is an outstanding challenge [G4011] • GPUs can read/write directly to each other's memory [G2051] • Inter-GPU communication is optimized for specific layers [G2051]	**Network and hardware communication.** • Optimal algorithms exist for network latency optimization, multi-GPU training presents ongoing challenges. • AlexNet optimized inter-GPU communication through direct memory access and selective layer communication. • The CuDNN leaves multi-GPU communication to the user.
CF5	• The Transformers library provides modular components that greatly simplify the extension and ease of use of the library [D212]	• CuPy is NumPy compatible [G1062] • CuDNN requires more specialized C and CUDA knowledge [G1015] • Caffe provides easy CPU/GPU switching and clean Python/MATLAB bindings [G2041]	**Ease of use and hardware flexibility.** • DNN libraries emphasize modularity and ease of extension. • GPU frameworks vary in accessibility - some require specialized knowledge while others provide familiar APIs and easy hardware switching capabilities.

Scalability Relationships The motivation for scalability (MF1) leads to modular implementation approaches (CF2), ultimately resulting in communication limitations (LF3).

[MF1] Scalability. The connection is that scalability is a major shared motivating factor for both DNNs and GPU programming. The increasing scale of data and complexity of DNNs necessitates scalable solutions. GPU programming is motivated by providing the tools and optimizations needed to achieve this scalability, enabling DNNs to handle larger workloads, improve productivity, and become more cost-effective.

[CF2] Scalability. It is achieved by implementing a modular programming style. DNN frameworks abstract away the distributed infrastructure complexity – by being able to easily select distributed strategies when executing the code – while ML frameworks leverage GPU acceleration to hide low-level hardware details, allowing developers to focus on the application logic.

[LF3] Communication Overhead. One limitation involves the communication overhead. Both areas struggle to effectively manage cross-GPU communication, leading to performance bottlenecks that are challenging to overcome. There are not universally optimal solutions, as the best approaches are dependent on model architectures and hardware configurations. Community involvement is essential for innovation as this ensures faster progress.

Hardware Communication Challenges The drive to improve performance by optimizing hardware heterogeneity (MF4) leads to limitations in GPU programming. However, there exist effective algorithms in DNNs, partly due to the open ecosystem (CF4).

[MF4] Heterogeneous hardware. While DNNs are motivated to use heterogenous hardware to ensure broader applicability and performance, GPU programming acknowledge its importance by providing C APIs for CPU-GPU communication. Limitations in the GPU programming field exist due to latency and sub-optimal bandwidth utilization. Specifically, GPU libraries like cuDNN do not provide integrated support for multi-GPU communication, which must be handled manually by the user.

[CF4] Network and hardware communication. Concerning DNNs, there do exist multiple algorithms to optimally minimize network latency, which indicates that the open ecosystem is effective in coming up with solutions.

Performance Improving performance (MF2) involves managing computational complexity (CF3), while evaluation is done through model-specific metrics (EM2).

[MF2] Complexity and performance. The key shared motivation is to manage computational complexity while at the same time reach higher accuracy in common applications (i.e. NLP tasks). The neural network scaling laws predict that accuracy increases linearly with the computational power, which is why there is an ever increasing interest to develop sophisticated algorithms to increase both the model and dataset sizes.

[CF3] Performance. Performance optimizations are achieved through specialized techniques. DNNs are motivated by the need to minimize network bandwidth latency and achieve better scalability, while GPU programming provides optimized primitives by implementing large-matrix operations, harnessing extensive knowledge of the GPU architecture.

[EM2] Model architectures. Regarding the evaluation metrics, for DNNs, this involves assessing scalability with increasingly complex model architectures, while for GPU programming, assessment is geared towards optimizing performance by providing efficient primitives for the most popular operations (convolutions, self-attention, fully connected layers, etc.).

Community and the Available Tools DNNs favour an open ecosystem, while state-of-the-art solutions in GPU programming are proprietary (MF6). The proprietary nature of GPU programming leads to a trade-off between programming ease and performance (CF1).

[MF6] - [MF7]. Leveraging existing tools. DNN libraries promote an open ecosystem, and while there do exist open alternatives in the GPU programming field (i.e. CuPy), the dominant companies promote closed-source solutions (i.e. cuDNN). Despite ongoing effort to create open alternatives (i.e. MIOpen by AMD), currently they provide subpar performance with respect to Nvidia's state-of-the-art GPU programming frameworks.

[CF1] Programming ease. DNNs are generally flexible and often use popular interpreted programming languages like Python to promote ease of use. On the other hand, GPU programming relies on C++ and CUDA, which are critical in areas where speed is important. Many GPU programming libraries provide bindings to popular languages like Python to facilitate ease of use, which shows that the community plays a critical role in both areas.

Usability Tradeoffs The requirement to cover a broad range of applications (MF5) imrpoves ease of use (CF5), however this creates a trade-off between usability and performance (LF1).

[MF5] Applications. A shared motivation is to simplify development and improve the practical utility of both DNNs and GPU programming. Both fields are driven by the need to make life easier for developers to leverage parallel hardware effectively, and the motivation is to simplify deployment and provide reproducible research.

[CF5] Ease of use and hardware flexibility. The main challenge resides around the trade-off between ease of use

Table 6. Translations of evaluation scenarios and limitations

ID	Distributed Neural Networks	GPU Programming	Translation
EM1	• Evaluation was initially performed behind closed doors for internal processes (speech recognition systems) and subsequently for external applications (Google Search). [D301]	• In many frameworks, the GPU libraries can be switched on and off at compile time using a single flag. [G1014] • Some are designed for both research and industry, run on both CPU and GPU, have bindings for both Python and Matlab. For model architecture portability, Protocol Buffer files are used. [G3041]	**Deployment:** • Large companies employ staged deployment: first testing internally before external applications, enabling safe evaluation. • Framework designers can rollback through compile-time flags. CUDA code written in C, with Python and Matlab bindings ensure portability.
EM2	• The evaluation was done by scaling complex networks – based on Mixture of Experts – to 600B parameters using automatic sharding. [D305]	• The cuDNN library is assessed by measuring time and memory usage for convolutional layers. Mini-batch performance is also assessed. Scalability is not as much of a concern as different GPU architectures are benchmarked instead of assessing performance across GPU clusters. [G3011]	**Model Architectures:** • Scalability testing is a concern for DNNs, as these are the type of problems that are encountered in the real world. • However, for GPU programming, performance is measured by optimizing resource usage on a single GPU. The difficulty stands in optimizing performance as new NN architectures are developed.
EM3	• Evaluation tasks include: image classification [D303], machine translation [D303], [D305], NLP [D306] [D311], RL [D308]. • MoE models can be scaled up to 600 billion parameters for machine translation.	• CuDNN can be used in deep learning, CNNs, speech and language. [G3012] • CuPy can be extended to scientific computing and probabilistic modelling. [G3061]	**Task Domains:** • DNN libraries focus on deep learning tasks. • CuDNN was designed for deep learning. • CuPy can be used in broader domains.
EM4	• Impressive improvements in performance over older methods. [D304] • Evaluation is done against vision and NLP tasks [D306] and RL [D308]. • Wikipedia dataset is often used for evaluation. [D307] • Scaling gives consistent improvements in performance. [D311]	• Evaluation metric involves assessing performance and matrix multiplication. Speedup reaches up to 36% improvements. [G3013] • CuDNN is assessed against libraries like cuda-convnet2 and Caffe. Achieves portability across GPU architectures. [G3013] • Qualitative evaluation can offer valuable insights into performance. [G3051]	**Evaluation:** • Potential for gains through hardware-specific optimizations. • Broad applicability of DNNs, while GPU programming focuses on specific architectures. • Consistent scaling suggests promising future for DNNs, while GPU advances focus on specific implementations.
LF1	• To address usability, many libraries provide common APIs with other frameworks. [D402] • Training DNNs require special algorithms that are often architecture-specific. [D403] • Optimizations are challenging and error prone, requirement intimate knowledge of the network architecture. [D411] • Reinforcement learning libraries have bindings that allow both task-parallel and actor-based parallelism. [D408]	• Replication of results is challenging (can take months of work). [G4041] • The requirement to manually fine-tune architectures is time-consuming and requires deep knowledge of the GPU architecture. [G4012] • The memory profiles are used to assess performance. [G4012]	**Usability:** • To address the replication of SOTA results common APIs are used (Transformers [87] framework addresses this issue as it interfaces with [33]). • Being closed source, open-source frameworks cannot reliably match NVidia SOTA performance, as they have better knowledge of the GPU architecture. • Profiles are key to efficient debugging and optimization in both cases.
LF2	• No single algorithm that can perform optimally across all cases. [D406] • Communication overhead leads to resource under-utilization and solutions do not transfer across architectures. [D403], [D405] • Data parallelism models are designed for homogeneous setups. [D404]	• Main issues relate to memory management around matrix multiplication algorithms. Problems also relate to hyperparameter choice, as some stride sizes perform suboptimally. [G4013]	**Algorithmic Limitations:** • No algorithms perform optimally across all cases. • Memory optimizations remain a challenge for both GPU programming and DNNs.
LF3	• Tensorflow performs node placement and communication management which results in overhead. [D401] • Deepspeed incurs communication overhead by allocating data to CPU memory. [D407] • Some papers emphasize collaboration in the research community to ensure innovation. [D410]	• Sophisticated techniques to manage communication overhead by not updating parameters across GPUs on each layer. [G4051] • The cost of transferring data to the GPU outweighs the benefits of using a GPU. [G4061] • The very existence of CuDNN implies that cross-GPU programming is challenging which requires thorough understanding of the GPU architecture. [G4012], [G4011]	**Communication Overhead & Scalability:** • Tradeoffs between communication overhead and performance. • Both GPUs and DNNs face similar bottlenecks in terms of memory allocation that impact performance. • There are no simple universal solutions and choosing the right approach depends on model architectures and hardware. • Community is key to success for DNNs.

