"Hey CAI" - Conversational AI Enabled User Interface for HPC Tools

Pouya Kousha(✉), Arpan Jain, Ayyappa Kolli, Prasanna Sainath, Hari Subramoni, Aamir Shafi, and Dhableswar K. Panda

The Ohio State University, Columbus, OH 43210, USA
{kousha.2,jain.575,kolli.38,prasanna.11,shafi.16}@osu.edu,
{subramon,panda}@cse.ohio-state.edu

Abstract. HPC system users depend on profiling and analysis tools to obtain insights into the performance of their applications and tweak them. The complexity of modern HPC systems have necessitated advances in the associated HPC tools making them equally complex with various advanced features and complex user interfaces. While these interfaces are extensive and detailed, they require a steep learning curve even for expert users making them harder to use for novice users. While users are intuitively able to express what they are looking for in words or text (e.g., show me the process transmitting maximum data), they find it hard to quickly adapt to, navigate, and use the interface of advanced HPC tools to obtain desired insights. In this paper, we explore the challenges associated with designing a conversational (speech/text) interface for HPC tools. We use state-of-the-art AI models for speech and text and adapt it for use in the HPC arena by retraining them on a new HPC dataset we create. We demonstrate that our proposed model, retrained with an HPC specific dataset, can deliver higher accuracy than the existing state-of-the-art pre-trained language models. We also create an interface to convert speech/text data to commands for HPC tools and show how users can utilize the proposed interface to gain insights quicker leading to better productivity.

To the best of our knowledge, this is the first effort aimed at designing a conversational interface for HPC tools using state-of-the-art AI techniques to enhance the productivity of novice and advanced users alike.

Keywords: Conversational AI · Performance tools · Speech recognition · Natural language processing

1 Introduction and Motivation

Recently, High-Performance Computing (HPC) has been empowering advances in Artificial Intelligence (AI) and Deep Learning (DL). Popular DL frameworks such as TensorFlow [1] and PyTorch [26] are adopting high-performance messaging libraries for scaling-out workloads on HPC platforms [29]. This trend has resulted in AI practitioners and enthusiasts attempting to utilize HPC software

A.-L. Varbanescu et al. (Eds.): ISC High Performance 2022, LNCS 13289, pp. 87–108, 2022.
https://doi.org/10.1007/978-3-031-07312-0_5

and hardware resources for their applications. It is important for developers of HPC software subsystems to make this transition smoother by enhancing the productivity of HPC tools and libraries for the AI community where the expertise in traditional HPC technologies varies significantly.

One area where new and expert HPC users often struggle, alike, is understanding the performance of their parallel workloads. Analyzing performance bottlenecks for HPC and AI workload is a complicated task. This is, however, critical to improve performance and push boundaries of the state-of-the-art solutions. In this context, the challenge for traditional HPC software, tools, and frameworks is to provide *intuitive* and simple—yet efficient—interfaces to HPC software and hardware resources. *The goal here is to reduce the steep learning curves of HPC tools and libraries.*

There are various tools in HPC for monitoring, analyzing, and characterizing the performance of applications. Profiling tools can be categorized into user-level profiling and system-level profiling based on their usage and provided privileges. For example TAU [22], HPCToolkit [18], and mpiP [2] provide user-level profiling insights while Prometheus [4], TACC STATS [11], and LDMS [9] give us system-level monitoring insights. While HPC tool interfaces are comprehensive and extensive, they require a steep learning curve for learning terminologies and visual interfaces making them very hard to use for novice users with little HPC experience (depicted in Fig. 1). Consider the example of NVIDIA-Nsight [6] or TAU tools that give very detailed insights. Although their interfaces are excellent, navigating and using their interfaces by using keyboard and mouse still requires a lot of learning which includes referring to documentation and going over tutorials and instructional videos - all of which takes time and reduces overall productivity of end users.

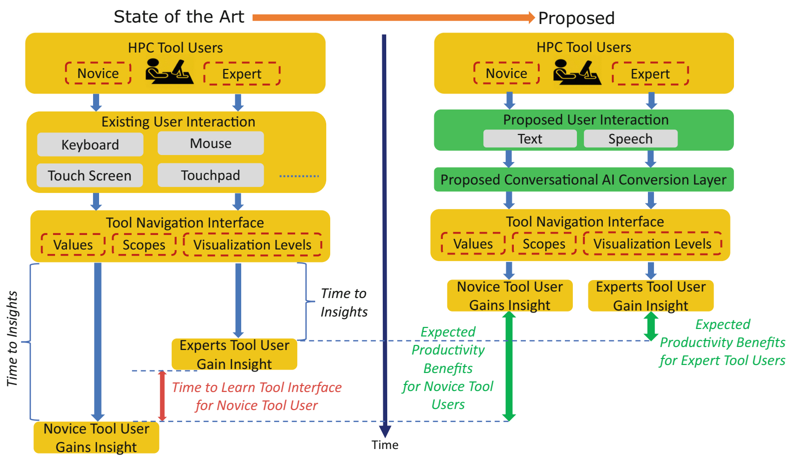
This *steep* learning curve reduces the productivity of expert users while deterring new HPC users to even try these tools that are important for identifying and fixing performance bottlenecks. On the other side, most HPC tool users are intuitively able to express what they are looking for in words or text. Unfortunately, there is no user interface available to HPC tools that can accept such forms of user input.

