FriendliAI

PeriFlow

AutoFlight Large-scale AI Training and Inference

What is PeriFlow?

PeriFlow is a reliable, speedy, and efficient service for training and serving your own large-scale AI models on any data of your choice. It removes the overhead of ML operations and (cloud or on-premise) resource management. With just one click, PeriFlow runs massive-scale AI workloads with our highly-optimized training and serving systems. PeriFlow can run on our self-managed cloud infrastructure or on the infrastructure that you own.

With datasets getting larger and models becoming ever bigger, the time has truly arrived for large-scale ML. Many companies are competitively investing in machines and brainpower to build sophisticated models, with parameter sizes ranging from hundreds of millions to even a few trillion. OpenAI, for example, well-known for GPT-3, the largest neural network ever created at the time, is ambitiously working on its sequel, GPT-4. There are many others as well such as Megatron-NLG, Gopher, OPT, BLOOM, GLaM, PaLM, and BlenderBot3,all of which are Transformer-based language models with a few billion to hundreds of billions of parameters. As such, companies are increasingly making their custom large-scale models or adapting existing models to their needs.

Utilizing such large-scale models, however, is no simple task. There are simply too many factors to take into consideration—developing optimized training and serving systems for your model to minimize costs, setting up resources and environments, managing faults and performance problems, and fine-tuning/adapting your model to particular tasks all become your responsibility.

This is where PeriFlow comes to the rescue. PeriFlow provides an **end-to-end machine learning pipeline for large-scale AI,** from developing models to serving inference requests. It provides an integrated interface for utilizing diverse cloud or on-premise resources and native support for large-scale distributed execution as well. It takes charge of both the training and serving process, all the while automatically resolving faults and performance problems, thereby removing the overhead for any manual monitoring.

On cloud or on premise

Provides an integrated interface that enables seamless utilization of diverse environments

