

SAI: AI-Enabled Speech Assistant Interface for Science Gateways in HPC

**Pouya Kousha, Arpan Jain, Ayyappa Kolli, Matt Lieber,
Mingzhe Han, Nick Contini, Hari Subramoni, DK Panda**

The Network Based Computing Laboratory

The Ohio State University

<http://mvapich.cse.ohio-state.edu/>



Follow us on

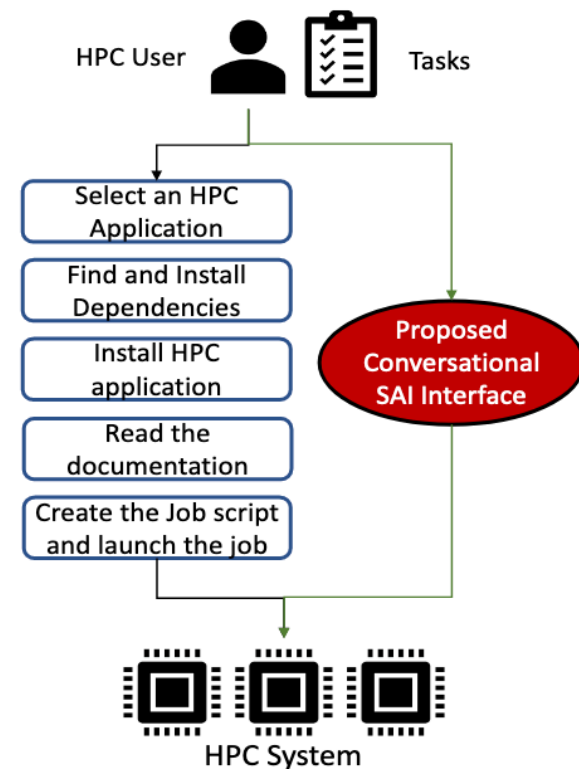
<https://twitter.com/mvapich>

Presentation Outline

- **Introduction and Motivation**
- Problem Statement
- Proposed Designs
- Performance Evaluation
- Discussion
- Concluding Remarks

Introduction

- HPC usage is expanding: New users without HPC background
- Execution of parallel workloads is complex!
 - Learning new interfaces, features, and the terminologies
 - Complex dependencies, installation and execution
- **Steep learning curves for executing tasks and utilizing HPC!**
- **Intuitive Expression:** Users naturally convey needs through words & text
- Emergence of science gateways like Open OnDemand
 - GUI-based interface: more accessible and easier



Presentation Outline

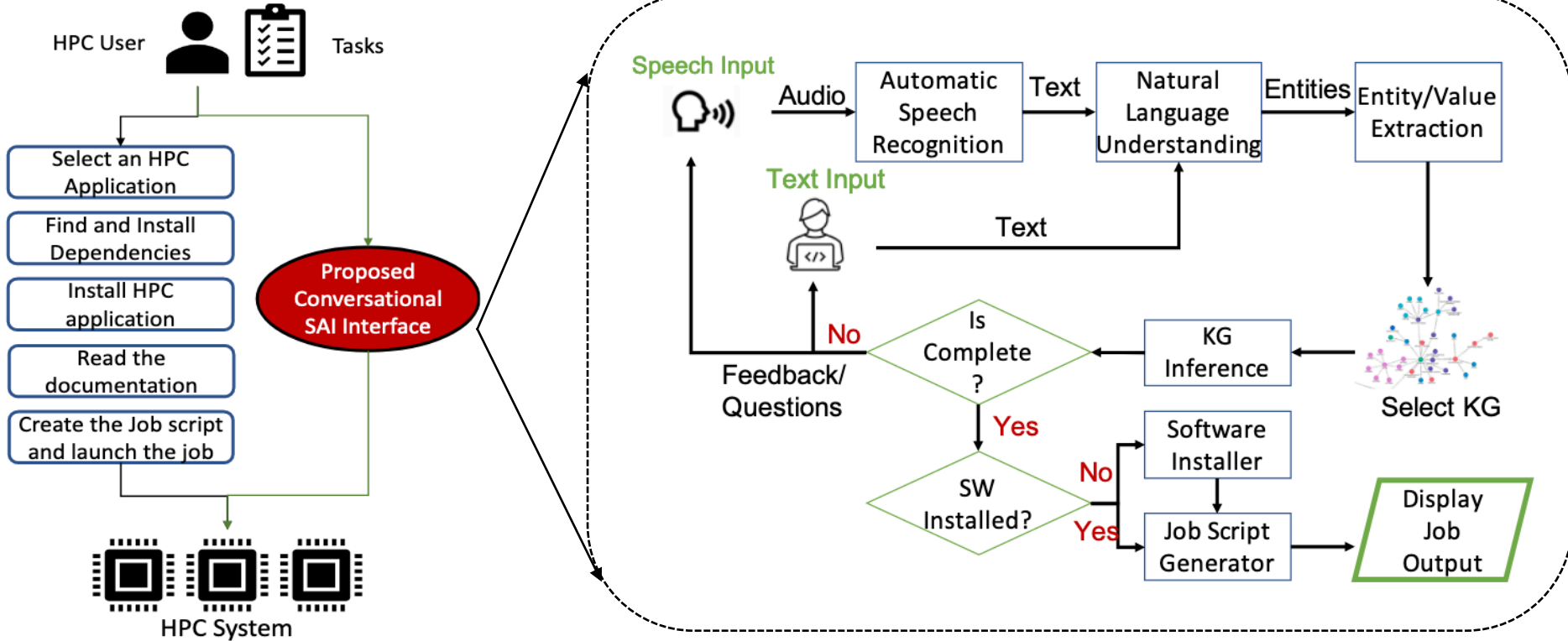
- Introduction and Motivation
- **Problem Statement**
- Proposed Designs
- Performance Evaluation
- Discussion
- Concluding Remarks