There are alternative interfaces that the user can utilize. Surveys of end users done by firms like [5] and [7] indicate that users are more likely to use a conversational AI interface as opposed to using older keyboard/mouse style inputs. For example in mobile devices graphical user interface (GUI) exists but, over time the users are more interested to perform daily tasks through Alexa or Siri or similar conversational interfaces. This shows that once the capability is introduced the users are likely to benefit from it as part of future interface expansion as conversational interface is more intuitive. Unfortunately, no interface exists that allows end-users to interact with state-of-the-art HPC tools using speech/text.

1.1 Contributions

In this paper, we take up this challenge and attempt to minimize the learning curve and complexities needed to use state-of-the-art performance profiling tools.

Our proposed solution, titled **Conversational AI Interface (CAI)** exploits Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU)—using DL behind the scenes. CAI has a novel Conversational User Interface (CUI) powered by AI/DL to provide relevant and contextual information to end users. In CUI, while ASR models convert spoken language to text, NLU classifies the text into an intent (the overall objective of the query like network topology) and assigns slots (optional arguments to customize the given intent) thereby allowing CAI to convert the conversational AI input to a format that is understood by the profiling tool. Thus, CAI uses a combination of ASR and NLU to realize a AI-based conversational interface for profiling tools that allows users to interact with these tools using ❶ text and ❷ speech. Both of these novel interfaces provide solutions to increase the productivity of users by reducing the learning curve and hiding the complexity of advanced tool interfaces. Note that the aim is not to replace existing GUI-based interfaces for HPC tools but to supplement t hem and enhance the overall user experience. Further, to demonstrate the feasibility of our approach, we take one HPC profiling tool, OSU INAM [8], and create a conversational AI interface for it as a sample case study. Figure 1 depicts a high-level overview and motivation for CAI. As depicted in Fig. 1, we believe such a solution will result in productivity benefits for novice and expert users alike.



State of the Art
Proposed
HPC Tool Users
Novice
Expert
Existing User Interaction
Keyboard
Mouse
Touch Screen
Touchpad
Tool Navigation Interface
Values
Scopes
Visualization Levels
Time to Insights
Time to Insights
Experts Tool User Gain Insight
Time to Learn Tool Interface for Novice Tool User
Novice Tool User Gains Insight
Time
Proposed User Interaction
Text
Speech
Proposed Conversational AI Conversion Layer
Novice Tool User Gains Insight
Experts Tool User Gain Insight
Expected Productivity Benefits for Novice Tool Users
Expected Productivity Benefits for Expert Tool Users

Fig. 1. Comparison HPC tool usage for state-of-the-art and CAI. Insight#1: On the left side, there is a different overhead in getting performance insight for expert vs novice users but the proposed designs would eliminate this overhead. Insight#2: By using text or speech interface the response time for both users will be lower.

To summarize, the key contributions of this paper are as follows:

1. Proposed an AI-based conversational interface for HPC profiling tools that allows users to interact using speech/text.

2. Designed and created the first speech and text datasets that contain HPC specific terminologies for training ASR and NLU models to be used by the HPC community.
3. Fine-tuned Speech2Text and Wav2Vec ASR models with the proposed HPC dataset to convert a speech command to a text command and trained JointBert and StackPropagation NLU models to understand the context of text command.
4. Improved error rate for Speech2Text DL model from 64% to 2.8% for HPC dataset and 27% to 12% for HPC+TIMIT dataset.
5. Reported 93% accuracy for intent classification and 0.8773 score (F1 score) for slot detection using JointBert DL model for NLU on HPC dataset.
6. Compared the performance of DL inference on client/end user systems (e.g. laptops, desktops) that use HPC profiling tools. Also, implemented a centralized server for inferring on an in-house HPC cluster to reduce the latency for slower client/end user systems.
7. Implemented a simple web-based interface for a sample profiling tool and provided visibility into the intermediate results to better understand the data flow and final output.
8. Deployed and tested CAI and the CAI enhanced tool OSU INAM on a state-of-the-art production HPC system and evaluated the ability of CAI to correctly interpret speech/text input from multiple different volunteer users and display the correct visualization output from the HPC tool.

The rest of the paper is organized as follows: Sect. 2 describes the various challenges we address in this paper. Section 3 provides background on relevant technologies. Section 4 presents our design and implementation for CAI to enable AI-powered conversational user interface for a selected HPC tool. Section 5 evaluates our proposed framework via different performance metrics. Section 6 covers running CAI on client side versus centralized server deployment, trade-offs for speech model selection, explainable flow of CAI, and extending our designs to other HPC tools. Section 7 discusses the related work in the community. Section 8 concludes this paper.

2 Challenges in Exploiting Conversational AI for HPC Tools

We highlight the AI-specific and System-specific challenges associated with creating a conversational AI interface for HPC tools in this section.

AI-Challenge-1 (AI-#1): Creating Text and Speech Dataset with HPC Terminologies/Abbreviations—Each scientific field including HPC has its own terminologies and abbreviations—like Central Processing Unit (CPU), Host Channel Adapter (HCA)—that are typically well-understood in the community. We will refer to these as *HPC jargon* in the rest of the paper. Currently, available language datasets naturally do not provide coverage of HPC jargon. To develop

NLU/ASR DL models, the first step is to create such textual and speech-based datasets. To the best of our knowledge, these kinds of datasets do not exist today.