and hardware flexibility. Considering the broad community of developers, DNN libraries prioritize modularity and ease of extension to facilitate broader community involvement. Some GPU programming libraries sacrifice ease of use for lower-level control and potentially higher performance (i.e. cuDNN), while others strive for more user-friendly APIs (i.e. CuPy, Caffe).

[LF1] Usability. DNNs prioritize reproducibility, while GPU programming frameworks focus on increasing raw performance. However, due to their proprietary nature, they sacrifice usability by hiding internal implementation details. To help developers overcome this, many frameworks provide profiling tools and specialized debuggers to aid developer productivity.

[EM1] Deployment. DNN libraries follow a staged deployment process. This involves models being frequently deployed as part of a larger product (i.e. Google Search, advertising products, etc.), leading to safer deployment in real world applications. On the other hand, GPU programming frameworks can be easily switched on and off at execution time, making them more configurable and easier to integrate into ML frameworks.

[LF2] Algorithmic limitations. Algorithmic limitations are present in both areas. DNNs pose problems related to memory management in data and model parallelism, while GPU programming faces issues related to matrix multiplication algorithms and difficulties in optimizing algorithms for certain hyperparameter ranges (i.e. small batch size).

6. M.7 – Practical evaluation

The repository contains experiments evaluating `cuDNN`, `cuBLAS` and `Pytorch DDP`. The following represents an introduction to each topic, highlighting the code's main components.

V.1 – GPU programming. The GPU experiments implement a small neural network that represents a proof-of-concept for the capabilities each library provides (i.e. cuDNN and cuBLAS). The following components were implemented:

The main network class contains the following components:

- **forward**: Performs the forward pass.
- **backwardInput**: The backward step w.r.t. the inputs.
- **backwardWeights**: The backward step w.r.t. the weights.
- **zeroGradients**: This manually zeros the gradients, allowing their inspection using a Pytorch-like API.
- **updateWeights**: Takes a step in the opposite direction of the gradient with configurable learning rate.

There were developed two GPU kernels that run code directly on the GPU. The first one is used to compute the forward and backward pass using the ReLU activation function, while the second one is used to compute the MSELoss after the forward pass.

- **MSELoss**: Calculates the mean squared error loss after the forward pass.
- **ReLU**: Activation function that uses manual kernel implementation.

The architecture involves simple convolution and linear layers. They have the following dependencies:

- **Convolution**: Uses cuDNN convolution primitives.
- **Linear**: Uses cuBLAS for the matrix multiplication.

V.2 - V.3 – Distributed Training. The distributed training experiment uses the AlexNet architecture [53] and generates fake data to simulate the training process. The code allows to do the following:

- **Multi-GPU training**: split the dataset across GPUs.
- **Multi-node training**: by producing multiple processes.
- **Collect metrics**: using Weights&Biases (i.e. helps to calculate raw throughput every second).
- **Reproducibility**: is achieved through Docker containers.

The code dynamically spans multiple GPU IDs corresponding to the number of GPUs used in the launch command (i.e. `--nproc_per_node`). By launching multiple processes, the approach allows the model to use more memory of a single GPU, effectively enabling multi-GPU training on a single machine.

In a similar way, the `--nnodes` flag allows to simulate multi-node training across multiple machines. This is achieved by launching multiple containers, where communication is performed through the default loopback interface. Essentially, the master node is the one that synchronizes all nodes and saves performance metrics to `Weights&Biases`.

Limitation. Although the integration with Docker allows to simulate the distributed training environment, it does not provide a true multi-GPU and multi-node performance. The reason is that using a single GPU, it is not possible to run two code instances truly simultaneously. This is due to the context switching mechanism inside the GPU which does not allow for true parallelism. The code could potentially be reutilized in a cloud environment with minimal code changes.

7. M.8 Report findings

Now, to answer the research questions defined in Section 3.

RQ_1. DNN frameworks and communities. The most common frameworks for distributed neural network training are listed in Table 2, which includes the number of citations and stars as an indicator for the community size. The DNN

frameworks are characterized by larger communities, two important examples being Tensorflow and Pytorch, where each pays special attention to usability to promote wider adoption. The dominant programming language is Python, which harnesses the open-source ecosystem to support a wide range of applications. Figure 4 shows the timeline of the release date of the major frameworks.

RQ_2. GPU programming libraries and communities. Popular frameworks in the GPU programming community are listed in Table 3. Due to the fact that most frameworks are closed-source, the communities are generally smaller and more specialized compared to DNNs. The area is dominated by Nvidia libraries such as cuDNN and cuBLAS, as they currently offer state-of-the-art performance in deep learning applications. Nonetheless, AMD and Intel are building tools to provide alternatives with a joint effort for cross-platform compatibility, while AMD particularly focuses on enlarging the ecosystem through open-source contributions.

RQ_3. Overlaps and shared limitations. Scalability is the main shared motivation in both domains, where managing the computational complexity is an important challenge. There are three main ways to manage increased complexity: 1) by increasing the raw computational power, 2) by decreasing the amount of memory used, and 3) by reducing the latency and communication overhead between interconnected parts.

GPU programming primarily focuses on squeezing more performance out of the existing hardware, by providing efficient primitives for matrix multiplication and convolution operations. DNNs, on the other hand, focus on reducing the memory consumption and communication overhead. Both areas face algorithmic challenges. DNNs have the advantage of being able to leverage the large open-source ecosystem which promotes innovation and a larger community involvement. On the other hand, GPU programming requires a deeper understanding of the GPU architectures, and while optimal solutions exist in the open-source community, maintainability is a problem as GPU architectures evolve. This is why closed-source solutions are dominant, as companies can afford to have specialized teams to support the hardware.

8. Conclusion

This review has provided students and practitioners with a basic understanding of the key frameworks in distributed neural network training and GPU programming, offering an introduction to navigate these complex domains.

Contributions. By systematically reviewing the literature and identifying key frameworks, this study addresses the need for a unified view of the field, clarifying the relationships between distributed DNN training and GPU programming, particularly in terms of the common goal to enhance performance and allow the training of increasingly scalable models.

The practical experiments implemented using PyTorch DDP and cuDNN serve to illustrate the concepts discussed, offering a proof-of-concept for the training process using common frameworks.