Research Challenges

Creating an AI-enabled conversational interface for HPC faces several challenges, including:

- **Technical HPC terms:** ASR and NLU models lack tailored HPC datasets, hindering accurate interpretation of technical terms and abbreviations.
- **Complex relationships:** Mapping complex HPC component relationships can be time-consuming and complex
- **Software installation:** HPC software package installation is a challenge, especially for novice users, even with package managers like Spack
- **Conversational AI integration:** Integrating conversational AI into science gateways requires a modular interface and determining the interface between conversational AI and science gateways

Proposed Framework for Conversational AI for HPC Tasks



Presentation Outline

- Introduction and Motivation
- Problem Statement
- **Proposed Designs**
- Performance Evaluation
- Discussion
- Concluding Remarks

Generating HPC Dataset for Speech and Text

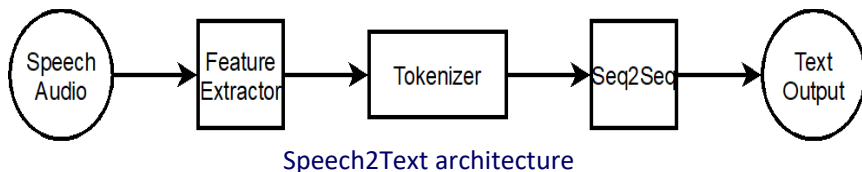
We create an HPC dataset for text and speech containing HPC terminology:

1. Generate basic text queries and label each entity into five broad categories
 - System, software, model, argument, data
 2. Generate combinations of entities with different arguments.
 - Example: the number of combinations for running MPI benchmarks amounts to 315K queries.
 3. Develop synonyms for HPC terminologies to create additional queries.
 - Example: the mentioned MPI-benchmarks query set extends to 19 million queries
 4. Include permutations of phrases to restrict DNN from learning any ordering of arguments
 - Example: MPI-benchmark dataset contains 7 million rows just by including permutations.
- The datasets are called HPC-ASR and HPC-NLU

Training Speech and Text Processing Models for HPC

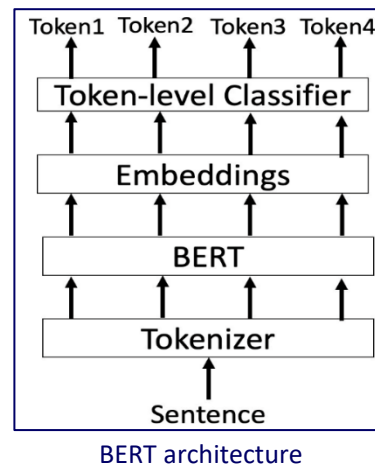
Automatic Speech Recognition

- Model: Transformer based Speech2Text
- Dataset: TIMIT + HPC-ASR [proposed]
- Pretrained on LibriSpeech
- Output: transcribed text