AI-Challenge-2 (AI-#2): Custom ASR Model for HPC—The performance, in terms of accuracy, of existing off-the-shelf ASR models degrades when the input contains HPC jargons. Figure 2 shows a real example of two sentences, representing a typical interaction of a user with an HPC profiling tool, being transformed to wrong texts by existing ASR models. Our evaluation of state-of-the-art ASR models, presented later in Table 2, depict that the Word Error Rate (WER) for such input data is 64.6 and 77.3. For natural languages, the WER is 2. This clearly motivates that need to retrain and fine-tune existing ASR models for HPC-specific dataset.

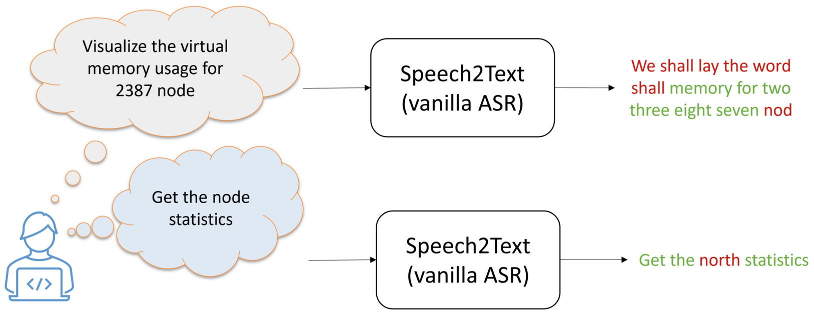

Visualize the virtual memory usage for 2387 node
Speech2Text (vanilla ASR)
We shall lay the word shall memory for two three eight seven nod
Get the node statistics
Speech2Text (vanilla ASR)
Get the north statistics

Fig. 2. Real Output of Automatic Speech Recognition (ASR) by Speech2Text model for two HPC phrases - By using original ASR model, HPC phrases on the left would transform to wrong text (in red) on the right (Color figure online)

AI-Challenge-3 (AI-#3): Custom NLU Model for HPC—Existing off-the-shelf NLU models do not have intents and slots required in HPC tools. In fact, there is no existing model nor dataset for NLU for HPC. This motivates the need to retrain off-the-shelf NLU models with hyper parameter tuning using HPC dataset capable of performing well for this kind of input data.

System-Challenge-1 (Sys-#1): Defining Interface between Conversational AI and HPC Tools—A conversational AI interface to a tool requires a layer that can translate and communicate the result of speech/text input from the user to a format the tool understands. This involves labeling missing arguments for specific user intents and proper mapping of these to the tool performance insight features. User input can have multiple values with different formats and the interface should correctly distinguish them. Considering that HPC tools are written in variety of programming languages and have their own framework, it is challenging to ascertain the communication interface, or standard, between the NLU module and the HPC profiling tool.

System-Challenge-2 (Sys-#2): Integration of Conversational AI to HPC Tools—Another system specific challenge is the integration of NLU+ASR models into existing profiling tools. The conversational interface component needs to be modular in order to accommodate better NLU and ASR models in future without a significant revamp. Also, we plan to evaluate the automation of this integration process for an existing profiling tool. A challenge here is to ascertain and minimize the changes needed to enable the end-to-end pipeline.

3 Background

3.1 Deep Neural Networks Training

Deep neural networks (DNNs) are multi-layer variants of traditional Artificial Neural Networks (ANNs). Each layer in DNN is a collection of basic mathematical functions like weighted summation, called neurons. The forward pass is used to make predictions that are compared with actual output to compute the error. Errors are used to adjust the weights in the backward pass. This process continues till set iterations or till there is a desired loss/convergence. One pass over the entire dataset is known as an epoch, each model requires dozens or even hundreds of epochs to converge.

3.2 Deep Learning Frameworks

Deep learning frameworks are the packages for easy development of the Deep Learning models. They support building and training models for both GPUs and CPUs with built-in libraries for model definition. PyTorch is a well-known open-source Deep Learning framework with define-by-run approach. It provides libraries for defining layers in deep learning models, which developers can use while building their model and it handles the remaining work in training and inferencing of the model.

3.3 OSU INAM

OSU INAM [23] is a HPC network communication profiling, monitoring, and analysis tool designed to provide a holistic online and scalable insight for the understanding of communication traffic on HPC interconnect and GPU through tight integration with MPI runtime, job scheduler, and MPI-based application [19]. INAM runs on one node in the cluster and remotely gathers information from HPC layers in scalable manner [8]. It provides insight and profiling for various HPC users like administrators, software developers, and domain scientists. INAM is capable of gathering and storing performance counters at sub-second granularity for very large clusters ($\approx$2,500 *nodes*). It supports gathering metrics from the PBS and SLURM job schedulers. INAM has been deployed at various HPC clusters and downloaded more than 4,400 times from the project website [23].

4 Design and Implementation

In this Section, we elaborate our design and implementation to enable the conversational interface for HPC profiling tools. Our goal is to remain as modular as possible and integrate the proposed conversational interface for an existing profiling tool (Reference deleted to follow the double-blind policy). Although we choose one HPC tool, the design choices and implementations for NLU and ASR are portable to other tools. However, the interface and integration components require some adjustment to port it to another HPC profiling tool (refer to Sect. 6.4 for more info). Figure 3 shows the high level perspective of our design components that we describe in this section.