Limitations. While the methodology used a rigorous systematic review protocol – which is documented to minimise reviewer bias – it is acknowledged that due to the rapidly evolving nature of these fields, some recent developments may not be fully captured. The ongoing effort to advance the field is supported by the substantial impact of the technologies on people's lives.

Future work. The GPU training experiments could be expanded to more realistic architectures. Although, the existing code shows that the network effectively learns, it would be instructive to implement other primitives (i.e. pooling layers, dropout and batch normalization modules) to enhance the scope and usefulness of the possible applications. Finally, the DNN modules could be applied in a cloud environment to more realistically assess performance.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, and Dean. TensorFlow, Large-scale machine learning on heterogeneous systems. https://github.com/tensorflow/tensorflow, 2015. 6

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, and Dean. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. http://arxiv.org/abs/1603.04467, 2016. arXiv:1603.04467 [cs]. 6, 9, 1, 2

[3] Lightning AI. Overview Lightning AI. https://lightning.ai/docs/overview/getting-started, 2010. 6, 9

[4] AMD. amd/amd-lab-notes. https://github.com/amd/amd-lab-notes, 2025. original-date: 2022-10-20T02:26:59Z. 7, 9

[5] AMD. ROCm/HIP. https://github.com/ROCm/HIP, 2025. original-date: 2016-01-07T17:41:56Z. 7, 9

[6] AMD. ROCm/MIOpen. https://github.com/ROCm/MIOpen, 2025. original-date: 2017-06-27T17:51:22Z. 6, 7, 9

[7] Microsoft Azure. Azure/azureml-examples. https://github.com/Azure/azureml-examples, 2025. original-date: 2020-08-21T18:04:26Z. 6

[8] Tal Ben-Nun and Torsten Hoefler. Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis. https://dl.acm.org/doi/10.1145/3320060, 2020. 1, 4

[9] Francesco Berloco, Vitoantonio Bevilacqua, and Simona Colucci. A Systematic Review of Distributed Deep Learning Frameworks for Big Data, 2022. 2, 9

[10] Tim Besard. oneAPI.jl. https://github.com/JuliaGPU/oneAPI.jl, 2022. 7, 9

[11] Francisco Bolaños, Angelo Salatino, Francesco Osborne, and Enrico Motta. Artificial intelligence for literature reviews: opportunities and challenges. https://doi.org/10.1007/s10462-024-10902-3, 2024. 4

[12] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. jax-ml/jax. https://github.com/jax-ml/jax, 2025. original-date: 2018-10-25T21:25:02Z. 6

[13] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. https://linkinghub.elsevier.com/retrieve/pii/S016412120600197X, 2007. 2

[14] David Budgen, Pearl Brereton, Sarah Drummond, and Nikki Williams. Reporting systematic reviews: Some lessons from a tertiary study. https://www.sciencedirect.com/science/article/pii/S0950584916303548, 2018. 2

[15] ByteDance. bytedance/byteps. https://github.com/bytedance/byteps, 2025. original-date: 2019-06-25T07:00:13Z. 6

[16] Karanbir Chahal, Manraj Singh Grover, and Kuntal Dey. A Hitchhiker's Guide On Distributed Training of Deep Neural Networks. http://arxiv.org/abs/1810.11787, 2018. arXiv:1810.11787 [cs]. 1, 2, 8, 9

[17] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. https://www.semanticscholar.org/paper/MXNet%3A-A-Flexible-and-Efficient-Machine-Learning-Chen-Li/62df84d6a4d26f95e4714796c2337c9848cc13b5, 2015. 6, 9, 1, 2

[18] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. http://arxiv.org/abs/1410.0759, 2014. arXiv:1410.0759 [cs]. 5, 6, 7, 9, 10, 3, 4

[19] Google Cloud. Vertex AI documentation. https://cloud.google.com/vertex-ai/docs, 2021. 6, 9

[20] Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet. Torch7: A Matlab-like Environment for Machine Learning. https://ronan.collobert.com/pub/matos/2011_torch7_nipsw.pdf, 2011. 6, 7, 9, 3

[21] NVIDIA Corporation. NVIDIA/cuda-samples. https://github.com/NVIDIA/cuda-samples, 2025. original-date: 2018-03-27T17:36:24Z. 2, 7, 9

[22] NVIDIA Corporation. NVIDIA/Megatron-LM. https://github.com/NVIDIA/Megatron-LM, 2025. original-date: 2019-03-21T16:15:52Z. 6

[23] cupy. cupy/cupy. https://github.com/cupy/cupy, 2025. original-date: 2016-11-01T09:54:45Z. 7

[24] DeepSeek. DeepSeek. https://chat.deepseek.com, 2025. 5

[25] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and Bing Xue. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. https://arxiv.org/abs/2501.12948, 2025. 7

[26] Mohammad Dehghani and Zahra Yazdanparast. From distributed machine to distributed deep learning: a comprehensive survey. https://doi.org/10.1186/s40537-023-00829-x, 2023. 1, 2, 9

[27] Vinicius dos Santos, Anderson Y. Iwazaki, Katia R. Felizardo, Érica F. de Souza, and Elisa Y. Nakagawa. Sustainable systematic literature reviews. https://www.sciencedirect.com/science/article/pii/S0950584924001563, 2024. 2

[28] Vinicius Dos Santos, Anderson Y. Iwazaki, Katia R. Felizardo, Érica F. De Souza, and Elisa Y. Nakagawa. Sustainable systematic literature reviews. https://linkinghub.elsevier.com/retrieve/pii/S0950584924001563, 2024. 2

[29] ENCCS. ENCCS/gpu-programming: Meta-GPU lesson covering general aspects of GPU programming as well as specific frameworks. https://github.com/ENCCS/gpu-programming/tree/main, 2025. 5

[30] Hugging Face. huggingface/accelerate. https://github.com/huggingface/accelerate, 2025. original-date: 2020-10-30T13:27:12Z. 6

[31] FairScale. FairScale Documentation | FairScale documentation. https://fairscale.readthedocs.ioindex.html, 2021. 6, 9

[32] FairScale authors. Fairscale: A general purpose modular pytorch library for high performance and large scale training. https://github.com/facebookresearch/fairscale, 2021. 6

[33] William Falcon and The PyTorch Lightning team. PyTorch Lightning. https://github.com/Lightning-AI/lightning, 2019. 6, 14

[34] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing, 2018. 6, 9

[35] Gemini Authors. Gemini: A Family of Highly Capable Multimodal Models. http://arxiv.org/abs/2312.11805, 2024. arXiv:2312.11805 [cs]. 4

[36] Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. http://arxiv.org/abs/1308.4214, 2013. arXiv:1308.4214 [stat]. 6, 7, 9, 3

[37] Google. Chat to supercharge your ideas. https://gemini.google.com, 2023. 4

[38] Google. Google NotebookLM | Note Taking & Research Assistant Powered by AI. https://notebooklm.google/, 2023. 4

[39] GPipe. tensorflow/lingvo. https://github.com/tensorflow/lingvo, 2025. original-date: 2018-07-24T22:30:28Z. 6

[40] Brian E. Howard, Jason R. Phillips, Kyle Miller, Arpit Tandon, Deepak Mav, Mihir R. Shah, Stephanie D Holmgren, Katherine E Pelch, Vickie R. Walker, Andrew A. Rooney, Malcolm Robert Macleod, Ruchir R. Shah, and Kristina Thayer. Swift-review: a text-mining workbench for systematic review. https://api.semanticscholar.org/CorpusID:5970071, 2016. 4

[41] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. http://arxiv.org/abs/1811.06965, 2019. arXiv:1811.06965 [cs]. 6, 9, 1, 2

[42] Intel. intel/compute-samples. https://github.com/intel/compute-samples, 2025. original-date: 2017-10-20T22:04:08Z. 7, 9

[43] Samireh Jalali and Claes Wohlin. Systematic literature studies: database searches vs. backward snowballing. https://doi.org/10.1145/2372251.2372257, 2012. 4

[44] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. http://arxiv.org/abs/1408.5093, 2014. arXiv:1408.5093 [cs]. 6, 7, 9, 3, 4

[45] Yimin Jiang, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters, 2020. 6, 9, 1, 2

