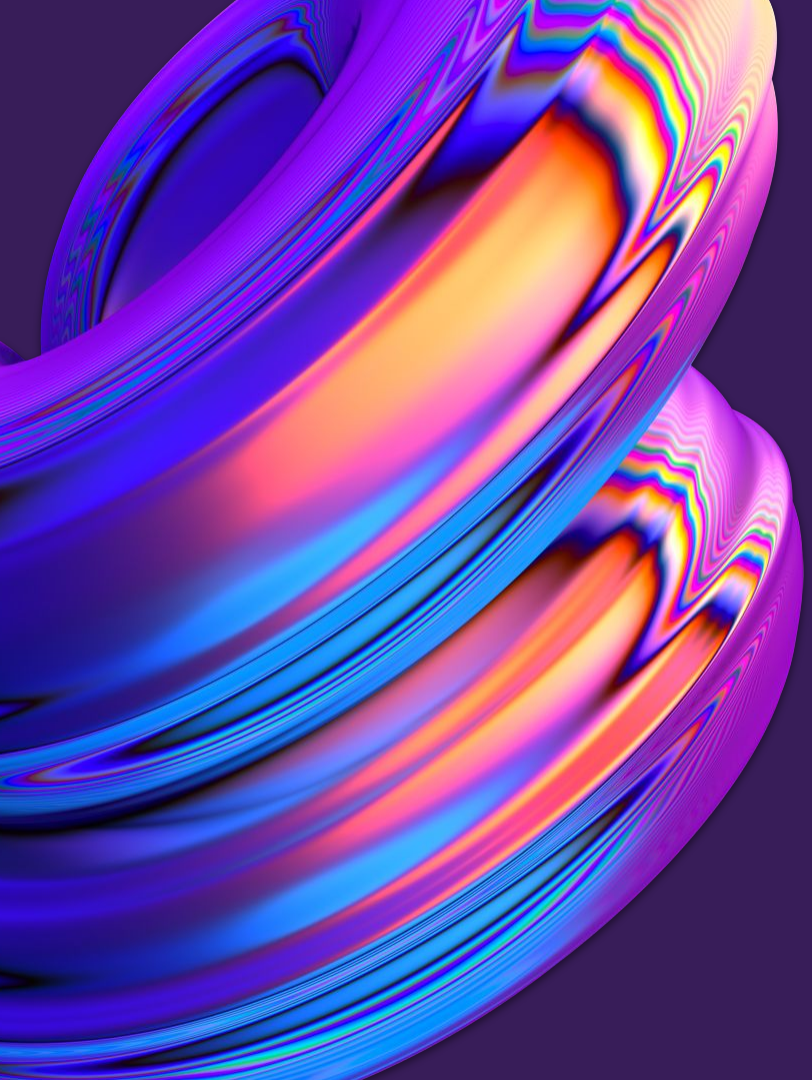


MERANTIX MOMENTUM

Squirrel meets safe·trAIn

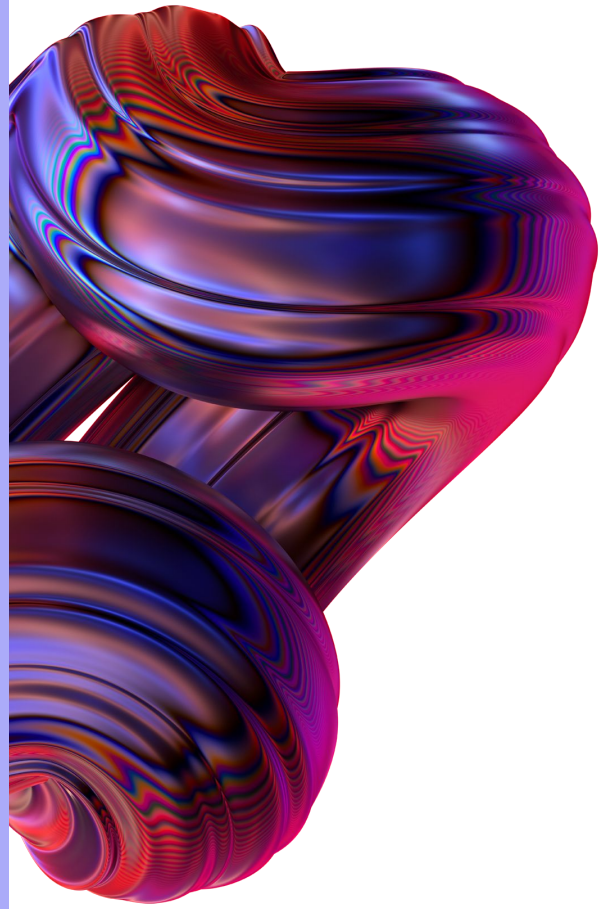
Merantix Momentum Research

Alexander Koenig
ML Research Engineer

6th December 2022

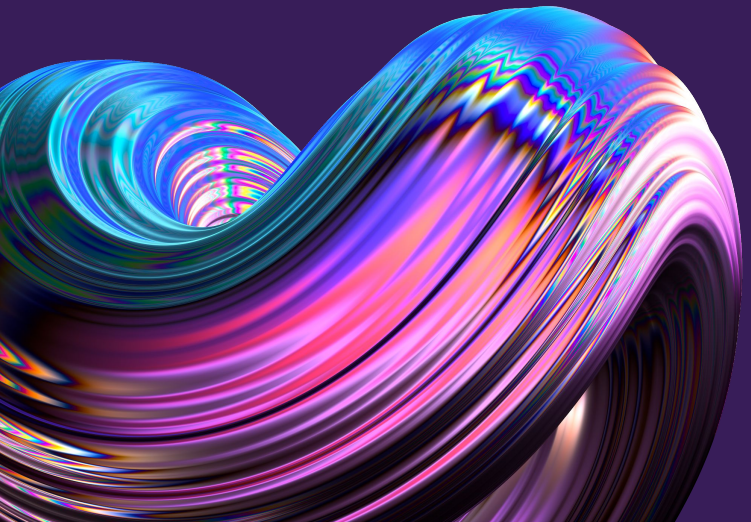
Agenda

1. Motivation
2. What is Squirrel?
3. Why do we use Squirrel?
4. How can you use Squirrel?
5. Conclusion



MERANTIX MOMENTUM

MERANTIX MOMENTUM



Motivation

Feeding the Beast

Fig. 1: Data Loading

Figure: Dr. Thomas Wollmann: Squirrel - Efficient Data Loading for Large-Scale Deep Learning

GPU Stall

Problem: Low GPU utilization leads to longer training times and high cost!

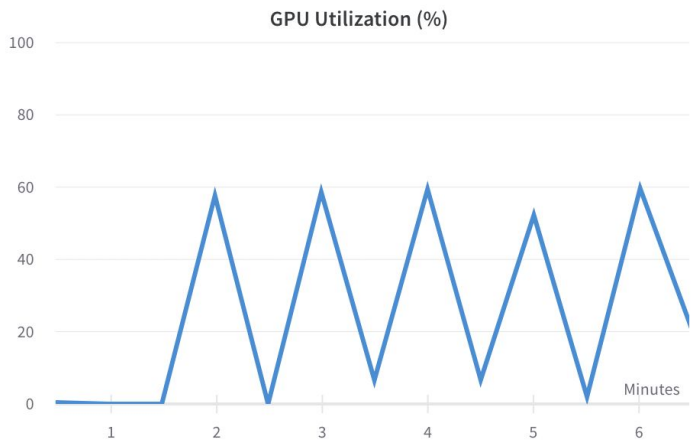


Fig. 2: GPU stalls - bad!