Natural Language Understanding(NLU)

- Model: Bert entity recognition
- Datasets: HPC-NLU [proposed]
- Output: Identify entities and tasks



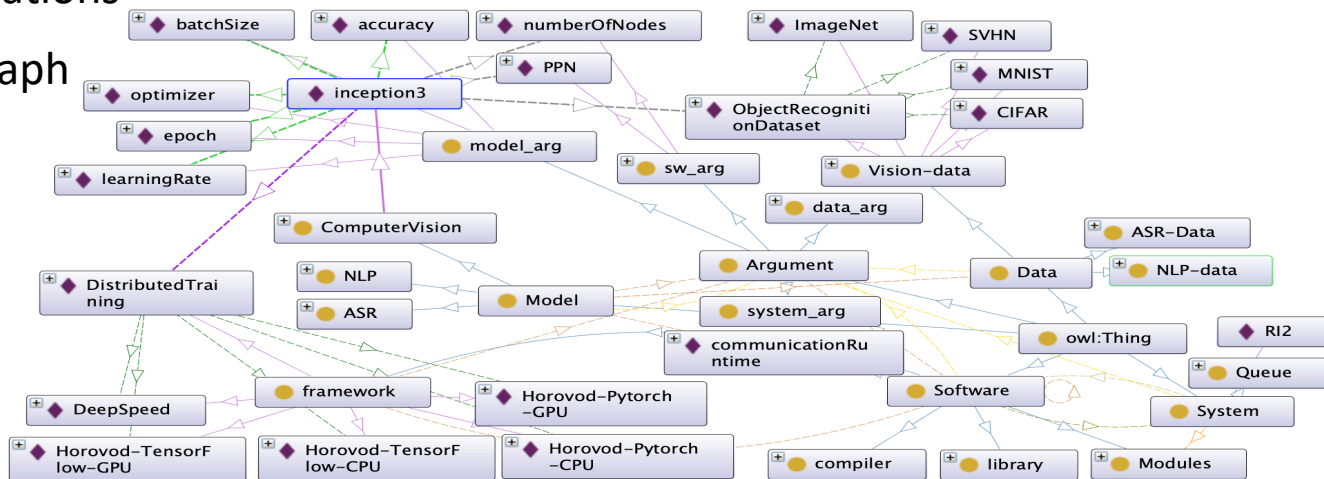
Creating HPC Ontology

- Capture workload relations
- Propose and create SAI-O ontology for capturing complex workload dependencies and relations
- Create Knowledge Graph per application

Table 1. Major object properties in SAI-O ontology

Relation Property	Domain	Range	Description
canBe	any	any	Defines possible values (OR)
	any	Software or Model	Captures run capability
depends	Software	System	Captures software dependency
needs	any	any	Defines requirements (no default)
hasArgs	any	Argument	Defines optional values (defaults)
hasSoftware	any	Software	Captures software availability

Knowledge Graph based on SAI-O ontology for Inception-3 Model



SAI Workflow

Knowledge Graph Query

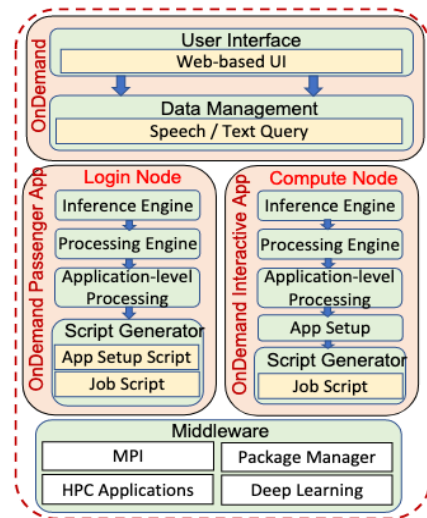
- Given extracted entities
- Queries all KGs
 - Uses SPARQL
 - Max-hit for KG selection
- Gathers arguments
 - Required args
 - Optional args
- Check against user input
- Query user till completion

Software Installer

- Checks & installs dependencies via Spack
- Single Spack config
- Single Spack environment
- Reuse environment if it already exists!
- Asynchronous installation

OnDemand Integration

- HPC Integration and accessibility
- Deployment modes:
- Passenger: shared resources
 - Interactive: exclusive resources



Presentation Outline

- Introduction and Motivation
- Problem Statement
- Proposed Designs
- **Performance Evaluation**
- Discussion
- Concluding Remarks

Evaluation Platform and Methodology

DL Framework: PyTorch is used to define and train DNNs for ASR and NLU.