[46] JuliaGPU. JuliaGPU/AMDGPU.jl. https://github.com/JuliaGPU/AMDGPU.jl, 2025. original-date: 2020-07-02T16:16:24Z. 7, 9

[47] JuliaGPU. JuliaGPU/CUDA.jl. https://github.com/JuliaGPU/CUDA.jl, 2025. original-date: 2019-09-06T13:35:08Z. 7, 9

[48] KhronosGroup. KhronosGroup/OpenCL-SDK. https://github.com/KhronosGroup/OpenCL-SDK, 2025. original-date: 2020-02-10T17:18:01Z. 7, 9

[49] KhronosGroup. KhronosGroup/SYCL-Docs. https://github.com/KhronosGroup/SYCL-Docs, 2025. original-date: 2019-05-13T16:07:19Z. 7, 9

[50] Barbara Kitchenham. Procedures for Performing Systematic Regviews, 2004. 2

[51] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. Evidence-Based Software Engineering and Systematic Reviews. CRC Press, 2015. Google-Books-ID: bGfdCgAAQBAJ. 3

[52] Alex Krizhevsky. Google Code Archive - Long-term storage for Google Code Project Hosting. https://code.google.com/archive/p/cuda-convnet/, 2011. 7

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html, 2012. 1, 5, 6, 7, 9, 15, 3, 4

[54] Matthias Langer, Zhen He, Wenny Rahayu, and Yanbo Xue. Distributed Training of Deep Learning Models: A Taxonomic Perspective. https://ieeexplore.ieee.org/document/9120226/, 2020. 1

[55] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. http://arxiv.org/abs/2006.16668, 2020. arXiv:2006.16668 [cs]. 6, 9, 1, 2

[56] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. https://arxiv.org/abs/2006.15704, 2020. 6, 9, 1, 2

[57] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training. http://arxiv.org/abs/2110.14883, 2023. arXiv:2110.14883 [cs]. 6, 9, 1, 2

[58] Microsoft. microsoft/DeepSpeed. https://github.com/microsoft/DeepSpeed, 2025. original-date: 2020-01-23T18:35:18Z. 6

[59] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A Distributed Framework for Emerging AI Applications. http://arxiv.org/abs/1712.05889, 2018. arXiv:1712.05889 [cs]. 6, 9, 1, 2

[60] MXNet. apache/mxnet. https://github.com/apache/mxnet, 2025. original-date: 2015-04-30T16:21:15Z. 6

[61] nccl. NVIDIA/nccl. https://github.com/NVIDIA/nccl, 2025. original-date: 2015-11-14T00:12:04Z. 6, 7, 9

[62] Giang Nguyen, Stefan Dlugolinsky, Martin Bobák, Viet Tran, Álvaro López García, Ignacio Heredia, Peter Malík, and Ladislav Hluchý. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. https://doi.org/10.1007/s10462-018-09679-z, 2019. 2, 9

[63] numba. numba/numba. https://github.com/numba/numba, 2025. original-date: 2012-03-08T11:12:43Z. 7, 9

[64] NVIDIA. cuBLAS. https://developer.nvidia.com/cublas, 2007. 6, 7, 9

[65] NVIDIA. CuPy : A NumPy-Compatible Library for NVIDIA GPU Calculations. https://www.semanticscholar.org/paper/CuPy-%3A-A-NumPy-Compatible-Library-for-NVIDIA-GPU-Okuta-Unno/a59da4639436f582e483347a4833e7659fd3e598, 2017. 5, 7, 9, 10, 3, 4

[66] oneDNN Contributors. oneAPI Deep Neural Network Library (oneDNN). https://github.com/oneapi-src/oneDNN, 2025. original-date: 2016-05-09T23:26:42Z. 6, 7, 9

[67] Google Cloud Platform. GoogleCloudPlatform/vertex-ai-samples. https://github.com/GoogleCloudPlatform/vertex-ai-samples, 2025. original-date: 2021-05-27T00:06:43Z. 6

[68] PyTorch. examples/distributed at main · pytorch/examples. https://github.com/pytorch/examples/blob/main/distributed, 2025. 2, 9

[69] PyTorch. pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. https://github.com/pytorch/pytorch, 2025. 6

[70] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. https://dl.acm.org/doi/10.1145/3394486.3406703, 2020. Conference Name: KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining ISBN: 9781450379984 Place: Virtual Event CA USA Publisher: ACM. 6, 9, 1

[71] ray project. ray-project/ray. https://github.com/ray-project/ray, 2025. original-date: 2016-10-25T19:38:30Z. 6

[72] rccl. ROCm/rccl. https://github.com/ROCm/rccl, 2025. original-date: 2016-01-07T17:41:56Z. 6, 7, 9

[73] rocblas. ROCm/rocBLAS. https://github.com/ROCm/rocBLAS, 2025. original-date: 2017-06-27T17:51:22Z. 6, 7, 9

[74] sdgilley. Azure Machine Learning documentation. https://learn.microsoft.com/en-us/azure/machine-learning/?view=azureml-api-2, 2021. 6, 9

[75] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. http://arxiv.org/abs/1802.05799, 2018. arXiv:1802.05799 [cs]. 6, 9, 1, 2

[76] Amazon Web Services. Amazon SageMaker Documentation. https://docs.aws.amazon.com/sagemaker/, 2021. 6, 9

[77] Amazon Web Services. aws/amazon-sagemaker-examples. https://github.com/aws/amazon-sagemaker-examples, 2025. original-date: 2017-10-23T05:55:22Z. 6

[78] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. http://arxiv.org/abs/1909.08053, 2020. arXiv:1909.08053 [cs]. 6, 9, 1, 2

[79] HPC-AI Tech. hpcaitech/ColossalAI. https://github.com/hpcaitech/ColossalAI, 2025. original-date: 2021-10-28T16:19:44Z. 6

[80] Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS. https://github.com/NVIDIA/cutlass, 2023. original-date: 2017-11-30T00:11:24Z. 7, 9

[81] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. Kokkos 3: Programming Model Extensions for the Exascale Era. https://github.com/kokkos/kokkos, 2022. Issue: 4 Pages: 805–817 Publication Title: IEEE Transactions on Parallel and Distributed Systems Volume: 33 original-date: 2015-04-08T21:55:55Z. 7, 9

[82] Uber. horovod/horovod. https://github.com/horovod/horovod, 2025. original-date: 2017-08-09T19:39:59Z. 6

[83] Keele University. Guidelines for performing systematic literature reviews in software engineering. https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf, 2007. [Accessed 13-01-2025]. 2, 3

[84] UXL. uxlfoundation/oneCCL. https://github.com/uxlfoundation/oneCCL, 2025. original-date: 2019-09-09T21:57:46Z. 6, 7, 9

[85] UXL. uxlfoundation/oneMath. https://github.com/uxlfoundation/oneMath, 2025. original-date: 2020-03-05T23:55:03Z. 6, 7, 9

[86] Papers with Code. Papers with Code - The Methods Corpus. https://paperswithcode.com/methods, 2025. 1

[87] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. HuggingFace's Transformers: State-of-the-art Natural Language Processing. http://arxiv.org/abs/1910.03771, 2020. arXiv:1910.03771 [cs]. 6, 9, 14, 1

[88] Eric P. Xing, Qirong Ho, Pengtao Xie, and Wei Dai. Strategies and Principles of Distributed Machine Learning on Big Data. http://arxiv.org/abs/1512.09295, 2015. arXiv:1512.09295 [stat]. 1

[89] Xin Zhou, Yuqin Jin, He Zhang, Shanshan Li, and Xin Huang. A Map of Threats to Validity of Systematic Literature Reviews in Software Engineering. https://ieeexplore.ieee.org/abstract/document/7890583, 2016. ISSN: 1530-1362. 5