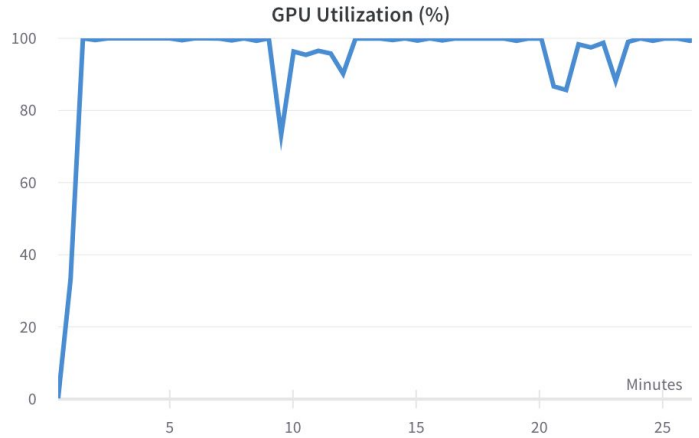


Fig. 3: GPU highly utilized - good!

Data Pipelines

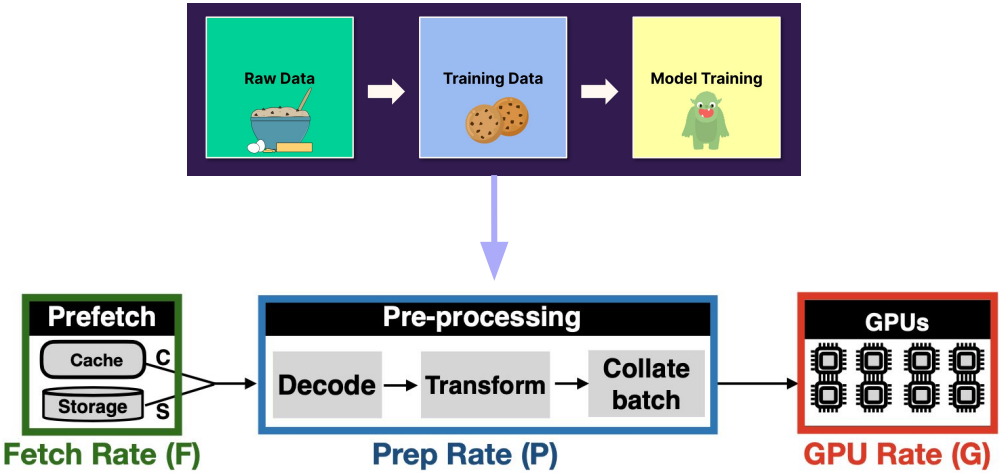


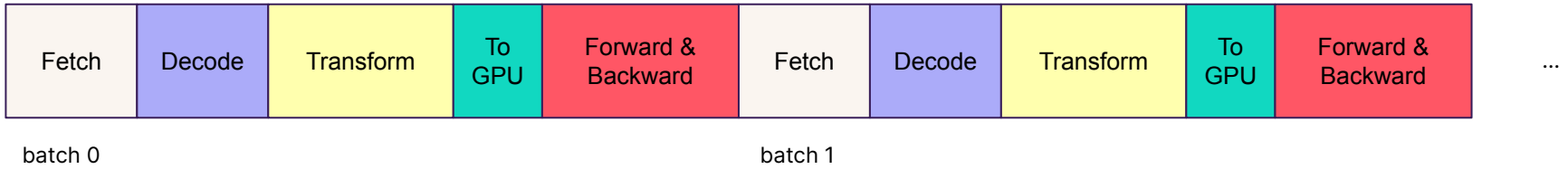
Figure 1: Data Pipeline in DNN training. This figure shows the different stages in the data pipeline.

"DNNs spend up to 65% of the epoch time in data pre-processing, primarily on redundant decoding"

Mohan, Jayashree, et al. "Analyzing and mitigating data stalls in DNN training" 2020