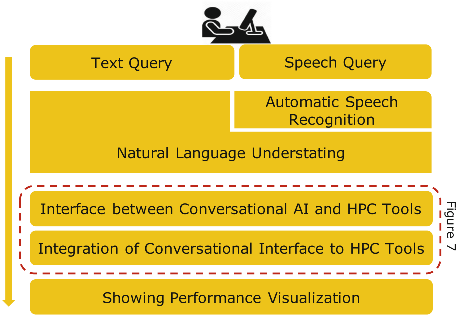

Text Query
Speech Query
Automatic Speech Recognition
Natural Language Understating
Interface between Conversational AI and HPC Tools
Integration of Conversational Interface to HPC Tools
Figure 7
Showing Performance Visualization

Fig. 3. High level design and flow of transforming HPC user query into performance visualization

4.1 Terminologies and Performance Metrics

Terms and legends used in this paper are explained below.

- **ASR**: Automatic Speech Recognition
- **NLU**: Natural Language Understanding
- **Intent**: An intent is high-level goal that the user is trying to accomplish
- **Slot**: Optional arguments that customizes the intent.
- **TIMIT Dataset**: A publicly available speech dataset consisting of 8 major American English dialects.
- **HPC-ASR Dataset**: An in-house ASR dataset created by us for HPC terminologies.
- **HPC-NLU Dataset**: Slots and Intents dataset created by us for training NLU models for HPC profiling tools.
- **Speech Query**: This is an audio that is passed to the ASR and NLU models to generate the visualization. This is spoken by the user. It is one of the ways in which the user can interact with CAI.
- **Text Query**: This is a text that is passed to the NLU model to generate the visualization. This is typed by the user on the web UI. This is the other way in which the user can interact with CAI.
- **WER**: Word Error Rate is the performance metric commonly used to evaluate the ASR models. WER is a metric that works by comparing words in the predicted text and the reference text.
  The formula is as follow:
  $WER = \frac{(S+D+I)}{N}$ where "S" is the number of substitutions, "D" is the number of deletions, "I" is the number of insertions, "C" is the number of correct words, and "N" is the number of words in the reference $(N = S + D + C)$.

- **F1 score**: The performance metrics used to evaluate the NLU models for the slot accuracy and classification accuracy for intents.

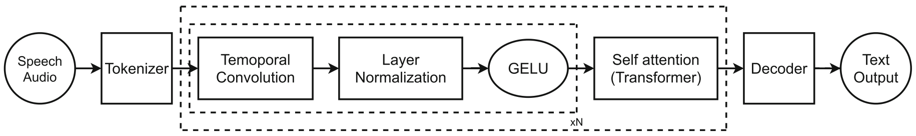
$$F1_{score} = \frac{2 * (precision * recall)}{Precision + Recall} \quad Accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

4.2 Generating HPC Dataset for Speech and Text

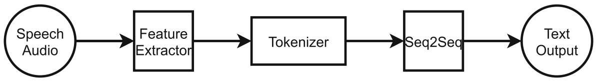
To address AI-#1 (Sect. 2), we create an HPC dataset for text and speech containing HPC terminology. For HPC-dataset, we generated basic queries and labeled their slots and intents. Then, we developed synonyms for HPC terminologies (like CPU, Core, Processor, Central-processor, host-processor for CPU) and English accents are covered by TIMIT. Then we used the synonyms to generate combinations of queries and labeled their slots and intents in human-supervised manner. Both HPC-NLU and HPC-ASR output has been human supervised. The dataset contains four intents, each corresponding to common profiling tool usages: 1) node_usage, 2) net_usage, 3) process_usage, and 4) statistics. The semantic label for each utterance is a dictionary with the intent and a number of slots. An example of a command and its corresponding semantics is shown in Fig. 13 under the slot detected box. The scripts are produced with a few variations of phrases in HPC terminology for each of the intents and recorded from 12 different people with 6 dialects by reading the scripts. The recordings are denoised and verified through human supervision for all of the HPC-ASR database. We labeled the intents and slots for the text in the dataset to create HPC-NLU dataset. We randomly divide the HPC-NLU and HPC-ASR datasets into two subgroups each, one for training (70% of total) and another for testing (30% of total).

4.3 Fine-Tuning Automatic Speech Recognition (ASR) for HPC

To address AI-#2 (Sect. 2) we need a DL model which can understand the audio and transcribe it to a meaningful sentence. We selected and trained two off-the-shelf models - Wav2Vec [12] and Speech2Text [30] where the vanilla (base) models were pre-trained on LibriSpeech ASR corpus, a dataset consisting of approximately 1,000 h of English speech for ASR. The architecture of the models is shown in Fig. 4. We train the models with hyper-parameters tuning on a combination of our in-house HPC-ASR dataset and TIMIT [15]. The TIMIT dataset is used to accommodate different dialects of users and enhance the speech utterance. For Speech2Text and Wav2Vec models, the texts are lower-cased, included with numbers, and tokenized using SentencePiece [20]. By using the HPC dataset create in Sect. 4.2, our models are able to handle complex HPC phrases for understanding HPC user query. Figure 5 shows the same example in Sect. 2 being transformed to the correct text after fine-tuning Speech2Text ASR model on HPC+TIMIT dataset. We tested our models in 4 configurations in Sect. 5.3 of the base vanilla model, the model trained on HPC dataset, the model trained on TIMIT dataset, and the model trained on a combination of HPC and TIMIT datasets.



(a) Wav2Vec model architecture



(b) Speech2Text model architecture

Fig. 4. Architecture of models used for ASR in our proposed design.