Survey on Distributed Neural Networks and GPU Programming Frameworks

Supplementary Material

Table 7. The passages on distributed neural networks

Cat.	ID	Text Passages	Ref.	Codes
RQ1: Key Motivating Factors	D101 [2]	In addition, often in close collaboration with the Google Brain team, more than 50 teams at Google and other Alphabet companies have deployed deep neural networks using DistBelief in a wide variety of products, including Google Search [11], our advertising products, our speech recognition systems [50, 6, 46], Google Photos [43], Google Maps and StreetView [19], Google Translate [18], YouTube, and many others.	[2, 56]	• Internal need to scale existing products
	D102 [17]	The scale and complexity of machine learning (ML) algorithms are becoming increasingly large. Almost all recent ImageNet challenge [12] winners employ neural networks with very deep layers, requiring billions of floating-point operations to process one single sample. The rise of structural and computational complexity poses interesting challenges to ML system design and implementation.	[17, 55, 78]	• Increasingly complex models/datasets • Keen interest for scientific inquiry
	D103 [41]	We scale the architecture along two dimensions to stress the flexibility of GPipe: (i) along the depth by increasing the number of layers in the model and (ii) along the width by increasing the hidden dimension in the feed-forward layers and the number of attention heads (...) We notice that increasing the model capacity, from 400M params (T (6, 8192, 16)) to 1.3B (T (24, 8192, 16)), and further, to 6B (T (64, 16384, 32)), leads to significant quality improvements across all languages.	[41, 55]	• Improve performance • Critical in many domains
	D104 [45]	Data center clusters that run DNN training jobs are inherently heterogeneous. They have GPUs and CPUs for computation and network bandwidth for distributed training. However, existing distributed DNN training architectures, all-reduce and Parameter Server (PS), cannot fully utilize such heterogeneous resources. In this paper, we present a new distributed DNN training architecture called BytePS. BytePS can leverage spare CPU and bandwidth resources in the cluster to accelerate distributed DNN training tasks running on GPUs.	[45]	• Utilization of heterogenous hardware
	D105 [55]	Neural network scaling has been critical for improving the model quality in many real-world machine learning applications with vast amounts of training data and compute. Although this trend of scaling is affirmed to be a sure-fire approach for better model quality, there are challenges on the path such as the computation cost, ease of programming, and efficient implementation on parallel devices.	[17, 41, 55, 78]	• Increasingly complex models/datasets • Improve performance • Critical in many domains
	D106 [56]	Deep Neural Networks (DNN) have powered a wide spectrum of applications, ranging from image recognition [20], language translation [15], anomaly detection [16], content recommendation [38], to drug discovery [33], art generation [28], game play [18], and self-driving cars [13]. Many applications pursue higher intelligence by optimizing larger models using larger datasets, craving advances in distributed training systems. Among existing solutions, distributed data parallel is a dominant strategy due to its minimally intrusive nature. (...) During the past year, we have seen significant adoption both internally and externally. Within Facebook, a workload study from 05/11/20 to 06/05/20 shows that more than 60% of production GPU hours during that period were spent on the PyTorch distributed data parallel package across a wide variety of applications, including speech, vision, mobile vision, translation, etc.	[2, 56]	• Meeting user's requirements • Emerging applications • Internal need to scale existing products
	D107 [57]	Methods such as PipeDream [25], GPipe [16], and Chimera [20] were proposed to split the model into several chunks of consecutive layers and each chunk is allocated to a device as shown in Figure 3c. Intermediate activations and gradients are passed between pipeline stages to complete the forward and backward pass. As a result, our method reduces cross-node communication. Pipeline parallelism allows multiple devices to compute simultaneously, leading to a higher throughput. (...) Inspired by Alpa, Colossal-AI has included an experimental automatic parallelism feature to improve upon the Alpa project.	[57]	• Improving/Building on existing frameworks
	D108 [59]	In our evaluation, we study the following questions: (...) 2. What overheads are imposed on distributed primitives (e.g., allreduce) written using Ray's API? (Section 5.1) 3. In the context of RL workloads, how does Ray compare against specialized systems for training, serving, and simulation? (Section 5.2) 4. What advantages does Ray provide for RL applications, compared to custom systems? (Section 5.3)	[59]	• Extending existing tools to new domains i.e. Reinforcement Learning
	D109 [70]	DeepSpeed is compatible with PyTorch. One piece of our library, called ZeRO, is a new parallelized optimizer that greatly reduces the resources needed for model and data parallelism while massively increasing the number of parameters that can be trained. Researchers have used these breakthroughs to create Turing Natural Language Generation (Turing-NLG), which at the time of its release was the largest publicly known language model at 17 billion parameters.	[70, 75]	• Cross-framework compatibility • Large-scale training
	D110 [75]	Existing methods for enabling multi-GPU training under the TensorFlow library entail non-negligible communication overhead and require users to heavily modify their model-building code, leading many researchers to avoid the whole mess and stick with slower single-GPU training. In this paper we introduce Horovod, an open source library that improves on both obstructions to scaling: it employs efficient inter-GPU communication via ring reduction and requires only a few lines of modification to user code, enabling faster, easier distributed training in TensorFlow.	[70, 75, 87]	• Cross-framework compatibility • Large-scale training • Ease of use
	D111 [78]	Natural Language Processing (NLP) is advancing quickly in part due to an increase in available compute and dataset size. The abundance of compute and data enables training increasingly larger language models via unsupervised pretraining... Empirical evidence indicates that larger language models are dramatically more useful for NLP tasks such as article completion, ques-tion answering, and natural language inference (...) In summary, our approach as de-scribed above is simple to implement, requiring only a few extra all-reduce operations added to the forward and back-ward pass. It does not require a compiler, and is orthogonal and complementary to the pipeline model parallelism advo-cated by approaches such as (Huang et al., 2018).	[17, 55, 78]	• Increasingly complex models/datasets • No need for compilers
	D112 [87]	An increasingly important goal of Transformers is to make it easy to efficiently deploy model to pro-duction. Different users have different production needs, and deployment often requires solving significantly different challenges than training. The library thereforce allows for several different strategies for production deployment. One core propery of the libary is that models are available both in PyTorch and TensorFlow, and there is interoperability between both frameworks.	[70, 75, 87]	• Ease of use • Cross-framework compatibility
RQ3: Critical Factors	D201 [2]	A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices such as GPU cards.	[2, 45]	• Utilization of heterogenous hardware
	D202 [17]	Most ML systems embed a domain-specific language (DSL) into a host language (e.g. Python, Lua, C++). Possible programming paradigms range from imperative, where the user specifies exactly "how" computation needs to be performed, and declarative, where the user specification focuses on "what" to be done.	[17, 55]	• Programming paradigms
	D203 [41]	In many cases, increasing model capacity beyond the memory limit of a single accelerator has required developing special algorithms or infrastructure. These solutions are often architecture-specific and do not transfer to other tasks. To address the need for efficient and task-independent model parallelism, we introduce GPipe, a pipeline parallelism library that allows scaling any network that can be expressed as a sequence of layers.	[41, 70, 78]	• Scaling
	D204 [45]	BytePS is a unified distributed DNN training acceleration system that achieves optimal communication efficiency in heterogeneous GPU/CPU clusters.	[45, 57, 75]	• Communication efficiency
	D205 [55]	In automatic sharding model description should be separated from the partitioning implementation and optimization. This separation of concerns let model developers focus on the network architecture and flexibly change the partitioning strategy, while the underlying system applies semantic-preserving transformations and implements efficient parallel execution.	[17, 55]	• Separation of concerns • Programming ease
	D206 [56]	PyTorch natively provides several techniques to accelerate distributed data parallel, including bucketing gradients, overlapping compu-tation with communication, and skipping gradient synchronization.	[56, 57, 70, 78]	• Performance
	D207 [57]	Methods such as PipeDream [25], GPipe [16], and Chimera [20] were proposed to split the model into several chunks of consecutive layers and each chunk is allocated to a device as shown in Figure 3c. Intermediate activations and gradients are passed between pipeline stages to complete the forward and backward pass. As a result, this method reduces cross-node communication. Pipeline parallelism allows multiple devices to compute simultaneously, leading to a higher throughput.	[56, 57, 70, 78]	• Performance
	D208 [59]	To learn a policy, an agent typically employs a two-step process: (1) policy evaluation and (2) policy improvement. To evaluate the policy, the agent interacts with the environment (e.g., with a simulation of the environment) to generate trajectories, where a trajectory consists of a sequence of (state, reward) tuples produced by the current policy.	[59]	• Policy learning in reinforcement learning
	D209 [70]	The latest trend in AI is that larger natural language models provide better accuracy; however, larger models are difficult to train because of cost, time, and ease of code integration. With the goal of advancing large model training by improving scale, speed, cost, and usability for model developers across the world, Since then, the DeepSpeed team has been hard at work extending the library to continue pushing the boundaries of scale and speed of deep learning training.	[41, 70, 78]	• Performance • Cost • Scalability • Usability
	D210 [75]	In early 2017 Baidu published an article [8] evangelizing a different algorithm for averaging gradients and communicating those gradients to all nodes (Steps 2 and 3 above), called ring-allreduce (...) allows worker nodes to average gradients and disperse them to all nodes without the need for a parameter server (...) This algorithm is bandwidth-optimal, meaning that if the buffer is large enough, it will optimally utilize the available network.	[57, 75]	• Network Latency
	D211 [78]	Our approach is to utilize model parallelism to split the model across multiple accelerators. This not only alleviates the memory pressure, but also increases the amount of parallelism independently of the microbatch size. (...) By increasing the minibatch size proportionally to the number of available workers (i.e. weak scaling), one observes near linear scaling in training data throughput. (...) We exploit the inherent structure in transformer based language models to make a simple model-parallel implementation that trains efficiently in PyTorch, with no custom C++ code or compiler required.	[70, 78]	• Performance • Cross-framework compatibility • Scalability • Usability
	D212 [87]	Each model is made up of a Tokenizer, Transformer, and Head. The model is pretrained with a fixed head and can then be further fine-tuned with alternate heads for different tasks.	[70, 78, 87]	• Ease of use
	D301 [2]	In addition, often in close collaboration with the Google Brain team, more than 50 teams at Google and other Alphabet companies have deployed deep neural networks using DistBelief in a wide variety of products, including Google Search [11], our advertising products, our speech recognition systems [50, 6, 46], Google Photos [43], Google Maps and StreetView [19], Google Translate [18], YouTube, and many others.	[2]	• Deployment via Google Apps
	D303 [41]	We demonstrate the advantages of GPipe by training large-scale neural networks on two different tasks with distinct network architectures: (i) Image Classification: We train a 557-million-parameter AmoebaNet model and attain a top-1 accuracy of 84.4% on ImageNet-2012, (ii) Multilingual Neural Machine Translation: We train a single 6-billion-parameter, 128-layer Transformer model on a corpus spanning over 100 languages and achieve better quality than all bilingual models.	[41, 55]	• Image Classification • Machine Translation