Implementing Data Pipelines

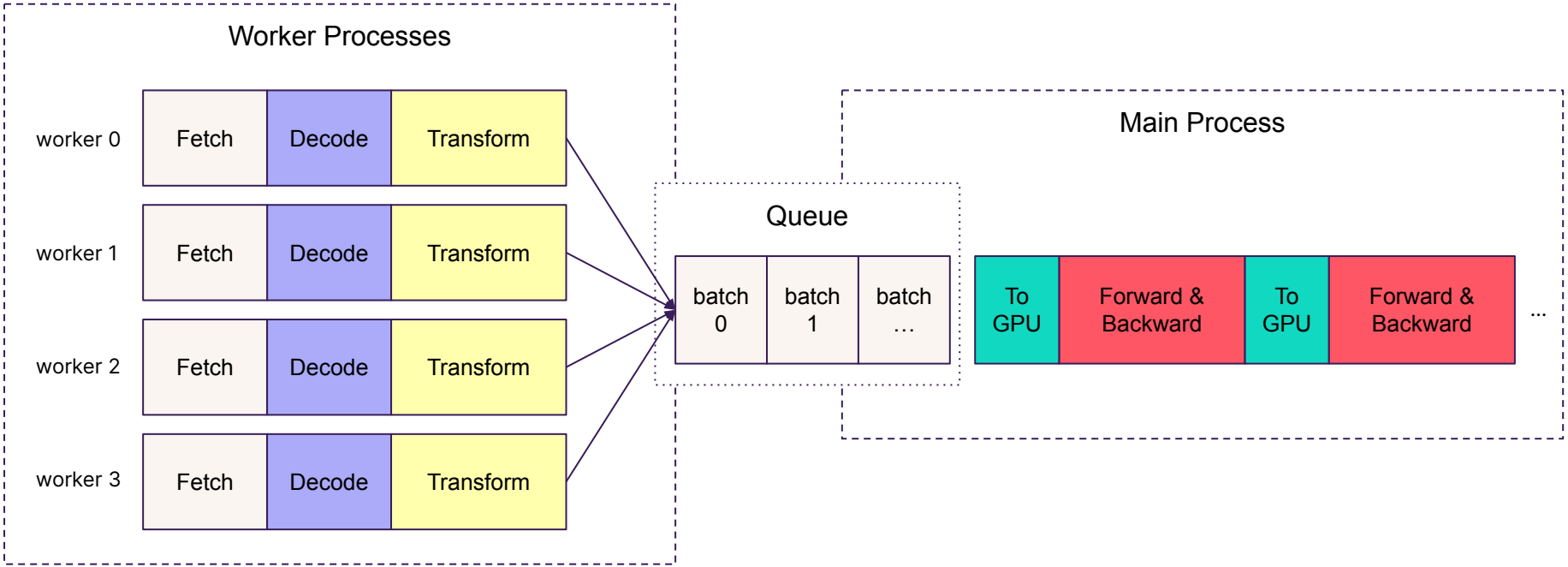
Naive Implementation - Single Process



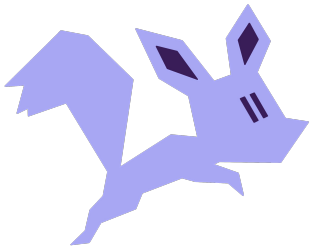
Implementing Data Pipelines

Implementation - Multi-Process

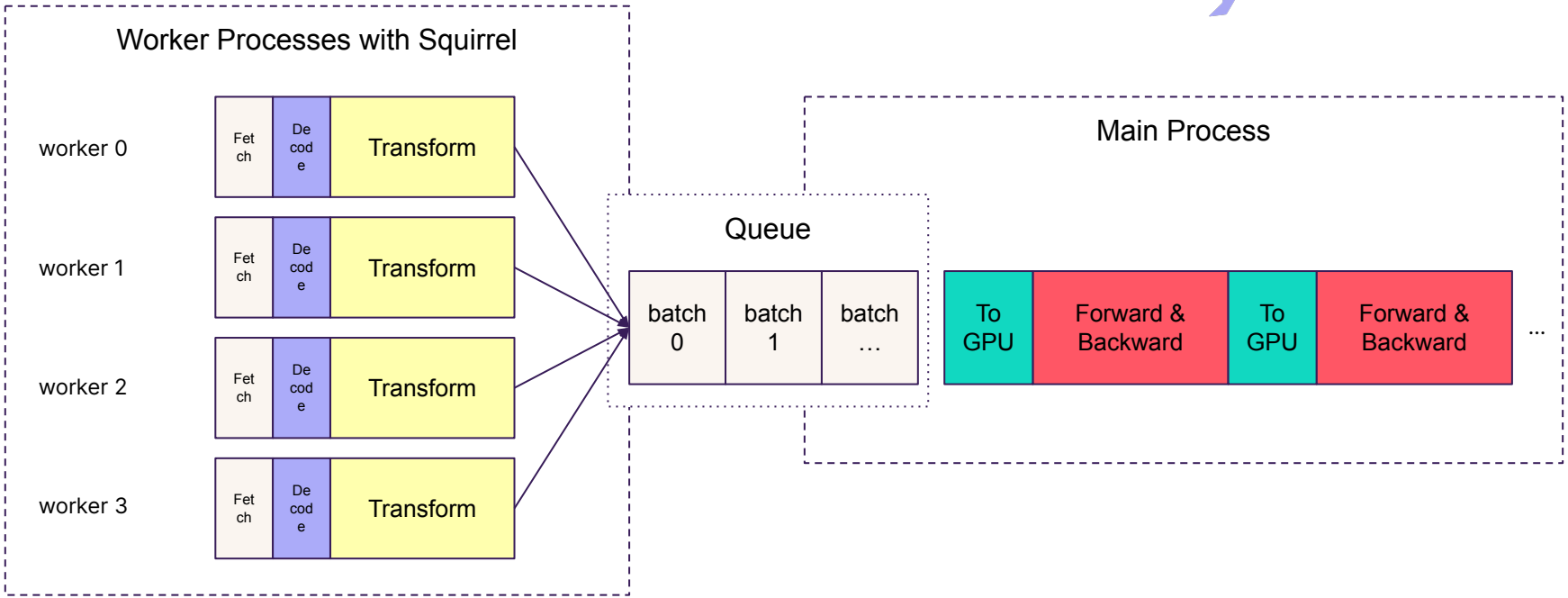
Problem: workers might not be fast enough to fill up queue and keep GPU busy

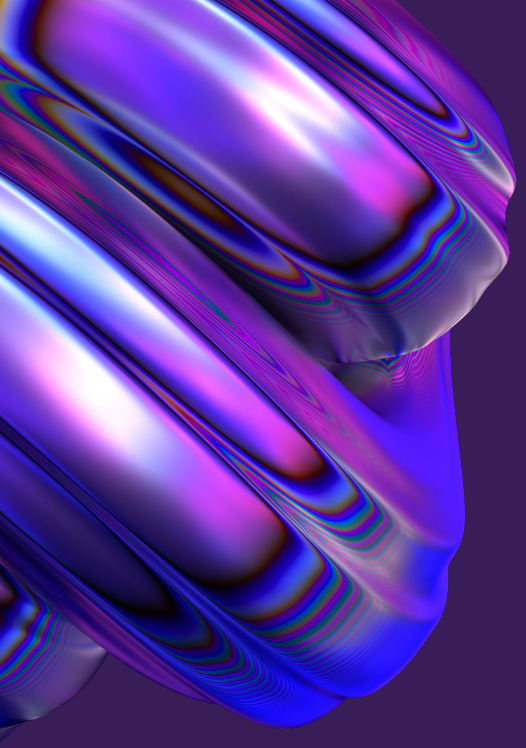


Implementing Data Pipelines



Implementation - Multi-Process





MERANTIX MOMENTUM

What is Squirrel?

Squirrel

GitHub

Merantix Momentum

19 followers · Germany · https://merantix-momentum.com/

merantix-momentum / **squirrel-core** Public

Squirrel Core

Share, load, and transform data in a collaborative, flexible, and efficient way

python 3.8 | pypi package v0.18.0 | conda-forge v0.18.0 | docs passing | Downloads 42k | License Apache 2.0

DOI 10.5281/zenodo.7311129 | Website Merantix Momentum | slack chat

```
pip install squirrel-core
squirrel-core-datasets
```

- **Speed**: Avoid GPU stall
- **Cost**: Reduced GPU time and less IO cost through loading in bundles
- **Flexibility**: Customizable data schemes
- **Collaboration**: Enables sharing of data and versioning

Core Components

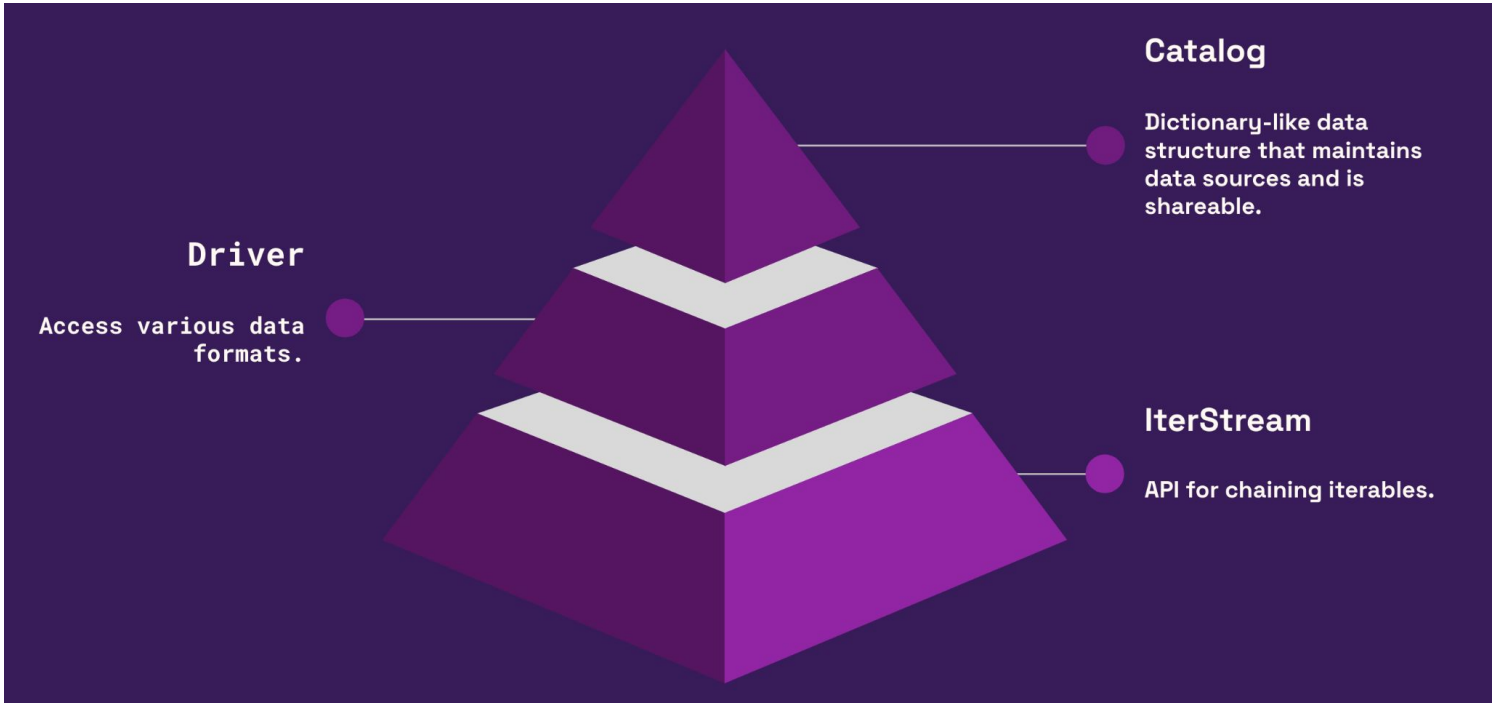


Figure: Dr. Thomas Wollmann: Squirrel - Efficient Data Loading for Large-Scale Deep Learning