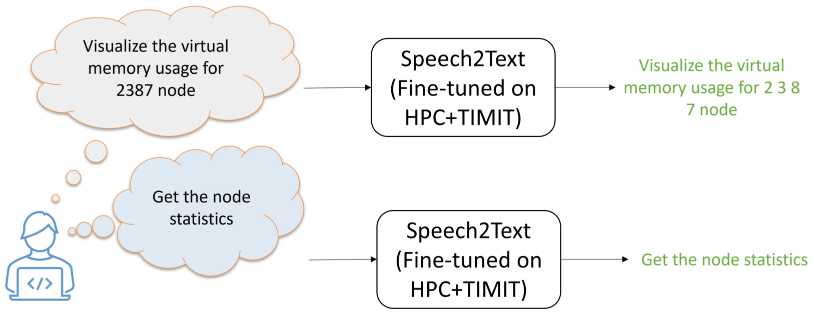
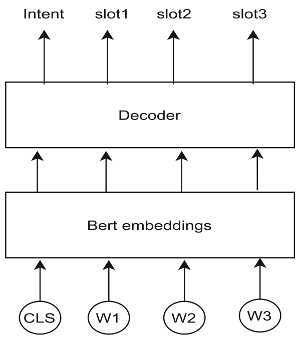

Visualize the virtual memory usage for 2387 node
Speech2Text (Fine-tuned on HPC+TIMIT)
Visualize the virtual memory usage for 2 3 8 7 node
Get the node statistics
Speech2Text (Fine-tuned on HPC+TIMIT)
Get the node statistics

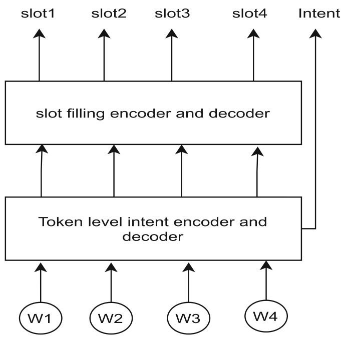
Fig. 5. Real Output of customized Speech2Text model with the two HPC phrases in Fig. 2 where the queries are transformed correctly

4.4 Designing a Natural Language Understanding (NLU) Scheme for HPC Tools

To address AI-#3 (Sect. 2) we train a DL model which can understand and extract the HPC-related information from the output of ASR as text or user's input. The important information to extract from the text is the intention of the user, HPC keywords, and numeric or identity values in the text that have different format. For example, the job number of "453" should not translate as "four five three" or "five hundred and fifty three". We trained two attention-based DL models (StackPropagation and JointBert) on HPC-NLU dataset. Figure 6 shows the architecture of StackPropogation and JointBert models for NLU. These models trained to perform intent detection and slot filling by taking a sentence as a sequence of tokens and assign a label to each token. Based on the tokens the models also detect the intention of the whole sentence. The output of NLU models is a list of Tokens with their assigned labels. This list enables us to extract the required keywords and values, and intent helps in identifying the corresponding visualization in the next modules.



(a) JointBert architecture


slot1
slot2
slot3
slot4
Intent
slot filling encoder and decoder
Token level intent encoder and decoder
W1
W2
W3
W4

(b) StackPropogation architecture

Fig. 6. Models used in CAI to understand the text query and predict intents and slots for HPC profiling tools.

The models use manually labeled HPC terminology to to label the tokens along with the values in the text. We followed prefix format for slot labels, this way phrases that are set of tokens in sequence are understood as representing the same entity (e.g. MPI process counter is labeled as B-process I-process as these two tokens in the sequence represent the same entity). Using the intent, slots labeled, and corresponding values for slots, we identify the task or request in the sentence as discussed in Sect. 4.5.

4.5 Interface Between Conversational AI and HPC Tools

As mentioned in Sys-#1 (Sect. 2), we need to design an interface to map the processed user query to the corresponding HPC tool visualization. NLU handles and labels the existing slots and values for the given input based on the speech/text query. The output of NLU is an intent and a list of label-value pair for each word where the label is slot or utterance as shown in Fig. 13. The interface layer should process the intent and corresponding slots to generate a specific tool-related request to the HPC tool. Figure 7 shows the processing steps for transition from NLU outcome to visualization in green. Each box is a separate modules, implemented as a stand-alone python module shown in Fig. 7. We walk through the interface in the order of the boxes in Fig. 7.

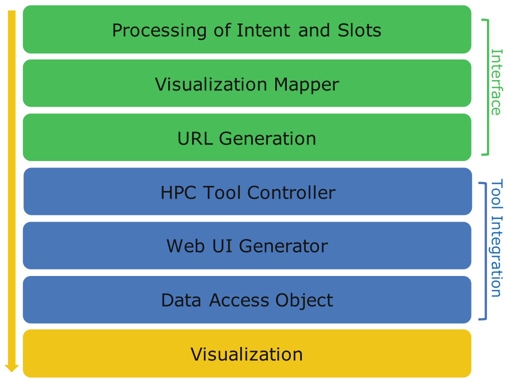


Fig. 7. Detailed modules of interface and interaction for our design

Processing of Intent and Slots: The first module handles three tasks for processing of intents and slots as follows: First, the interface layer handles missing slots and values as there could be missing slots when the user requests a profiling intent. For example, if a user asks for visualization of hardware counters but not specify the metric, considering we have several hardware counters then we assign unicast counters by the default. Second, this module handles incorrect slots and values for different queries and guarantees that from the tools perspective all arguments for decision making exist. Third, this module standardizes the format of different values like time, date, and various HPC measurements. For example, the user can request an insight for the last hour.