Comprehensive issue handling

Automatically detects and recovers from faults and performance problems

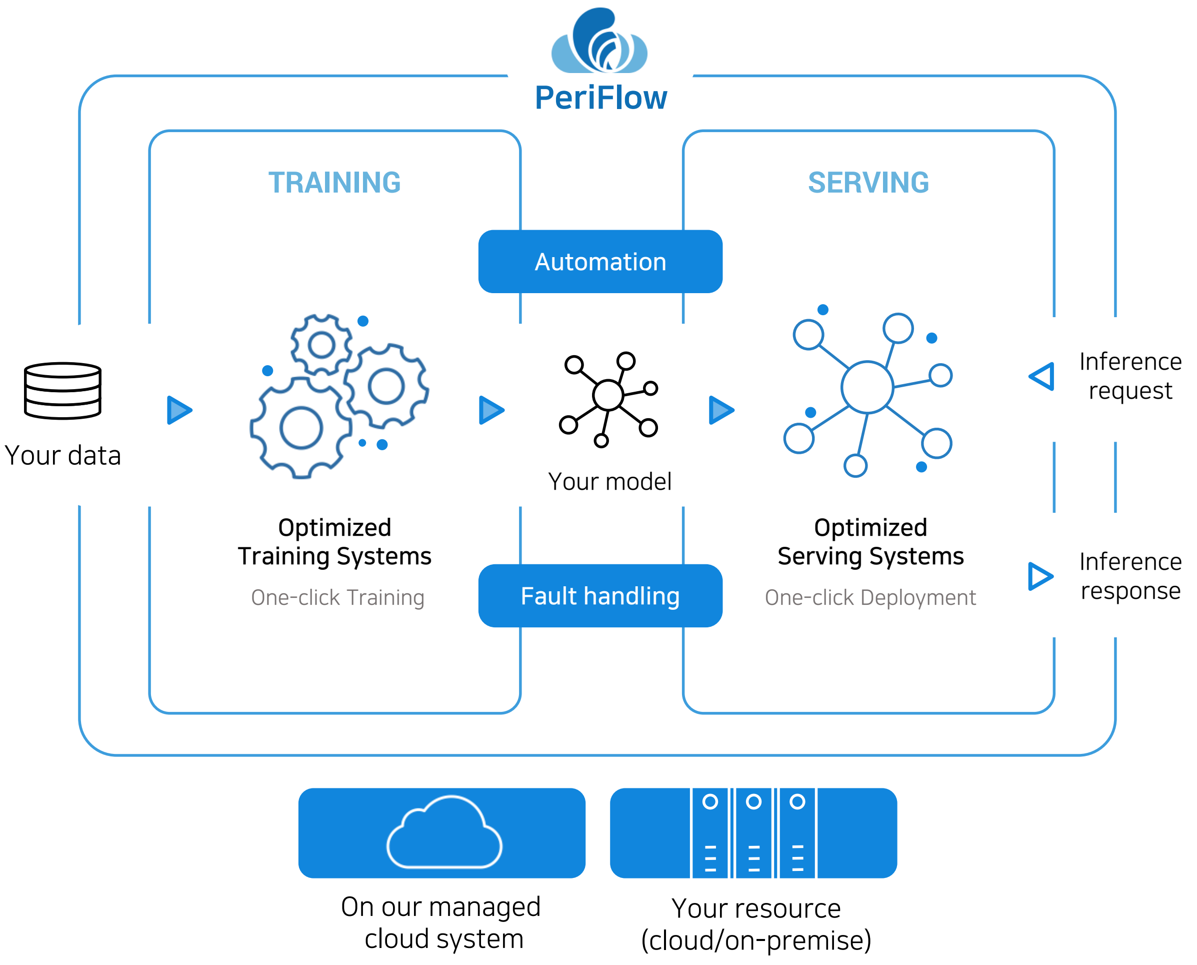
Optimized training and serving systems

Offers state-of-the-art systems from training to serving

PeriFlow Overview

The following diagram shows the overall workflow with PeriFlow.

PeriFlow automatically handles faults and problems that arise during the training/serving process. You can also choose to train/serve your model either on our managed cloud system or on your resources.



Benefits

Training

A. ML operation automation

When running large-scale ML models, setting up resources, moving between different environments and managing the overall operation workflow require painstaking effort. PeriFlow monitors the training procedure and makes all of the above automatic, enabling a smooth experience with large-scale models.

Support for diverse environments. Currently, it is the user or company's responsibility to adapt different cloud services for their specific use cases. However, PeriFlow provides an integrated interface to enable seamless utilization of multiple cloud resources (we manage) such as AWS, Azure, and GCP.

We are also working on releasing a version of PeriFlow that runs on the resources you own. If your company has a self-managed GPU cluster on the cloud, then you would be able to run PeriFlow on the cluster and train your model with it as well. On-premise deployment could be used, too, if you simply wish to do so.

Fault tolerance. When the model of your choice is too large to fit into a single GPU, then it becomes necessary to partition and distribute the parameters across multiple GPUs and machines. Using such distributed execution, even if only a single GPU instance is faulty, the entire training process could be disturbed.

In such cases, PeriFlow repairs the faulty component so that the training process is automatically resumed. Of course, models running on a single GPU still benefit from our fault management as well. Single or distributed, PeriFlow detects faults and resolves them, enabling an automatic and smooth operation.

Performance problem handling. When performing distributed training, a single clumsy node or link can slow down the entire process. Even if that were the case, PeriFlow automatically detects the problem and fixes it, ensuring excellent performance no matter the circumstances.

B. Optimized training systems

With a variety of training systems and techniques being developed every day (e.g., sequence parallelism and selective recomputation introduced in the Megatron-LM), keeping track of the latest technology and determining whether it is effective for your training workload can be quite a demanding job.

PeriFlow provides a state-of-the-art training system with the most up-to-date techniques and makes using them much easier. Among the techniques, our system finds the best combination for your model and cluster. Our team is also actively developing novel, cutting-edge distributed training techniques including but not limited to parallelization strategies, communication optimizations, memory management, and kernel optimizations.

Serving

A. ML operation automation

In PeriFlow, model training and serving can be performed seamlessly. With just a few clicks on a web page or a simple CLI command, you can easily serve your model in PeriFlow. Many different cloud systems are supported, so you can even train and serve your model on a different cloud each time and still get a smooth experience. This can be beneficial when exploiting the different characteristics of different clouds. For example, you could train with A100 GPUs and Infiniband in Azure but serve in some other cloud system depending on your circumstances. Few cloud systems offer Infiniband and Azure is one of them, so coupled with PeriFlow you can have more choices.