A) IterStream API

```python
from squirrel.iterstream import IterableSource
from squirrel.iterstream.base import Composable

it = (
    IterableSource(range(2000))  # your data source
    .shuffle(1000)
    .filter(lambda x: x % 2 == 0)
    .map(lambda x: x * 2)
    .async_map(lambda x: x**2, max_workers=100)
    .batched(100)
)

for data in it:
    assert len(data) == 100  # we get batches of 100

# Composable provides more stream manipulation
assert issubclass(IterableSource, Composable)
```

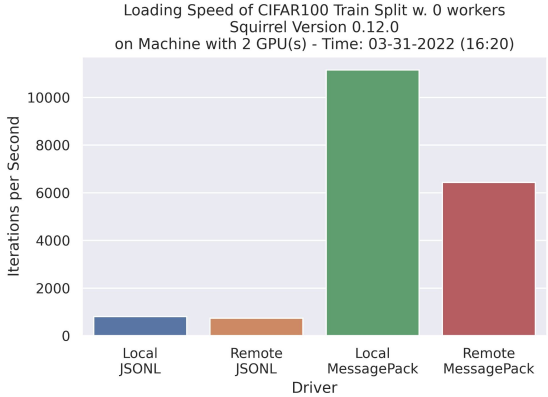
- Basic stream manipulation
  - .shuffle()
  - .filter()
  - .map()
  - .batched()
- More stream manipulation
  - .async_map()
  - .dask_map()
  - .loop()
  - .monitor()
  - ...

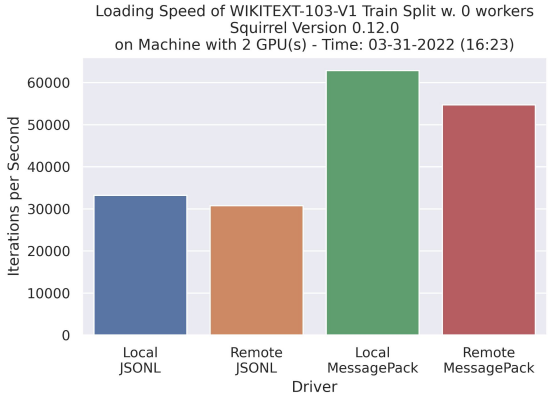
B) Driver

```python
from squirrel.driver.driver import Driver, IterDriver, MapDriver

# Drivers of type MapDriver (implements .get(), .keys())
from squirrel.driver.msgpack import MessagepackDriver
from squirrel.driver.zarr import ZarrDriver
from squirrel.driver.jsonl import JsonlDriver

# Drivers of type FileDriver (implements .open())
from squirrel.driver.file_driver import FileDriver
from squirrel.driver.csv_driver import CsvDriver
```

- Driver is the abstract base class
- IterDriver(Driver) implements .get_iter()
- MapDriver(IterDriver) implements .get(), .keys()

- Easily extensible to e.g. multi-modal data
- More drivers are added as we speak!





Dataloading Landscape

Hugging Face

ACTIVELOOP

hub

deeplake

PyTorch

TORCHVISION

Squirrel Datasets Core

Drivers of Squirrel-Datasets-Core

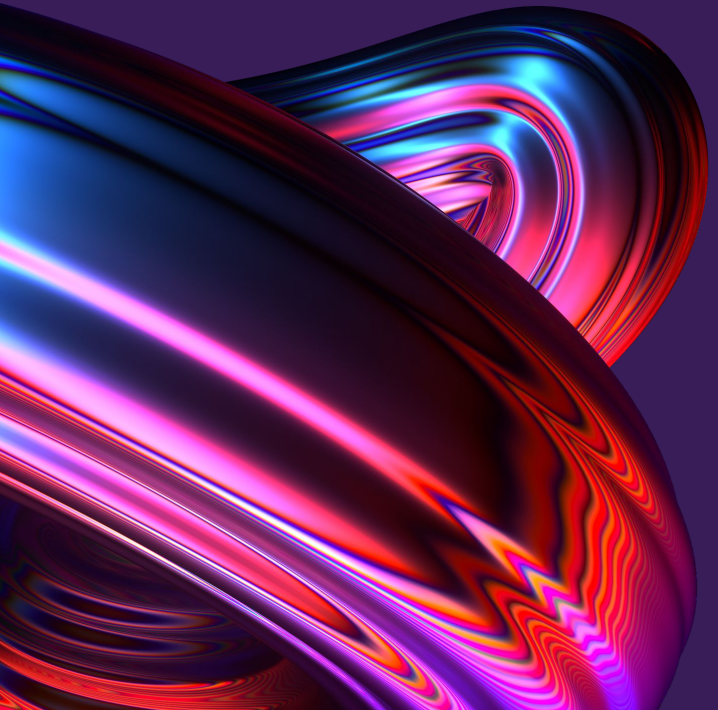
```python
from squirrel_datasets_core.driver import HubDriver, HuggingfaceDriver, TorchvisionDriver, DeeplakeDriver

it_hub = HubDriver("hub://activeloop/cifar10-train").get_iter()
it_dl = DeeplakeDriver("hub://activeloop/cifar10-train").get_iter()
it_hf = HuggingfaceDriver("cifar10").get_iter("train")
it_th = TorchvisionDriver("cifar10", download=True).get_iter()

print(next(iter(it_hub)))
# {'images': Tensor(key='images', index=Index([0])), 'labels': Tensor(key='labels', index=Index([0]))}
print(next(iter(it_dl)))
# {'images': Tensor(key='images', index=Index([0])), 'labels': Tensor(key='labels', index=Index([0]))}
print(next(iter(it_hf)))
# {'img': <PIL.PngImagePlugin.PngImageFile image mode=RGB size=32x32 at 0x14401F970>, 'label': 0}
print(next(iter(it_th)))
# (<PIL.Image.Image image mode=RGB size=32x32 at 0x143FD9C40>, 6)
```

C) Catalog API

```python
from squirrel.catalog import Catalog, Source

url = "gs://project/cifar10_squirrel"
cat_file = "catalog.yaml"
source_key = "train"

cat_source = Source(
    driver_name=MessagepackDriver.name,
    driver_kwargs={"url": url},
    metadata={
        "author": "Alexander Koenig",
        "license": "Only internal use!",
    },
)

cat = Catalog()
cat[source_key] = cat_source
Catalog.to_file(cat, cat_file)
```

```yaml
!YamlCatalog
version: 0.16.0
sources:
- !YamlSource
 identifier: train
 driver_name: messagepack
 driver_kwargs:
   url: gs://project/cifar10_squirrel
 version: 1
 metadata:
   author: Alexander Koenig
   license: Only internal use!
```

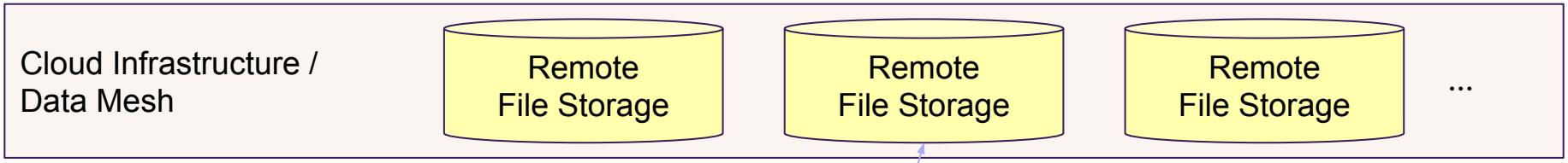
catalog.yaml



Why do we use Squirrel?