DNNs: Speech2Text, BERT-based entity detection and classification

Datasets: LibriSpeech, TIMIT, HPC-ASR, and HPC-NLU

Methodology:

1. Compare the performance of pre-trained and fine-tuned Speech2Text models on the HPC test dataset
2. Test the NLU model trained from scratch on predicting entities for given text or speech query
3. Evaluate end-to-end performance of ASR and NLU models
4. Provide overhead evaluation of the SAI pipeline, scaling of SAI as a Passenger App, and performance of SAI as an Interactive App

DL Models Performance Evaluation

ASR Evaluation:

- Improved word error rate for Speech2Text model from 86.2% to 3.7%

Train Dataset	Test Dataset	WER
Base (LibriSpeech)	HPC-ASR	86.2
Base+TIMIT+HPC-ASR	HPC-ASR	3.7

Improved transition of speech to text

NLU Evaluation:

- 60K training size, 5M test size
- Achieved 99% accuracy and precision

Test Dataset	F1-score	Precision	Recall
HPC-NLU (5M)	0.999	0.999	0.999

Predicting entities correctly!

ASR + NLU Evaluation:

- Pipelines ASR+NLU for inference accuracy
- Testing 100 queries from 4 individuals
- M1:** Adjust predicted sentence length to match original
- M2:** Drop less important/incorrect words to match original content

Metric	User 1	User 2	User 3	User 4	Average
WER	10.3	8.6	8.3	4.9	8.03
Accuracy M2	0.97	0.90	0.80	0.95	0.907
Accuracy M1	0.84	0.81	0.83	0.92	0.849

User #1 and #4 are new to SAI and not used in training

High success rate and flexibility recognizing new users' voices!

Overhead of SAI Passenger App Pipeline for Different Queries

End-to-end Overhead: SAI Full Pipeline as Passenger App

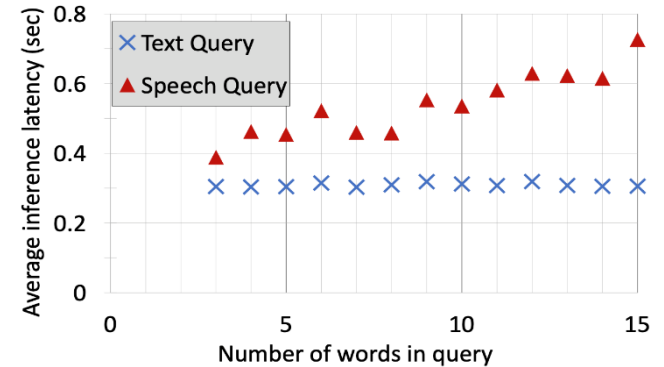
- Evaluate inference latency for various speech/text queries
- Exclude software installation and execution timing
- Speech latency increases with more query words
- Text latency remains constant

End-to-end Overhead of SAI Passenger App with Multiple Users

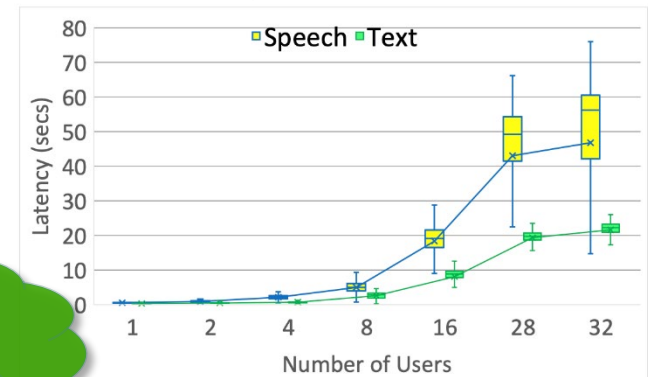
- Higher avg. latency for speech/text with more concurrent users
- Speech queries more affected than text queries