Table 7 – continued from previous page

Cat.	ID	Text Passages	Ref.	Codes
RQ_2: Evaluation Metrics	D304 [45]	We evaluate BytePS using six DNN models and three training frameworks (TensorFlow, PyTorch, MXNet) in production data centers. The results show that with 256 GPUs, BytePS consistently outperform existing all-reduce and PS solutions by up to 84% and 245%, respectively.	[45]	• Cross-platform evaluation (TensorFlow, PyTorch, MXNet)
	D305 [55]	GShard enabled us to scale up multilingual neural machine translation Transformer model with Sparsely-Gated Mixture-of-Experts beyond 600 billion parameters using automatic sharding. We demonstrate that such a giant model can efficiencly be trained on 2048 TPU v3 accelerators in 4 days to achieve far superior quality for translation from 100 languages to English compared to the prior art.	[41, 55]	• Models: MoEs • Machine Translation
	D306 [56]	We measure DDP per iteration latency and scalability using two popular models, ResNet50 [20] and BERT [15], to represent typical vision and NLP applications.	[56, 57]	• Tasks: NLP, Vision • Performance
	D307 [57]	To demonstrate the capability of dynamic tensor placement in ColossalAI, we trained GPT-2 model with 10 billion parameters on the Wikipedia dataset on System II. We set the batch size to 4 and scaled the data parallel training from 1 GPU to 8 GPU.	[56, 57, 59, 78]	• Performance
	D308 [59]	In our experiments, we demonstrate scaling beyond 1.8 million tasks per second and better performance than existing specialized systems for several challenging reinforcement learning applications.	[56, 57, 59, 78]	• Performance • Tasks: RL
	D311 [78]	Using the GPT-2 model we achieve SOTA results on the WikiText103 (10.8 compared to SOTA perplexity of 15.8) and LAMBADA (66.5% compared to SOTA accuracy of 63.2%) datasets. Our BERT model achieves SOTA results on the RACE dataset (90.9% compared to SOTA accuracy of 89.4%) (...) We demonstrate that scaling the model size results in improved accuracies for both GPT-2 (studied up to 8.3 billion parameters) and BERT (studied up to 3.9B parameters) models.	[56, 57, 59, 78]	• Tasks: NLP • Performance • Scaling
RQ_2: Tool limitations and challenges	D401 [2]	Once a system has multiple devices, there are two main complications: deciding which device to place the computation for each node in the graph, and then managing the required communication of data across device boundaries implied by these placement decisions. (...) A future version of this white paper will have a comprehensive performance evaluation section of both the single machine and distributed implementations.	[2, 57, 75]	• Communication overhead
	D402 [17]	Most ML systems embed a domain-specific language (DSL) into a host language (e.g. Python, Lua, C++). (...) Comparing to other open-source ML systems, MXNet provides a superset programming interface to Torch7, Theano, Chainer and Caffe, and supports more systems such as GPU clusters.	[17, 41]	• Ease of use • Common API with other frameworks
	D403 [41]	(Other) naive model parallelism strategies lead to severe under-utilization due to the sequential dependency of the network. (...) In many cases, increasing model capacity beyond the memory limit of a single accelerator has required developing special algorithms or infrastructure. These solutions are often architecture-specific and do not transfer to other tasks.	[17, 41, 45, 55]	• Resource under-utilization • No shared API with common frameworks
	D404 [45]	For distributed training, there are two families of data parallelism approaches, i.e., all-reduce and Parameter Server (PS). In all-reduce, no additional CPU machine is involved [to aggregate results from different accelerators] . Ring is the most popular all-reduce algorithm. (...) All-reduce has no way to utilize additional non-worker nodes, since it was designed for homogeneous setup.	[41, 45, 55]	• Designed for homogeneous setup • Resource under-utilization
	D405 [55]	There is a lack of support for efficient model parallelism algorithms under commonly used deep learning frameworks such as TensorFlow [21] and PyTorch [22]. Naive model parallelism with graph partition is supported but it would lead to severe under-utilization due to the sequential dependency of the network and gradient based optimization.	[41, 45, 55]	• Resource under-utilization • No efficient model parallelism algorithms
	D406 [56]	Despite the conceptual simplicity of the technique, the subtle dependencies between computation and communication make it non-trivial to optimize the distributed training efficiency. (...) Based on our observations, there is no single configuration that would work for all use cases, as it would highly depend on the model size, model structure, network link bandwidth, etc.	[2, 56, 75, 78]	• No optimal algorithm for all use cases • Optimization challenges
	D407 [57]	One drawback of pipeline parallel training is that there will be some bubble time, where some devices are idle when others are engaged in computation, leading to the waste of computational resources. (...) DeepSpeed's static policy will still offload all model data to the CPU memory, leading to low memory efficiency and high communication over-head.	[2, 57, 75]	• Redundant computation • Communication overhead
	D408 [59]	While in principle one could develop an end-to-end solution by stitching together several existing systems (e.g., Horovod [53] for distributed training, Clipper [19] for serving, and CIEL [40] for simulation), in practice this approach is untenable due to the tight coupling of these components within applications. As a result, researchers and practitioners today build one-off systems for specialized RL applications [58, 41, 54, 44, 49, 5]. (...) To satisfy these requirements, Ray implements a unified interface that can express both task-parallel and actorbased computations.	[17, 59]	• Tight coupling of components • Multiple programming paradigms
	D410 [75]	There are a few areas that we are actively working on to improve Horovod, including: Collecting and sharing learnings about adjusting model parameters for distributed deep learning: Facebook's paper [6] describes the adjustments needed to model hyperparameters to achieve the same or greater accuracy in a distributed training job compared to training the same model on a single GPU, demonstrating the feasibility of training a TensorFlow model on 256 GPUs. We believe this area of deep learning research is still in its early stages and hope to collaborate with other teams about approaches to further scale deep learning training.	[2, 57, 75]	• Cross-node communication challenges • Collaboration with external teams
	D411 [78]	However, large batch training introduces complications into the optimization process that can result in reduced accuracy or longer time to convergence, offsetting the benefit of increased training throughput. (...) for BERT models, careful attention to the placement of layer normalization in BERT-like models is critical to achieving increased accuracies as the model size increases.	[56, 78]	• Error prone utilization • Manual hyperparameter tuning