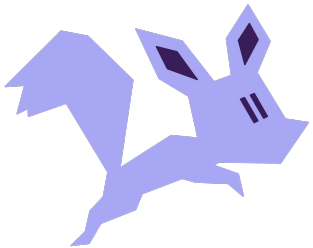
Sharing Catalogs

Idea: instead of passing datasets around, pass versioned configurations of data loaders that load from a decentralized **data mesh**

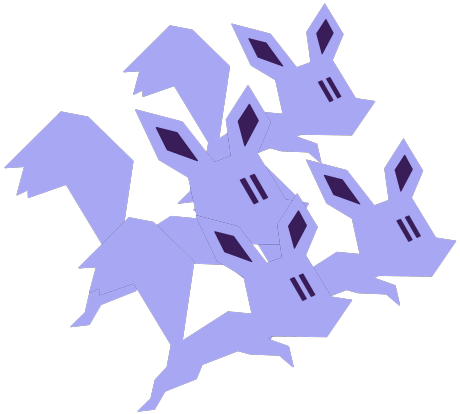



```
!YamlCatalog
version: 0.16.0
sources:
- !YamlSource
 identifier: train
 driver_name: messagepack
 driver_kwargs:
   url: gs://project/cifar10_squirrel
 version: 1
 metadata:
   author: Alexander Koenig
   license: Only internal use!
```

catalog.yaml

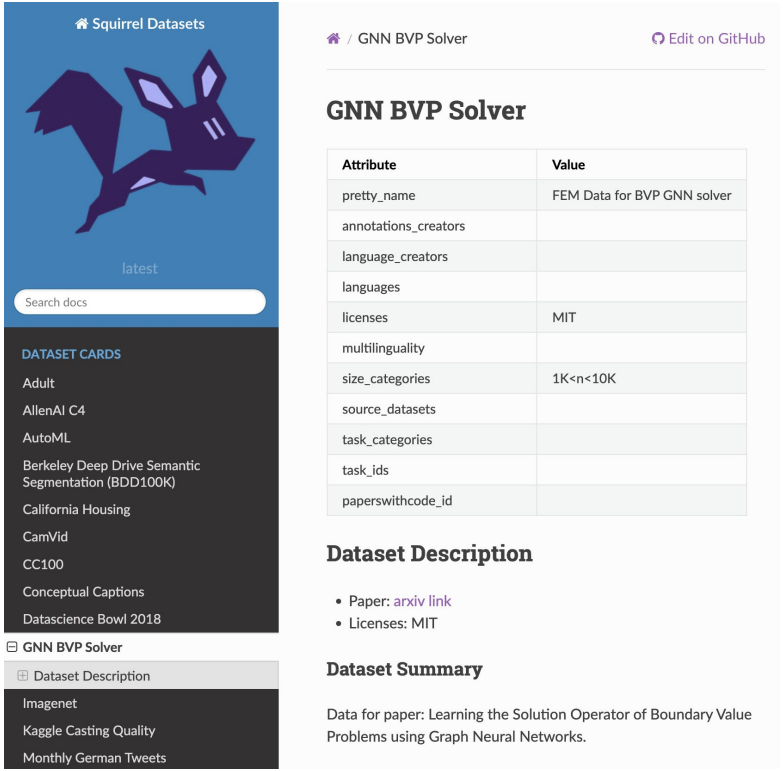


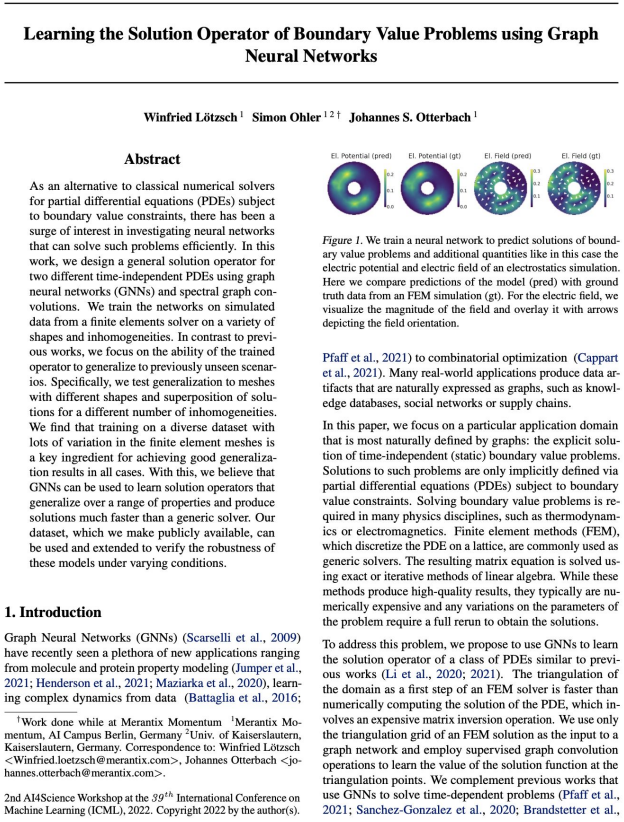
Developer



Merantix
Research Team

Sharing Datasets





Learning the Solution Operator of Boundary Value Problems using Graph Neural Networks

Winfried Lötzsch [1] Simon Ohler [1 2 †] Johannes S. Otterbach [1]

Abstract

As an alternative to classical numerical solvers for partial differential equations (PDEs) subject to boundary value constraints, there has been a surge of interest in investigating neural networks that can solve such problems efficiently. In this work, we design a general solution operator for two different time-independent PDEs using graph neural networks (GNNs) and spectral graph convolutions. We train the networks on simulated data from a finite elements solver on a variety of shapes and inhomogeneities. In contrast to previous works, we focus on the ability of the trained operator to generalize to previously unseen scenarios. Specifically, we test generalization to meshes with different shapes and superposition of solutions for a different number of inhomogeneities. We find that training on a diverse dataset with lots of variation in the finite element meshes is a key ingredient for achieving good generalization results in all cases. With this, we believe that GNNs can be used to learn solution operators that generalize over a range of properties and produce solutions much faster than a generic solver. Our dataset, which we make publicly available, can be used and extended to verify the robustness of these models under varying conditions.

1. Introduction

Graph Neural Networks (GNNs) (Scarselli et al., 2009) have recently seen a plethora of new applications ranging from molecule and protein property modeling (Jumper et al., 2021; Henderson et al., 2021; Maziarka et al., 2020), learning complex dynamics from data (Battaglia et al., 2016; Pfaff et al., 2021) to combinatorial optimization (Cappart et al., 2021). Many real-world applications produce data artifacts that are naturally expressed as graphs, such as knowledge databases, social networks or supply chains.

†Work done while at Merantix Momentum [1]Merantix Momentum, AI Campus Berlin, Germany [2]Univ. of Kaiserslautern, Kaiserslautern, Germany. Correspondence to: Winfried Lötzsch <Winfried.loetzsch@merantix.com>, Johannes Otterbach <johannes.otterbach@merantix.com>.

2nd AI4Science Workshop at the 39^{th} International Conference on Machine Learning (ICML), 2022.

Figure 1. We train a neural network to predict solutions of boundary value problems and additional quantities like in this case the electric potential and electric field of an electrostatics simulation. Here we compare predictions of the model (pred) with ground truth data from an FEM simulation (gt). For the electric field, we visualize the magnitude of the field and overlay it with arrows depicting the field orientation.