B. Automatic management of load and faults

PeriFlow Serving monitors the resources in use and requests (responses) being sent to (sent from) the currently deployed model, allowing you a more stable model serving experience. To optimize the costs, when the number of requests sent to the deployed model increases, PeriFlow automatically assigns more resources to the model, whereas it reduces resource usage when there are not as many requests. Furthermore, if a certain resource gets faulty then PeriFlow proceeds with recovery based on the monitoring results. Our monitoring system also regularly checks for model drift and sends feedback to the user.

C. Optimized serving systems

PeriFlow offers the state-of-the-art serving system Orca for your Transformer models based on our expertise on deep learning systems. Orca consists of both the inference server and engine for serving inference requests—Orca replaces NVIDIA Triton Inference Server and FasterTransformer, just to name a few. Our evaluation showed that Orca outperforms Triton Inference Server with FasterTransformer as its backend engine, by 20X to 80X in terms of throughput (at the same level of latency). Orca is exclusively available in PeriFlow.

Deployment

PeriFlow is provided to customers with a varying degree of support to best suit your needs. PeriFlow Training and Serving can be performed either on our managed cloud system or on your resources. Make use of just our optimized training/serving system, or utilize our fully-managed services alongside the system. In the latter case, automatic, seamless and fault-tolerant management will be ensured.

PeriFlow in Action

Training

Once all the necessary set-up from sign-up to dataset creation is complete, you can train your model with:

```
pf job run
```

For example, provided that you have your training code in `./my_dir` on your local file system and the details of the job are specified in `config.yaml`, to train the model you can run:

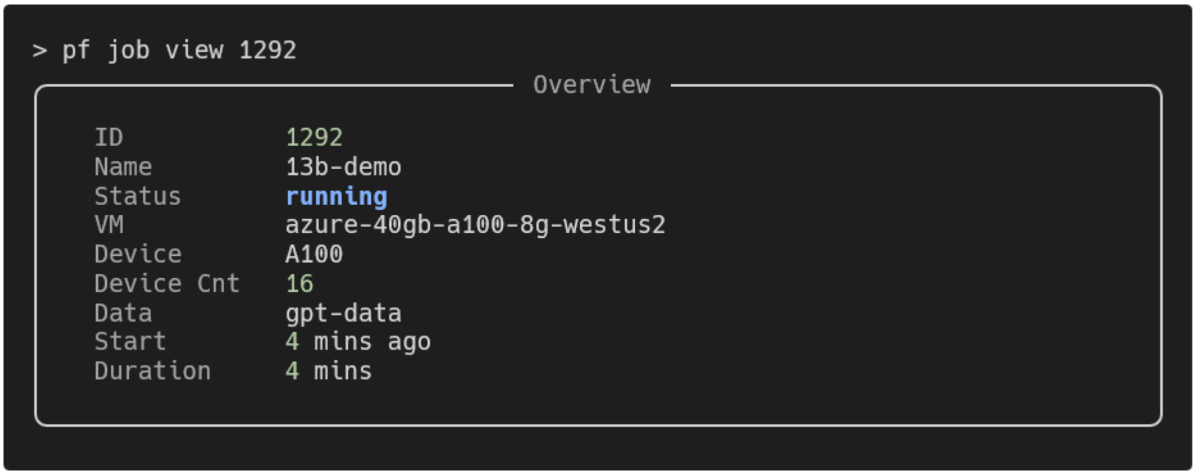
```
pf job run -f config.yaml -w ./my_dir
```

The following code snippet shows an example of `config.yaml`. Here, we train `GPT-13B` using 16 `A100` GPUs.

```
# The name of experiment
experiment: gpt-13b

# The name of job
name: 13b-demo

# The name of vm type
vm: azure-40gb-a100-8g-westus2

# The number of GPU devices
num_devices: 16
```

Once the job has been run successfully, you can check your `JOB_ID` with `pf job list` command. To monitor standard output and errors of the job, use `pf job log JOB_ID` command. Then you can view job details and a list of checkpoints created during training with `pf job view JOB_ID.`