Table 8. The passages on GPU programming

Cat.	ID	Text Passages	Ref.	Codes
RQ_4: Key Motivating Factors	G1011 [18]	Deep learning workloads are computationally intensive, and optimizing their kernels is difficult and time-consuming. As parallel architectures evolve, kernels must be reoptimized, which makes maintaining codebases difficult over time. The computations that arise when training and using deep neural networks lend themselves naturally to efficient parallel implementations.	[18, 53]	• Optimizing deep-learning kernels • Surging demand for scalability
	G1012 [18]	Parallel processors such as GPUs have played a significant role in the practical implementation of deep neural net-works. The computations that arise when training and using deep neural networks lend themselves naturally to efficient parallel implementations. The efficiency provided by these implementations al-lows researchers to explore significantly higher capacity networks, training them on larger datasets [7].	[18, 65]	• Breakthroughs that provide computational resources • Natural parallelizability • Data availability
	G1013 [18]	The deep learning community has been successful in finding optimized implementations of these kernels, but as the underlying architectures evolve, these kernels must be re-optimized, which is a significant investment. (...) To address this problem, we have created a library similar in intent to BLAS, with optimized routines for deep learning workloads. Our implementation contains routines for GPUs, although similarly to the BLAS library, these routines could be implemented for other platforms. The library is easy to integrate into existing frameworks, and provides optimized performance and memory usage.	[18, 65]	• Optimizing deep-learning kernels (investment) • Easy integration into existing frameworks
	G1014 [18]	Several deep learning projects at Baidu have integrated cuDNN. For example, it has been integrated into PADDLE, Baidu's internal deep learning framework. (...) cuDNN computation is transparent to the user through drop-in integration. The model schema and framework interfaces are completely unchanged. Setting a single compilation flag during installation equips Caffe with cuDNN layer implementations and sets cuDNN as the default computation engine.	[18, 44, 65]	• Integration into existing frameworks • Transparent integration
	G1015 [18]	Our implementation contains routines for GPUs, although similarly to the BLAS library, these routines could be implemented for other platforms. (...) The library exposes a host-callable C language API, but requires that input and output data be resi-dent on the GPU, analogously to cuBLAS.	[18]	• Interaction between GPU and CPU
	G1016 [18]	With cuDNN, it is possible to write programs that train standard convolutional neural networks without writing any parallel code, but simply using cuDNN and cuBLAS. (...) Firstly, deep learning frameworks can focus on higher-level issues rather than close optimization of parallel kernels to specific hardware platforms. Secondly, as parallel architectures evolve, library providers can provide performance portability, in much the same way as the BLAS routines provide performance portability to diverse applications on diverse hardware. Thirdly, a clearer separation of concerns allows specialization: library providers can take advantage of their deep understanding of parallel architectures to provide optimal efficiency. Our goal is to make it much easier for deep learning frameworks to take advantage of parallel hardware.	[18, 36]	• Meeting user requirements
	G1017 [18]	One of the primary goals of cuDNN is to enable the community of neural network frameworks to benefit equally from its APIs. Accordingly, users of cuDNN are not required to adopt any particular software framework, or even data layout. (...) Rather than providing a layer abstraction, we provide lower-level computational primitives, in order to simplify integration with existing deep learning frameworks, each with their own abstractions.	[18, 20]	• Self-contained framework • Lower-level abstractions
	G1031 [36]	The goal of the library is to facilitate machine learning research. This means that the library has a focus on flexibility and extensibility, in order to make sure that nearly any research idea is feasible to implement in the library. The target user base is machine learning researchers. Being "user friendly" for a research user means that it should be easy to understand exactly what the code is doing and configure it very precisely for any desired experiment.	[20, 36]	• Performance • Expert user-base • Flexible and extensible
	G1041 [44]	While deep neural networks have attracted enthusiastic interest within computer vision and beyond, replication of published results can involve months of work by a researcher or engineer. (...) But trained models alone are not sufficient for rapid research progress and emerging commercial applications, and few toolboxes offer truly off-the-shelf deployment of state-of-the-art models—and those that do are often not computationally efficient and thus unsuitable for commercial deployment.	[20, 36, 44]	• Scalability and deployment • Usability
	G1051 [53]	We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images (...) The specific contributions of this paper are as follows: we trained one of the largest convolutional neural networks to date on the subsets of ImageNet used in the ILSVRC-2010 and ILSVRC-2012 competitions [2] and achieved by far the best results ever reported on these datasets. We wrote a highly-optimized GPU implementation of 2D convolution and all the other operations inherent in training convolutional neural networks, which we make available publicly.	[18, 53]	• Task-specific optimizations • Object-recognition
	G1061 [65]	NumPy provides multi-dimensional arrays, the fundamental data structure for scientific computing, and a variety of operations and functions. (...) Deep learning computations principally require linear algebra computations, which is one of NumPy's strengths. However, NumPy does not support calculations on GPUs. This was the motivation to develop CuPy – to fully benefit from fast computations using the latest GPUs with a NumPy-compatible interface.	[18, 65]	• Existing tools not GPU compatible • Deep learning involves linear algebra computations
	G1062 [65]	CuPy 1 is an open-source library with NumPy syntax that increases speed by doing matrix operations on NVIDIA GPUs. It is accelerated with the CUDA platform from NVIDIA and also uses CUDA-related libraries, including cuBLAS, cuDNN, cuRAND, cuSOLVER, cuSPARSE, and NCCL, to make full use of the GPU architecture. CuPy's interface is highly compatible with NumPy.	[65]	• Building on existing tools
	G1071 [20]	With Torch7, we aim at providing a framework with three main advantages: (1) it should ease the development of numerical algorithms, (2) it should be easily extended (including the use of other libraries), and (3) it should be fast. (...) We found that a scripting (interpreted) language with a good C API appears as a convenient solution to "satisfy" the constraint (2). (...) Among existing scripting languages1 finding the ones that satisfy condition (3) severely restricted our choice. We chose Lua, the fastest interpreted language (with also the fastest Just In Time (JIT) compiler2) we could find.	[20, 36, 44]	• Performance • Usability • Extensibility
RQ_3: Critical Factors	G2011 [18]	It can provide immediate efficiency gains, and it is rigorously tested and maintained in order to be reliable and performant across a range of different processor architectures. Importantly, our library is designed to use the minimum possible amount of auxiliary memory, which frees up scarce memory for larger models and datasets. We also optimize performance across a wide range of potential use cases, including small mini-batch sizes.	[18, 20, 44]	• Scalability • Performance
	G2012 [18]	Firstly, deep learning frameworks can focus on higher-level issues rather than close optimization of parallel kernels to specific hardware platforms. Secondly, as parallel architectures evolve, library providers can provide performance portability, in much the same way as the BLAS routines provide performance portability to diverse applications on diverse hardware. Thirdly, a clearer separation of concerns allows specialization: library providers can take advantage of their deep understanding of parallel architectures to provide optimal efficiency. Our goal is to make it much easier for deep learning frameworks to take advantage of parallel hardware.	[18, 44]	• Separation of concerns • Focus on higher-level design • Performance portability
	G2021 [20]	Torch7 has been designed with efficiency in mind, leveraging SSE when possible and supporting two ways of parallelization: OpenMP and CUDA. Open Multi-Processing (OpenMP) provides a shared memory CPU parallelization framework on C/C++ and Fortran languages on almost every operating system and compiler toolset. (...) Torch7 is designed to be easily interfaced with third-party software thanks to Lua's interface	[20, 44]	• Performance • Cross-framework compatibility • Heterogenous hardware
	G2041 [44]	Caffe fits industry and internet-scale media needs by CUDA GPU computation, processing over 40 million images a day on a single K40 or Titan GPU (...) Switching between a CPU and GPU implementation is exactly one function call. (...) Separation of representation and implementation. Caffe model definitions are written as config files using the Protocol Buffer language. (...) The code is written in clean, efficient C++, with CUDA used for GPU computation, and nearly complete, well-supported bindings to Python/Numpy and MATLAB.	[18, 20, 44]	• Heterogenous hardware • Scalability • Separation of concerns • Usability
	G2051 [53]	Current GPUs are particularly well-suited to cross-GPU parallelization, as they are able to read from and write to one another's memory directly, without going through host machine memory. The parallelization scheme that we employ essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers.	[18, 53]	• Inter-GPU communication
	G2061 [65]	CuPy implements many functions on cupy.ndarray objects. See the reference 2 for the supported subset of NumPy API. Since CuPy covers most NumPy features, reading the NumPy documentation can be helpful for using CuPy.	[65]	• Declarative programming
RQ_2: Evaluation Metrics	G3011 [18]	The convolution routines in cuDNN provide competitive performance with zero auxiliary memory required. Figure 2 shows the performance on an NVIDIA Tesla K40 of three convolution implementations: cuDNN, Caffe, and cuda-convnet2. We evaluated these implementations using the layer configurations shown in table 2, which are commonly used for benchmarking convolution performance [1], and quote average throughput for all these layers. (...) Compared to Caffe, cuDNN per-formance ranges from 1.0× to 1.41×. Importantly, even with a small mini-batch size of only 16, cuDNN performance is still 86% of maximum performance, which shows that our implementation performs well across the convolution parameter space.	[18, 36, 44]	• Model Architecture (convolution performance, mini-batch evaluation, efficiency)
	G3012 [18]	We present a library of efficient implementations of deep learning primitives. (...) Convolutional Neural Networks (CNNs) [14] are an important and successful class of deep networks. (...) We are also using cuDNN in other domains besides image processing, such as speech and language. cuDNN's ability to convolve non-square inputs with asymmetric padding is particularly useful for these other domains.	[18, 65]	• Domains: deep learning, CNNs, general purpose
	G3013 [18]	NVIDIA provides a matrix multiplication routine that achieves a substantial fraction of floating-point throughput on GPUs. (...) Over-all, training time for 200 iterations improved by 36%, when training the bvlc reference caffenet model, using cuDNN R1 on an NVIDIA Tesla K40. (...) Figure 2 shows the performance on an NVIDIA Tesla K40 of three convolution implementations: cuDNN, Caffe, and cuda-convnet2. cuDNN performance ranges from 0.8× to 2.25× that of cuda-convnet2, with an advantage at smaller batch sizes. Compared to Caffe, cuDNN per-formance ranges from 1.0× to 1.41×. (...) This data illustrates how cuDNN provides performance portability across GPU architectures, with no need for users to retune their code as GPU architectures evolve.	[18]	• Evaluation: performance, benchmarking and portability
	G3031 [36]	Because the philosophy that Pylearn2 developers should write features when they are needed, and because most Pylearn2 developers so far have been deep learning researchers, Pylearn2 mostly con-tains deep learning models or models that are used as building blocks for deep architectures. This includes autoencoders [6], RBMs [47] including RBMs with Gaussian visible units [54], DBMs [45], MLPs [44], convolutional networks [33], and local coordinate coding [56].	[18, 36, 44]	• Model architectures
	G3041 [44]	It powers on-going research projects, large-scale industrial applications, and startup prototypes in vision, speech, and multimedia. (...) The same models can be run in CPU or GPU mode on a variety of hardware (...) The code is written in clean, efficient C++, with CUDA used for GPU computation, and nearly complete, well-supported bindings to Python/Numpy and MATLAB. (...) Separation of representation and implementation. Caffe model definitions are written as config files using the Protocol Buffer language.	[18, 36, 44]	• Deployment • Model architectures
	G3051 [53]	In the left panel of Figure 4 we qualitatively assess what the network has learned by computing its top-5 predictions on eight test images. Notice that even off-center objects, such as the mite in the top-left, can be recognized by the net. Most of the top-5 labels appear reasonable.	[53]	• Qualitative evaluation
	G3061 [65]	For example, a Python-based probabilistic modeling software, Pomegranate [4], uses CuPy as its GPU backend. We believe this is thanks to CuPy's NumPy-like design and strong performance based on NVIDIA libraries. (...) CuPy runs NumPy code at GPU calculation speeds. Though developed as the array back-end for the deep learning framework Chainer, it can also be used for general purpose, scientific computing on GPU.	[18, 65]	• Domain: scientific computing, probabilistic modelling
	G4011 [18]	Optimizing and maintaining all these specializations is a difficult task. As we envision this library being maintained for some time, and being ported to yet-to-be-conceived future architectures, we searched for something simpler that would perform more robustly across the parameter space and be easier to port to new architectures. (...) We are considering several avenues for expanding the performance and functionality of cuDNN. (...) Finally, we would like this library to help people use multiple GPUs to accelerate training.	[18, 53]	• Outstanding challenges due to future architectures • Multi-GPU training
	G4012 [18]	The deep learning community has been successful in finding optimized implementations of these kernels, but as the underlying architectures evolve, these kernels must be re-optimized, which is a significant investment. Optimizing these kernels requires a deep understanding of the underlying processor architecture, with careful scheduling of data movement, on-chip memory placement, register blocking, and other optimizations in order to get acceptable performance.	[18]	• Usability: Optimization of kernels is time-consuming