Visualization Mapper: This module decides on scope and level of visualization including the chart types and which HPC tool visualization we should select. The visualization level can be cluster, job, node, or process level. The scope could be the time frame that the user is requesting. For making the mapping of request to the corresponding visualizations, the visualization mapper needs input from available visualizations from the HPC profiling tool. Hence, the visualization mapper transforms intents and slots into corresponding performance visualizations. This encapsulates the time to train the user for navigating through different pages and sections of the tool. The alternative design choice would be to make the mapping at the tool level and pass the slots and intents to the tool. Our decision to do the visualization mapping in the interface helps to have fewer changes to the HPC profiler tool as fewer arguments would be passed to the tool by handling the decision at this level.

URL Generation: By this step of CAI the corresponding visualization and values are determined. The next step is to create a connection between the HPC tool and our python-based components. HPC profiling tools and the interface modules have different programming environments. The format and method to communicate between HPC tools and DL components is critical as it imposes the required changes to the HPC tool components to receive and process it. For our paper, the HPC tool supports web access and there are controller in place to handle different web pages and visualizations. Based on this, we decided to exploit this option and generate a Uniform Resource Locator (URL) to interact with the HPC tool. By passing the generated URL to the tool, the tool process the request and direct it to the corresponding web page to show the visualization. All visualization parameters required by the tool are merged into the URL as parameters separated by "&". For example, the partial URL for viewing cluster utilization for historical view of job ID 1456 is "$/network?view = historic\&jobid = 1456$". All the mentioned interface modules run as a python server listening to incoming voice/text requests from HPC tools to respond with appropiate URL.

4.6 Integration of Conversational AI to HPC Tools

In this section, we present our solution to address Sys-#2 that aims for integration of CAI interface into an existing profiling tool. By having a URL as input that gives the arguments and visualization selection for our tool, we aim to integrate The CAI Interface into the HPC profiler tool. Figure 7 shows the processing steps for transition from NLU outcome to visualization in blue.

The modifications to the tool are as follows: 1) The tool needs to record the voice and send it to CAI Interface 2) the tool needs to receive the URL respond and process it. We aim to support Web UI to allow users to benefit from CAI on different platforms and assures accessibility of CAI. For the second task, the flow is as follows: The HPC controller is a Spring Boot controller to redirect the response URLs to corresponding web pages inside the tool. Then, the web UI generator adjusts the values and scopes based on the user parameters extracted from URL for web page initialization. The Data Access Object generates the query to retrieve the profiling data from the time-series database and pass to visualization to plots the visualizations. *The Data Access Object and Visualization have not been changed. The only changes are required for the first two components of the HPC tool controller and Web UI Generator. In the case that the target HPC tool supports web-UI the changes are minimal.*

5 Performance Evaluation

5.1 Evaluation Platform

We conducted our experiments on a 58-node cluster with a combination of nodes of 28 Intel Xeon Broadwell CPU running at 2.40 GHz with NVIDIA Volta V100-32 GB or Skylake CPU running at 2.60 GHz with K80 nodes GPUs. Each node is equipped with a 35 MB L3 cache. The cluster is equipped with MT4119 ConnectX-5 HCAs and Interconnected using SB7790 InfiniBand EDR 100 Gb/s Switches, each having 36 ports.
MPI Library: MVAPICH2 v2.3 [3]
Deep Learning Framework: PyTorch [26] is used to define and train DNNs for ASR and NLU.
Deep Neural Networks: Speech2Text [30], Wav2Vec [12], JointBert [13], and StackPropagation [27].
Datasets: LibriSpeech [25] and TIMIT [15], HPC-ASR Dataset, HPC-NLU Dataset

Table 1. Hardware details of evaluation platform used to conduct the experiments

Architecture	Type	Cores	Speed (GHz)	Label
Broadwell (Server)	CPU	28	2.4	BDW
SkyLake (Server)	CPU	28	2.6	SKX
K80 (Server)	GPU	4992 (Dual socket)	-	K80
V100 (Server)	GPU	CUDA: 5120 Tensor: 640	-	V100
Intel Core i5 8th gen (Surface Pro)	CPU	4	1.8	Client-1
Intel Core i7 11th gen (HP Pavillion)	CPU	4	2.8	Client-2
Intel Core i5 (MacBook Pro)	CPU	4	1.4	Client-3

5.2 Experimental Methodology

In this section, we describe our evaluation methodology used to conduct experiments. In Sect. 5.3, we first individually test the performance of pre-trained vanilla ASR models (Speech2Text and Wav2Vec) on our HPC-ASR dataset and the publicly available TIMIT dataset. Then, we fine-tune ASR models using HPC-ASR and TIMIT training datasets to achieve better WER on the test set. Then we train NLU models (JointBert and StackPropagation) from scratch using our HPC-NLU dataset in Sect. 5.4 to predict the intents and slots for generating appropriate visualizations for the given query. We used two types of validation test, some new queries that did not exist in the training and the other queries are synonym versions of training queries. In Sect. 5.5, we evaluate the performance of both ASR and NLU models to get the end-to-end performance for a speech query. Section 5.6 provides the overhead of deep learning inference for a speech and text query for variable query length on client devices. To improve the performance of deep learning inference for slow client devices, we transfer the inference to a python server running on our in-house cluster with GPU nodes. In Sect. 6.2, we compare the inference time and overall request time for client and centralized python server running on RI2 cluster. In Sect. 6.3, we show the explainability of our proposed conversational UI by providing a detailed flow of information from speech to URL generation.