In the following picture, `pf job view 1292` yielded job overview and a list of checkpoints.



```
> pf job view 1292
┌──────────────────────────────── Overview ────────────────────────────────┐
│                                                                          │
│   ID            1292                                                     │
│   Name          13b-demo                                                 │
│   Status        running                                                  │
│   VM            azure-40gb-a100-8g-westus2                               │
│   Device        A100                                                     │
│   Device Cnt    16                                                       │
│   Data          gpt-data                                                 │
│   Start         4 mins ago                                               │
│   Duration      4 mins                                                   │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

(checkpoint list yielded with `pf job view 1292`*)*

```
                          Checkpoints
┌──────────────────┬─────────┬─────────┬───────────┬────────┬────────────┐
│ ID               │ Cloud   │ Region  │ Iteration │ Format │ Created At │
├──────────────────┼─────────┼─────────┼───────────┼────────┼────────────┤
│ 251bbf8f-e3c4-   │ azure-  │ westus2 │ 2000      │ ETC    │ 10s ago    │
│ 4dda-b55e-       │ blob    │         │           │        │            │
│ 654735900756     │         │         │           │        │            │
├──────────────────┼─────────┼─────────┼───────────┼────────┼────────────┤
│ d73ccb07-5241-   │ azure-  │ westus2 │ 1000      │ ETC    │ 2m 12s ago │
│ 42f8-8b82-       │ blob    │         │           │        │            │
│ ba8584a0900c     │         │         │           │        │            │
└──────────────────┴─────────┴─────────┴───────────┴────────┴────────────┘
```

Once the training is done, the `pf checkpoint download CHECKPOINT_ID` command lets you locally download the corresponding checkpoint, which you can deploy right away when serving the model. Again, you can find your `CHECKPOINT_ID` with `pf checkpoint list`.

A more detailed guide on model training could be found at periflow-doc.

Serving

You can deploy your trained model checkpoint with:

```
pf serve create
```

To get a list of jobs being served, type:

```
pf serve list
```

Below is a simple demonstration of the above introduced commands, where the serving job `13B-FAI-AWS (ID:ed13af85)` is being run on a `v100`.

```
> pf serve create -m 13B-FAI-AWS -g v100
Serve (pfs-inference-server-deployment-ed13af85) started successfully. Use
'pf serve view <id>' to see the serve details.
Run 'curl http://serve.friendly.ai/pfs-inference-server-deployment-
ed13af85/v1/completions' for inference request
> pf serve list
                                Serves
┌──────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│ ID       │ Name     │ Status   │ VM       │ Device   │ Device   │ Start    │
│          │          │          │          │          │ Cnt      │          │
├──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ ed13af85 │ 13B-FAI- │ enqueued │ P3.8xlar │ v100     │ 4        │ 0 mins   │
│          │ AWS      │          │ ge       │          │          │ ago      │
└──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘
```

You can view the details of a specific job with its `JOB_ID`:

```
pf serve view JOB_ID
```

```
> pf serve view ed13af85

    ID            ed13af85
    Name          13B-FAI-AWS
    Status        running
    VM            p3.8xlarge
    Device        v100
    Device Cnt    4
    Start         1 mins ago
    Endpoint      http://serve.friendly.ai/pfs-inference-server-deployment-
                  ed13af85/v1/completions
```

Lastly, you can send an inference request to the model with:

```
pf serve request JOB_ID
```

```
> pf serve request ed13af85
Chat with (ed13af85) inference engine.
For exit chat, enter empty line.
> Python is a popular
!
programming language.
And it’s used a lot
for scientific computing.
```

Summary

PeriFlow is the all-in-one solution that you need for using large-scale custom AI on data of your choice. Training and serving large-scale AI models can be technically challenging and costly. With PeriFlow, however, working with large-scale models becomes much friendlier.

First of all, PeriFlow enables smooth workflow when working with diverse environments, allowing seamless transition between your local system and other external environment better suited for larger-scale development. With PeriFlow, switching between different cloud systems and working on-premise becomes much easier as well.

Secondly, PeriFlow provides automatic fault management, performance problem handling and adaptation based on its monitoring results, making the overall workflow continuous and painless.

Last but not least, our optimized training/serving systems guarantee high performance, while dramatically reducing costs.

Experience end-to-end AI workflow from model development to live service on PeriFlow.

FriendliAI

Visit our official website friendli.ai
Contact us via *contact@friendli.ai*

End of Document