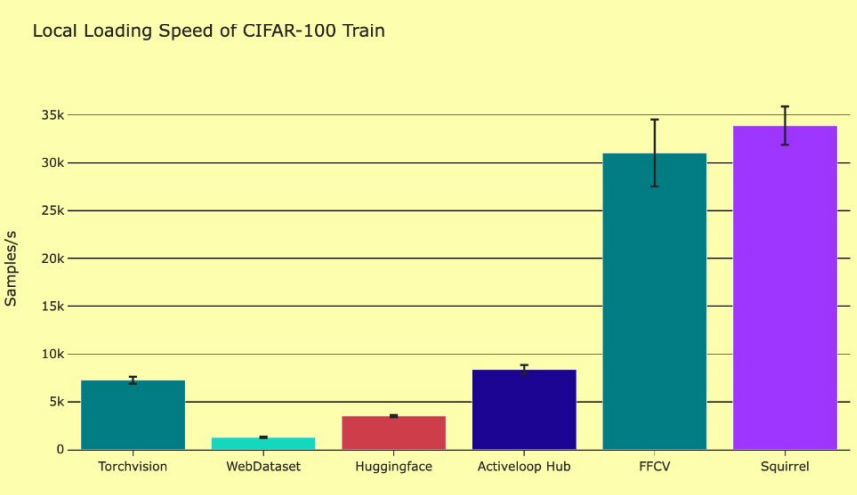
In this paper, we focus on a particular application domain that is most naturally defined by graphs: the explicit solution of time-independent (static) boundary value problems. Solutions to such problems are only implicitly defined via partial differential equations (PDEs) subject to boundary value constraints. Solving boundary value problems is required in many physics disciplines, such as thermodynamics or electromagnetics. Finite element methods (FEM), which discretize the PDE on a lattice, are commonly used as generic solvers. The resulting matrix equation is solved using exact or iterative methods of linear algebra. While these methods produce high-quality results, they typically are numerically expensive and any variations on the parameters of the problem require a full rerun to obtain the solutions.

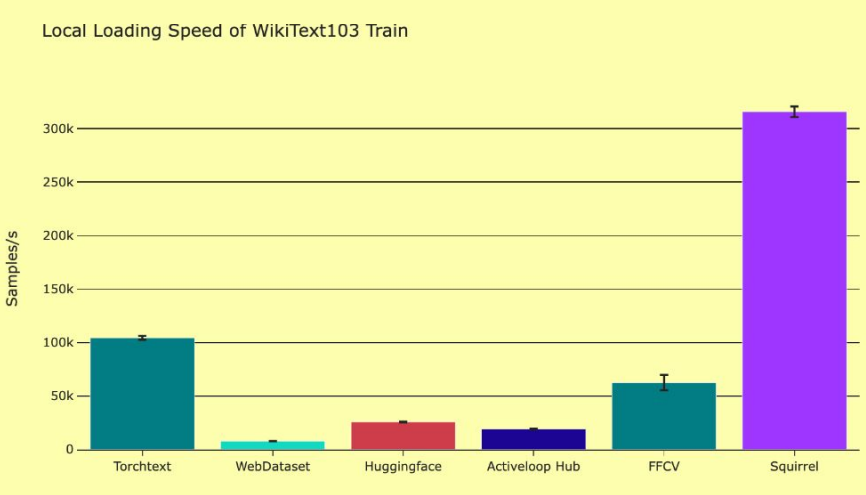
To address this problem, we propose to use GNNs to learn the solution operator of a class of PDEs similar to previous works (Li et al., 2020; 2021). The triangulation of the domain as a first step of an FEM solver is faster than numerically computing the solution of the PDE, which involves an expensive matrix inversion operation. We use only the triangulation grid of an FEM solution as the input to a graph network and employ supervised graph convolution operations to learn the value of the solution function at the triangulation points. We complement previous works that use GNNs to solve time-dependent problems (Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020; Brandstetter et al.,

AI4Science Workshop Paper at ICML, 2022

Benchmarks (1/3) - Local Loading

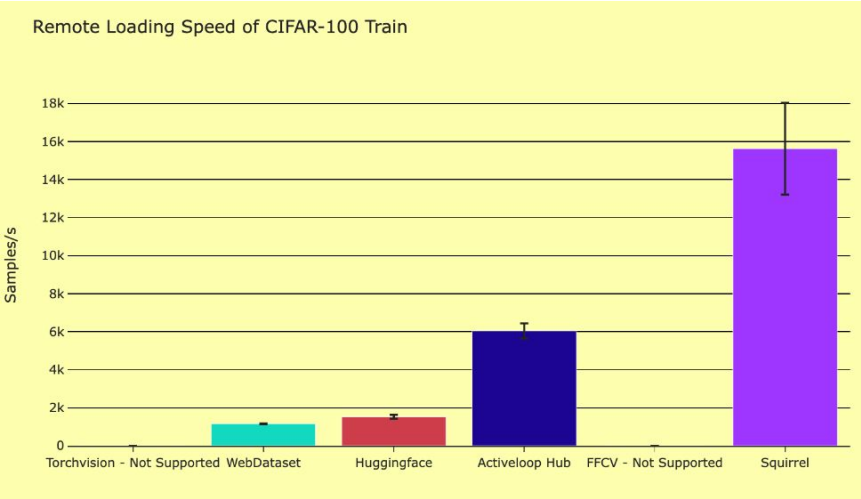
- Takeaway: Squirrel is en-par with the fastest framework for images and fastest for text

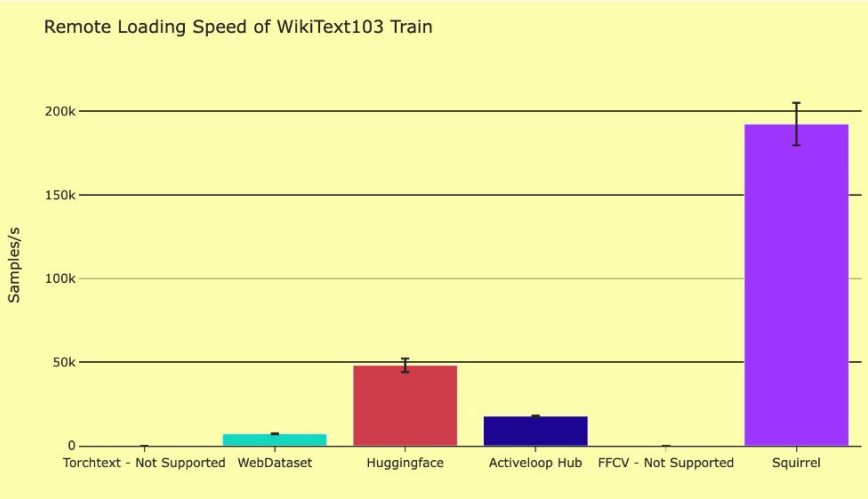




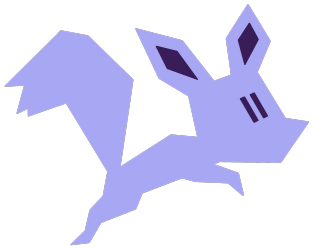
Benchmarks (2/3) – Remote Loading

- Takeaway: Squirrel offers remote loading and outperforms competitors by a large margin