Table 8 – continued from previous page

Cat.	ID	Text Passages	Ref.	Codes
RQ_2: Limitations & Challenges	G4013 [18]	There are several ways to implement convolutions efficiently. Our goal is to provide performance as close as possible to matrix multiplication, while using no auxiliary memory. GPU memory is high bandwidth, but low capacity, and is therefore a scarce resource. (...) Some approaches, like lowering convolutions to matrix multiplication or using Fast Fourier Transform (FFT), have drawbacks (...) Another common approach is to compute the convolutions directly. This can be very efficient, but requires a large number of specialized implementations to handle the many corner cases implicit in the 11-dimensional parameter space of the convolutions. (...) Additionally, the FFT based approach does not per-form efficiently when the striding parameters u and v are greater than 1, which is common in many state-of-the art networks	[18, 18]	• Algorithmic limitation: high memory usage (FFT, matrix multip. operations), direct compute (specialized implementations)
	G4041 [44]	While deep neural networks have attracted enthusiastic interest within computer vision and beyond, replication of published results can involve months of work by a researcher or engineer. Sometimes researchers deem it worthwhile to release trained models along with the paper advertising their performance. But trained models alone are not sufficient for rapid research progress and emerging commercial applications, and few toolboxes offer truly off-the-shelf deployment of state-of-the-art models—and those that do are often not computationally efficient and thus unsuitable for commercial deployment.	[44]	• Usability: Replication of results is challenging
	G4051 [53]	The parallelization scheme that we employ essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers. This means that, for example, the kernels of layer 3 take input from all kernel maps in layer 2. However, kernels in layer 4 take input only from those kernel maps in layer 3 which reside on the same GPU. Choosing the pattern of connectivity is a problem for cross-validation, but this allows us to precisely tune the amount of communication until it is an acceptable fraction of the amount of computation.	[18, 53]	• Communication overhead
	G4061 [65]	It is accelerated with the CUDA platform from NVIDIA and also uses CUDA-related libraries, including cuBLAS, cuDNN, cuRAND, cuSOLVER, cuSPARSE, and NCCL, to make full use of the GPU architecture. For small matrices, CuPy is slower than NumPy since there is some overhead in CuPy from the CUDA kernel launch. For larger matrices, the overhead is small compared to the actual GPU computation, and CuPy is up to six times faster than CPU-based NumPy.	[18, 65]	• Limitations: small matrix performance