Login node performance degrades significantly with increased passenger mode usage

Deploy interactive app for smoother scaling



SAI passenger evaluation with different queries – avg 200 iterations



SAI passenger end-to-end latency across multiple users – 8 words text/speech and average of 200/100 iterations

Interactive App: Deployment and Portability

Interactive App: Address performance

degradation with **scaling** users

- Exclusive resources
- User-selected architecture

Observations:

- Lower latency on V100 GPU node
- Improved over passenger deployment

Architecture /Model	Deployment type	Total latency	ASR module	NLU module	KG module
BDW speech	Interactive	0.4919	0.23865	0.02275	0.22655
	Passenger	0.50245	0.2366	0.0217	0.2274
BDW text	Interactive	0.2665	N/A	0.0227	0.24335
	Passenger	0.27125	N/A	0.0218	0.24795
SKX speech		0.44085	0.24105	0.0174	0.1754
SKX text		0.22095	N/A	0.0242	0.19585
V100 speech	Interactive	0.40735	0.16585	0.0172	0.224
V100 text		0.2664	N/A	0.0225	0.2433
K80 text		0.2676	N/A	0.0225	0.2448

Compare breakdown & total latency on different architectures, 8-word text/speech query (100 speech/400 text iterations)

Extending SAI Support to New HPC Software

- Two-step process:
 - Create application KG using SAI-O ontology & supported relationships
 - Add application-specific terms to HPC-ASR and dataset
 - NLU is generic to detect new entities!
- SAI provides scripts for ASR model fine-tuning & NLU performance improvement
- Modular design: KG portability across systems, simplified deployment
- Integration with Open OnDemand for easy porting to new system architectures

Presentation Outline

- Introduction and Motivation
- Problem Statement
- Proposed Designs
- Performance Evaluation
- **Discussion**
- Concluding Remarks

Will ChatGPT/Bard suffice?

- General-purpose language model
 - Versatile and it can handle a wide range of language tasks relatively
 - May not excel in any specialized task
- Can be very effective in communicating with humans, but it's far from the best when it comes to interacting with HPC systems
- There is a need for fine-tuned with the specific objective of effective and reliable interaction between both HPC users and HPC systems