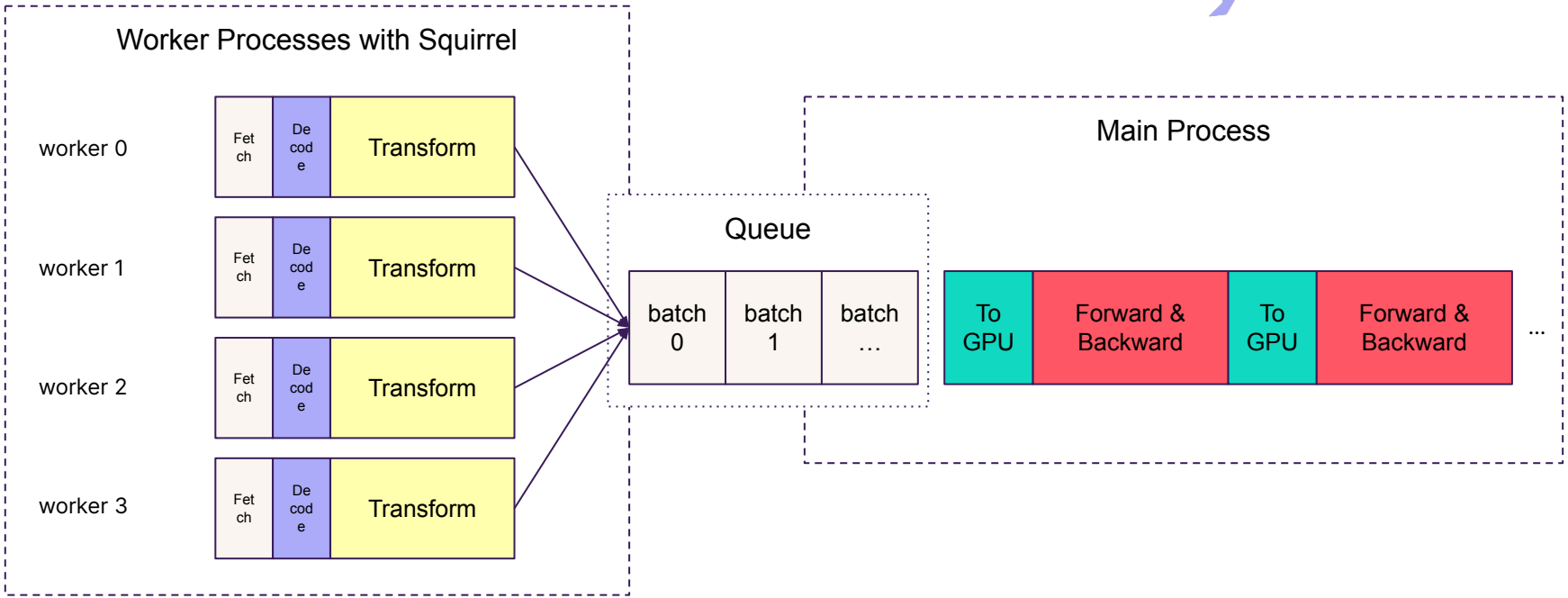




Implementing Data Pipelines

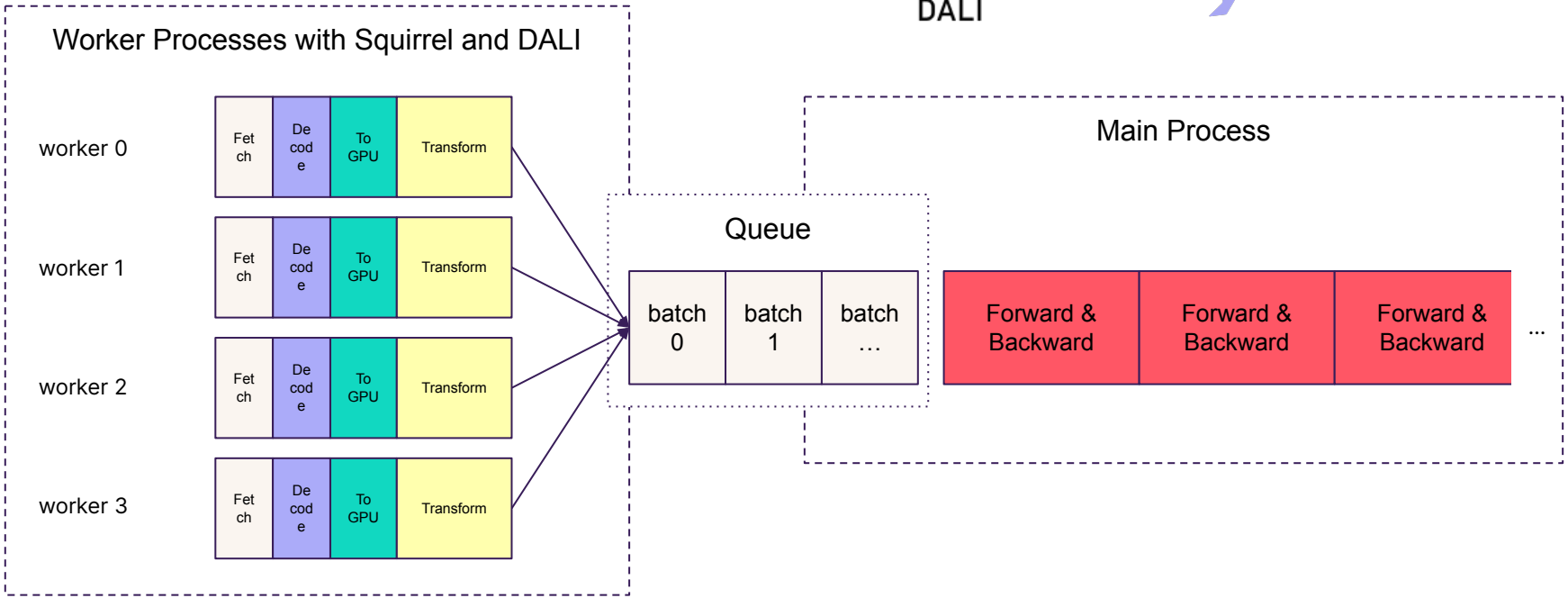


Implementation - Multi-Process



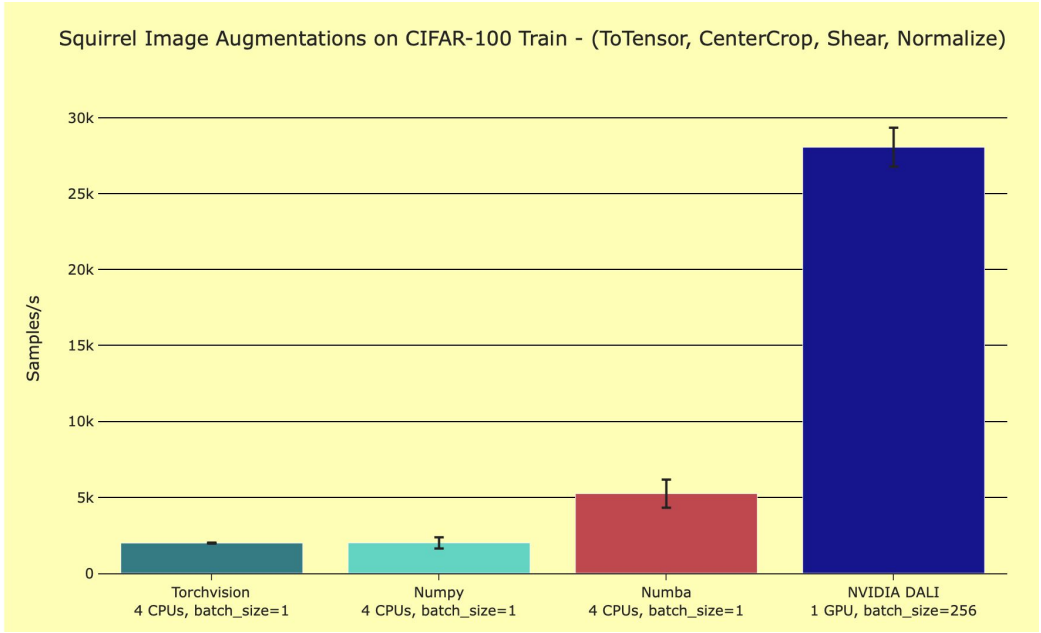
Implementing Data Pipelines

Implementation - Multi-Process

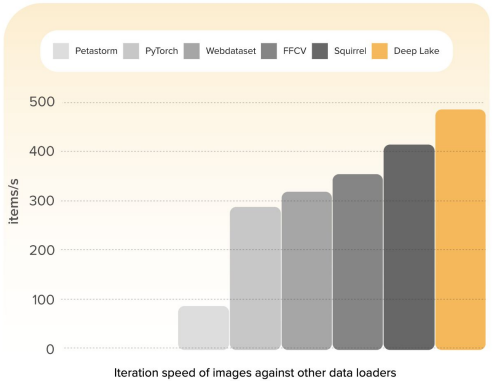


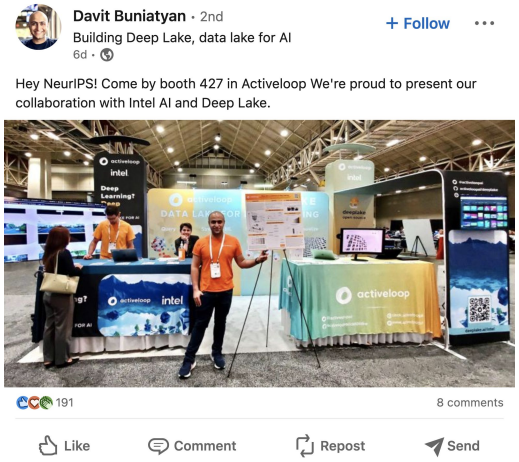
Benchmarks (3/3) – Data Augmentations

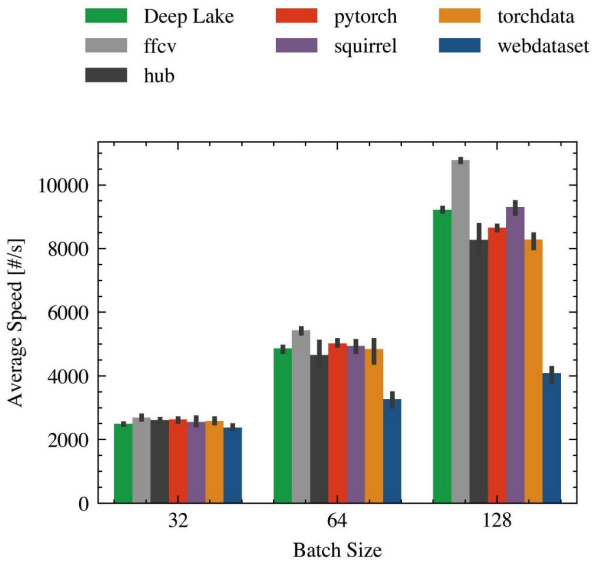
- Takeaway: Squirrel is usable as a backend for the highly-optimized NVIDIA DALI library that runs heavy run-time augmentations on GPU with almost no performance decrease



Third-Party Benchmarks

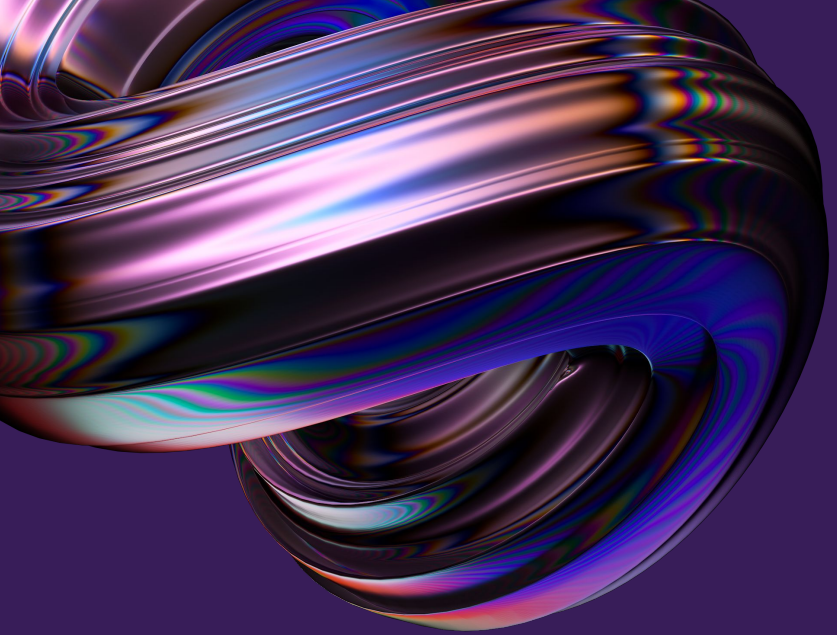






Hambardzumyan, Sasun, et al. "Deep Lake: a Lakehouse for Deep Learning." *arXiv preprint arXiv:2209.10785* (2022).

Ofeidis, Iason, Diego Kiedanski, and Leandros Tassiulas. "An Overview of the Data-Loader Landscape: Comparative Performance Analysis." *arXiv preprint arXiv:2209.13705* (2022).



MERANTIX MOMENTUM

How can you use Squirrel?

Adopting Squirrel

Native PyTorch

```
train_data/
  images/
    image_01.jpg
    image_02.jpg
  seg_masks/
    seg_mask_01.png
    seg_mask_02.png
```

```python
import typing as t