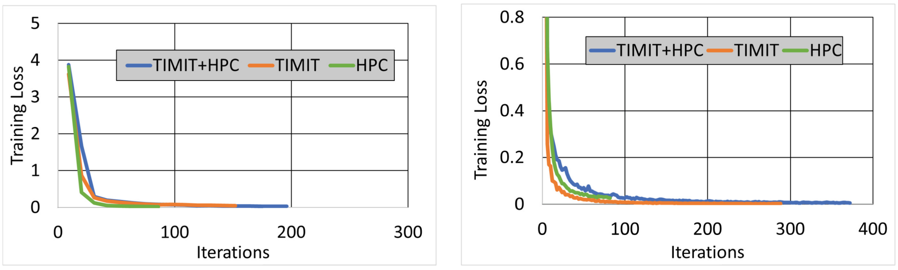
5.3 ASR Results

In these experiments, we evaluate the performance of pre-trained vanilla ASR models (Speech2Text and Wav2Vec) on our HPC-ASR dataset and publicly available TIMIT dataset. As discussed in Sect. 2, the existing ASR models are not suitable for CAI conversational needs as models do not recognize HPC terminologies. Our HPC-ASR dataset has HPC terminologies and the publicly available TIMIT dataset has different accents, which will make our proposed design available to a wide range of speakers. Figure 8(a) and 8(b) show the fine-tuning (training) of Speech2Text and Wav2vec on three combinations of two datasets (training on HPC, TIMIT, and HPC+TIMIT datasets). Final test WER on TIMIT and HPC test set is shown in Table 2.

Table 2. Evaluation of Automatic Speech Recognition (ASR) models using Word Error Rate (WER) - Lower WER is better

Train Dataset		Dataset used for Test			
HPC	TIMIT	Speech2Text		Wav2Vec	
		HPC WER	HPC+Timit WER	HPC WER	HPC+Timit WER
✗	✗	64.613	27.53	67.92	27.16
✗	✓	71.15	33.18	77.38	35.43
✓	✗	2.85	21.8	3.24	65.6
✓	✓	2.92	12.18	3.09	14.24

The first row of the table constitutes the base vanilla models which are publicly available trained versions of speech2text and Wav2Vec (trained on LibriSpeech). The lower WER shows that training on our HPC dataset increases the accuracy of the models to HPC terminologies and combining our training with the TIMIT dataset gives us a better-generalized model when comparing the WER of the same row of the table for different test datasets. As WER depends on the dataset being used, comparing the numbers on the same column shows that using the HPC dataset leads to better (lower) WER. TIMIT dataset has several accents; therefore, we see higher WER for the TIMIT+HPC dataset, but it makes the ASR model more general and applicable to a wide variety of users. Speech2text performs slightly better than Wav2Vec and hence we use speech2text as the default ASR model.



(a) Wav2Vec (b) Speech2Text

Fig. 8. Training loss for ASR models fine-tuned on different combinations of HPC ASR and TIMIT datasets. We show that both models are trained till the improvement in training loss is negligible.

5.4 NLU Results

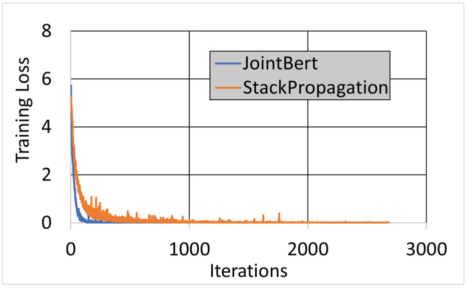


Fig. 9. Training loss of JointBert and StackPropogation models trained on HPC-NLU dataset for NLU. We train models till the improvement in training loss is negligible.

As discussed in Sects. 2 and 4.4, no pre-trained NLU model is available for HPC profiling tools; therefore, we trained NLU models (JointBert and StackPropagation) from scratch using our HPC-NLU dataset. In this section, we evaluate the accuracy of predicting intents and filling slots for our trained NLU models versus human-supervised and labeled HPC-NLU dataset. The output of the model is compared to actual human-supervised HPC-NLU output, that contains synonyms, to calculate accuracy and F1-score. Figure 9 shows the training of JointBert and StackPropagation on HPC-NLU dataset. Table 3 shows the final test accuracy and F1 score for two StackPropagation and JointBert models on the HPC-NLU test set. These two models are trained on the dataset to understand the text and detect the intent and slots in it. We choose JointBert as our defualt model for NLU module as it gives better accuracy for both intents and slots.

Table 3. Evaluation of natural language understanding deep learning models for labeling slots and intents - higher value is better

Model	F1 Score for slots	Intent Accuracy
StackPropagation	0.775	91.79%
JointBert	0.8773	93.36%

5.5 ASR + NLU Analysis

In this experiment, ASR and NLU modules are evaluated together as a pipeline to see if a user provides a speech query how accurately can we detect and assign slots and intents based on our models. Therefore, we use our trained NLU and ASR models to calculate inference accuracy. Table 4 shows the results on end-to-end inference. This shows the results of the chosen NLU model (JointBert) based on the output of our trained ASR models. From Table 4 it can be seen that the F1 score for slots is marginally better when Wav2Vec is used as the ASR model and the intent accuracy is marginally better when speech2text is used as the ASR model. In this work, we use Speech2Text with JointBert to make inference for speech queries. In future, we will use ensemble methods to get better accuracy by training multiple instances of the same model and taking majority decision to allot intent and slots.