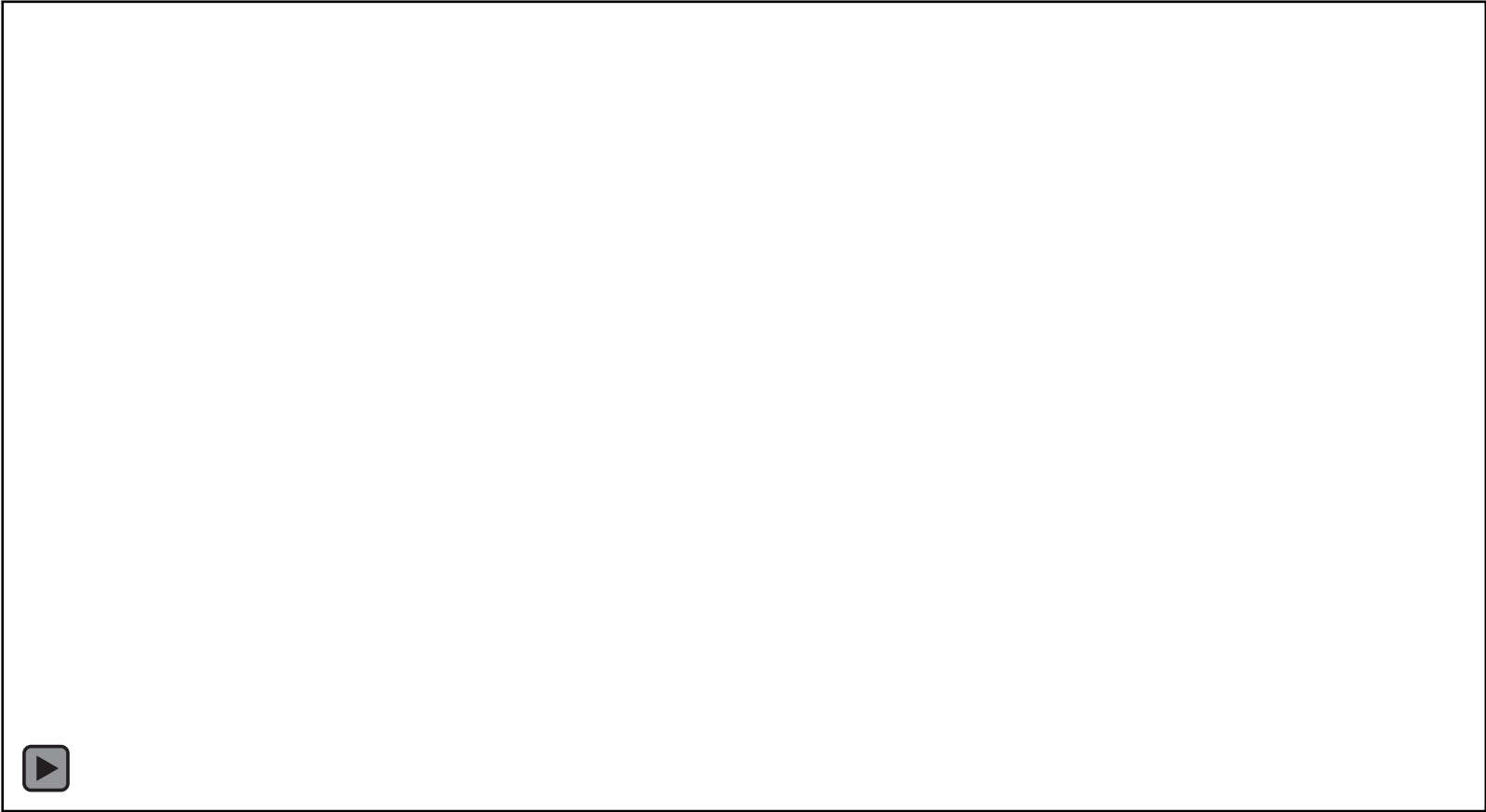
VS



A Swiss army knife falls short if what we need is a multi-head screwdriver!

Comparison with State of the Art: ChatGPT

- **Query** - show me how to benchmark resnet on 2 nodes with 1 PPN



Comparison with State of the Art: ChatGPT (Cont.)

- Qualitative Observations:
 - Sensitivity to user inputs
 - Lack of consistent answers
 - Same question had different answers
 - Can lead to reproducibility issues
- Incorrect answers
 - Data is not distributed among GPUs!
 - Compilation issues

Limitations

- ChatGPT sometimes writes plausible-sounding but incorrect or nonsensical answers. Fixing this issue is challenging, as: (1) during RL training, there's currently no source of truth; (2) training the model to be more cautious causes it to decline questions that it can answer correctly; and (3) supervised training misleads the model because the ideal answer depends on what the model knows, rather than what the human demonstrator knows.
- ChatGPT is sensitive to tweaks to the input phrasing or attempting the same prompt multiple times. For example, given one phrasing of a question, the model can claim to not know the answer, but given a slight rephrase, can answer correctly.

Courtesy: <https://openai.com/blog/chatgpt>

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                       download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
                                          shuffle=True, num_workers=2)

criterion = nn.CrossEntropyLoss()
```

Screenshot of ChatGPT generated code – missing data distribution among processes

Will ChatGPT/Bard suffice?

- ChatGPT produce irrelevant information
 - Provided code and extra parameters
 - Used CIFAR dataset without inquiring user
- Not best practice always!
 - Disabled InfiniBand for NCCL
 - No knowledge of best practices for performance
- Lack of specific HPC system configuration and knowledge

```
bash
#!/bin/bash

#SBATCH --job-name=pytorch_resnet      # Job name
#SBATCH --nodes=2                      # Number of nodes
#SBATCH --ntasks-per-node=2           # Tasks per node
#SBATCH --cpus-per-task=2              # CPUs per task, adjust as needed
#SBATCH --gres=gpu:1                  # Request for 2 GPUs per node
#SBATCH --mem=32G                      # Memory per node, adjust as needed
#SBATCH --time=10:00:00                # Time limit hrs:min:sec, adjust as needed
#SBATCH --output=/path/to/log_%j.log  # Log file with job number

# Initialize PyTorch distributed
source /path/to/your/conda/activate.sh
conda activate your_env

export NCCL_DEBUG=INFO
export NCCL_SOCKET_IFNAME=^docker0,lo
export NCCL_IB_DISABLE=1 # Uncomment this if you are not using Infiniband