import torch.utils.data as tud

class MyIterableDataset(tud.IterableDataset):
    def __iter__(self) -> t.Iterator:
        return iter([...])

ds = MyIterableDataset()
loader = tud.DataLoader(ds)

for item in loader:
    # ... train step
```

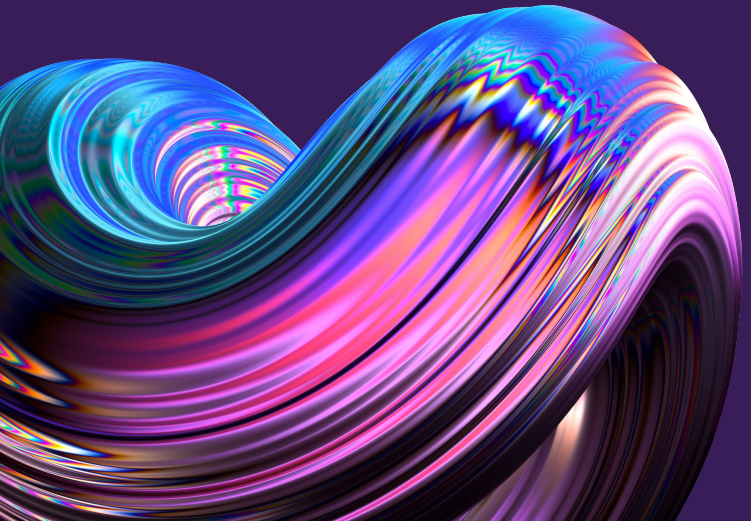
Squirrel & Pytorch

```
train_data/
  shard_1.gz
  shard_2.gz
  shard_3.gz
  shard_3.gz
```

```python
import torch.utils.data as tud
from squirrel.driver.msgpack import MessagepackDriver

url = "gs://my-project-data/train_data"

it = MessagepackDriver(url).get_iter().to_torch_iterable()
loader = tud.DataLoader(it)

for item in loader:
    # ... train step
```

- more speed
- native remote loading
- expressive API

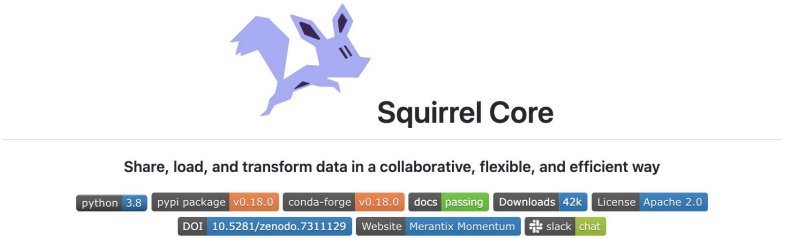
MERANTIX MOMENTUM



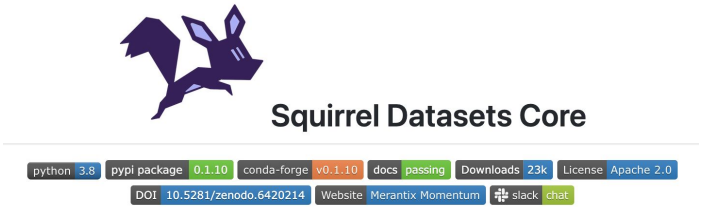
Conclusion

Conclusion

- Squirrel is an open-source data loading tool
- USPs
  - **Speed**: Avoid GPU stall
  - **Cost**: Reduced GPU time and less IO cost through loading in bundles
  - **Flexibility**: Customizable data schemes
  - **Collaboration**: Enables sharing of data and versioning




```
pip install
squirrel-core
```




```
pip install
squirrel-core-datasets
```