# PyTorch distributed
python -m torch.distributed.launch --nproc_per_node=2 --use_env your_pytorch

module load cuda/10.0
module load python/3.8.0
source ~/my_python_env/bin/activate

srun python -m torch.distributed.launch --nproc_per_node=1 --use_env ber
```

Google Bard?

- Task: Requesting same prompt from Bard

- Different answers for the same prompt
- Incorrect answers
- Reproducibility

The generated code is for non-distributed training!

◆ Sure. Here are the steps on how to benchmark ResNet on 2 nodes with 1 PPN

1. Install the necessary dependencies.
2. Clone the ResNet repository.
3. Create a file called `benchmark.py` and add the following code:

```
Python
import tensorflow as tf
import tensorflow_datasets as tfds

def benchmark(batch_size):
    """Benchmarks ResNet on 2 nodes with 1 PPN."""
    dataset = tfds.load("imagenet2012", split="train", shuffle=False)
    images = dataset["image"].batch(batch_size)
    labels = dataset["label"].batch(batch_size)
    with tf.Session() as sess:
        resnet = tf.keras.applications.ResNet50(include_top=False)
        predictions = resnet(images)
        loss = tf.keras.losses.categorical_crossentropy(labels, predictions)
        result = sess.run(loss)
    return result

if __name__ == "__main__":
    for batch_size in [16, 32, 64, 128, 256]:
        print("Batch size: %d" % batch_size)
        result = benchmark(batch_size)
        print(result)
```

Use code with caution. [Learn more](#)

4. Run the `benchmark.py` file.

Wrong Horovod API for init()

show me how to benchmark resnet on 2 nodes with 1 PPN

◆ Sure, here are the steps:

1. Install the necessary dependencies.
 - PyTorch
 - Torchvision
 - Tensorboard
 - Horovod
2. Create a dataset. You can use the ImageNet dataset, which is a large dataset of images that is commonly used for benchmarking machine learning models.
3. Create a ResNet model. You can use the ResNet implementation that is provided in PyTorch.
4. Set up Horovod. Horovod is a distributed training framework that can be used to train machine learning models on multiple GPUs.
5. Run the benchmark. You can use the following command to run the benchmark:

Missing broadcast of model and optimizer parameters using Horovod

```
horovodrun -np 2 python benchmark.py
```

This command will run the benchmark on 2 nodes, each with 1 GPU. The results of the benchmark will be logged to Tensorboard.

Here is an example of a benchmark script that you can use:

```
Python
import torch
import torchvision
import tensorboard
import horovod

def benchmark(model, device):
    inputs = torch.randn(128, 3, 224, 224).to(device)
    outputs = model(inputs)
    return outputs

if __name__ == "__main__":
    torch.manual_seed(0)

    model = torchvision.models.resnet18()
    device = torch.device("cuda:0")

    with horovod.initialize():
        benchmark(model, device)
```

Use code with caution. [Learn more](#)

This script will benchmark the ResNet18 model on a single GPU. To run the benchmark on 2 nodes with 1 GPU each, you would need to change the `device` variable to `cuda:0,cuda:1`.

The results of the benchmark will be logged to Tensorboard. You can view the results by opening Tensorboard and navigating to the `events` directory.

Google Bard with a Descriptive Request

Query: show me how to benchmark resnet50 on 2 nodes with 1 PPN with CIFAR10 dataset using pytorch and horovod

show me how to benchmark resnet50 on 2 nodes with 1 PPN with CIFAR10 dataset using pytorch and horovod

Sure, here are the steps on how to benchmark ResNet50 on 2 nodes with 1 PPN with CIFAR10 dataset using PyTorch and Horovod:

1. Install the necessary dependencies.

```
pip install torch
pip install horovod
```

2. Create a file called `benchmark.py` and add the following code:

```
Python
import torch
import horovod.torch as hvd

def benchmark(model, dataloader):
    losses = []
    for i, (images, labels) in enumerate(dataloader):
        images = images.cuda()
        labels = labels.cuda()

        loss = model(images).mean()
        losses.append(loss.item())

    return losses

if __name__ == "__main__":
    torch.manual_seed(0)
    hvd.init()

    model = torch.nn.ResNet50().cuda()
    dataloader = torch.utils.data.DataLoader(
        CIFAR10(root="", train=True, download=True),
        batch_size=hvd.size(),
        shuffle=True,
    )

    losses = benchmark(model, dataloader)
    print(losses)
```

Use code with caution. [Learn more](#)

3. Run the `benchmark.py` file.

```
python benchmark.py
```

This will benchmark ResNet50 on 2 nodes with 1 PPN with CIFAR10 dataset using PyTorch and Horovod. The output will be a list of losses over 100 epochs.

Here is a breakdown of the code:

- The `import` statements import the necessary dependencies, namely PyTorch and Horovod.
- The `benchmark` function takes a model and a dataloader as input and returns a list of losses.
- The `if __name__ == "__main__":` block is the main execution block. It initializes the random seed, initializes Horovod, creates a ResNet50 model, and creates a dataloader for the CIFAR10 dataset. It then calls the `benchmark` function and prints the output.

I hope this helps! Let me know if you have any other questions.

Mixing Horovod and
pytorch.distributed APIs

No usage of Horovod

show me how to benchmark resnet50 on 2 nodes with 1 PPN with CIFAR10 dataset using pytorch and horovod

Sure, here are the steps on how to benchmark ResNet50 on 2 nodes with 1 PPN with CIFAR10 dataset using PyTorch and Horovod:

1. Install the following dependencies:

- PyTorch
- Horovod
- NVIDIA apex (optional)

2. Create a Python file called `benchmark.py` and add the following code:

```
Python
import torch
import torch.nn as nn
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import horovod.torch as hvd

# Horovod initialization
hvd.init()

# Define the model
model = ResNet50()

# Define the optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

# Load the CIFAR10 dataset
train_loader = torch.utils.data.DataLoader(
    datasets.CIFAR10(
        "data",
        train=True,
        download=True,
        transform=transforms.Compose([transforms.ToTensor(), transforms.Normalize([0.5, 0.5, 0.5], [0.299, 0.299, 0.299])])
    ),
    batch_size=hvd.size(),
    shuffle=True,
)

# Horovod Distributed training
dataloader = hvd.DistributedDataParallel(model)

def main():
    for epoch in range(10):
        for images, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(images)
            loss = nn.functional.cross_entropy(outputs, labels)
            loss.backward()
            optimizer.step()

# Horovod Metrics
print("Horovod metrics:")
print(hvd.all_gather(torch.tensor([hvd.size(), hvd.rank()])))
```

Use code with caution. [Learn more](#)

3. Run the Python file by executing the following command:

```
python benchmark.py
```

Presentation Outline

- Introduction and Motivation
- Problem Statement
- Proposed Designs
- Performance Evaluation
- Demo
- **Concluding Remarks**

Concluding Remarks

- Proposed SAI, a conversational AI-enabled interface for science gateways in HPC, with Automatic Speech Recognition and Entity detection and classification model
 - Created an HPC speech and text dataset, defined a new ontology called SAI-O, and used knowledge graphs to check and validate user tasks, allowing for a general approach for any HPC application
 - Demonstrated capability by supporting three different HPC applications, and integrated SAI in Open OnDemand, deploying it on real HPC systems
 - Evaluated performance and functionality, with positive feedback from early users
- As future work we plan on releasing various components developed
 - HPC-ASR and HPC-NLU datasets
 - The retrained ASR and NLU models
 - Preform user survey

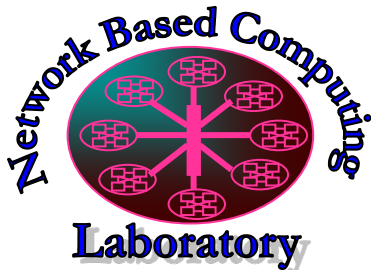


Follow us on

<https://twitter.com/mvapich>

Thank You!

Panda@cse.ohio-state.edu



Network-Based Computing Laboratory

<http://nowlab.cse.ohio-state.edu/>



The High-Performance MPI/PGAS Project

<http://mvapich.cse.ohio-state.edu/>



High-Performance
Big Data

The High-Performance Big Data Project

<http://hibd.cse.ohio-state.edu/>



High-Performance
Deep Learning

The High-Performance Deep Learning Project

<http://hidl.cse.ohio-state.edu/>