Table 4. Evaluation of ASR+NLU pipeline with JointBert as the NLU model. Higher value is better

ASR Model	F1 Score for slots	Intent Accuracy
Speech2Text	0.8295	92.92%
Wav2Vec	0.8349	92.47%

5.6 End-to-End Overhead

In this experiment, we aim to evaluate the overhead of our full pipeline: from user speech/text input in Sect. 5.5) to generating URL and passing it to the tool controller and Web UI generator. Since different visualizations vary in rendering time and it is tool-specific implementation, the numbers do not include the timing for rendering visualizations. Figure 10 showcases the time taken to process speech and text queries of varying lengths on an client device. In general, it can be seen that the time taken to process speech increases with an increase in the number of words in the query. This is expected as the ASR model takes an input of the varying size and hence bigger inputs take more time. The time taken to process a text query is more or less constant as the input size of the NLU model is fixed.

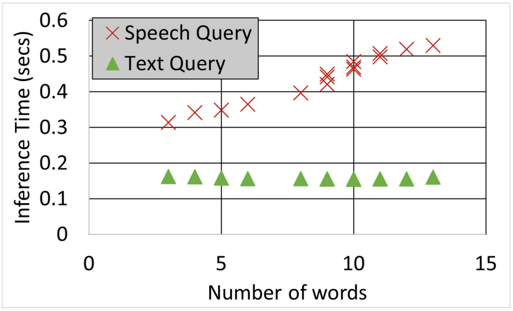


Fig. 10. Inference latency evaluation of ASR+NLU models on client side for 15 different queries consisting of different words

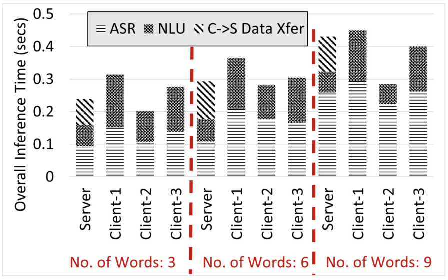
6 Discussion

6.1 Trade-Offs for Converting Speech to Intent

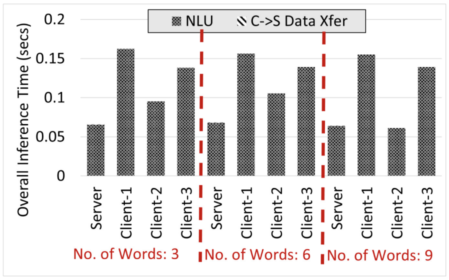
In our design, we use speech-to-text (ASR) followed by text-to-command (NLU) for processing user inputs and mapping them to the tool intents. The alternative approach for speech processing is to directly use speech-to-intent models. In this section, we discuss the trade-offs between the two approaches. We selected ASR + NLU approach since using speech-to-intent model proposes some problems. By having Speech-to-intent model working then any changes require the whole model to train again. In summary, our selection is due to the following reasons. First, speech-to-intent model requires creating a speech and intent dataset for all the tools and HPC applications and map each one to the output intent and slots which will affect modularity and portability of CAI as it limits replacing ASR and NLU models with state-of-the-art models for maintenance. Second, a different text-to-intent-model is required to be trained again for handling text inputs. Third, the speech-to-intent models are still upcoming as we discuss in related work section.

6.2 Comparison of Client-Side vs Server-Side Inference in CAI

In this section, we evaluate two choices of running CAI server on the client or server. If the server is running on client then the inference will be processed on the user computer to get the URL. In the other case, the inference will be done by sending the user's input to a centralized server.



(a) Speech Query



(b) Text Query

Fig. 11. Overall inference time comparing client and server configurations. Note that in Fig. 11(b) the transfer time for text is from client to server is and is thus hard to see. However, it is included

Figure 11 shows the end-to-end inference time versus the number of words in a query for both Speech Query and Text Query on Client devices and a Central Server. The Speech query end-to-end inference time on client device includes time taken by ASR to convert to transcript + time taken by NLU to extract intents and slots. Similarly total inference time for Text query is the time taken by NLU to extract intents and slots. End to end inference time for the server has time taken for transfer from client to the server in addition to that of time taken by client devices for both Text and Speech queries.

Table 5. Inference latency of CAI for user input processing comparing client versus central server for 100 iterations - the client nodes have a "Client" label

Type of device	Speech query (secs)	Text query (secs)
Client-1	1.2914	0.1243
Client-2	0.7409	0.0994
Client-3	0.7949	0.1406
BDW	0.4361	0.0320
SKX	0.4279	0.0291
K-80	0.2825	0.0195
V-100	0.2791	0.0121

Table 5 shows the inference time for speech query and text query on 3 Client devices and Central server nodes like BDW, SKX, K-80, and V-100. Speech query inference time includes time taken by ASR to convert to transcript + time taken by NLU to extract intents and slots+ time taken by python server to generate URL and for Text query, it is the time taken by NLU to extract intents and slots+ time taken by python server to generate URL. The inference time on the server is less compared to that of on client devices. We can see that the shortest inference time is obtained on V100 nodes of the server.

6.3 Insights for Getting Explainable Flow of CAI

It is important to show the clear transition between components from input (speech or text) to the final output (visualization). Having this insight implemented and transparent to the user makes our pipeline process more explainable as results of the solution can be understood by humans, to show which decision was made based on the input provided to each design module and enables understanding the decision making process.

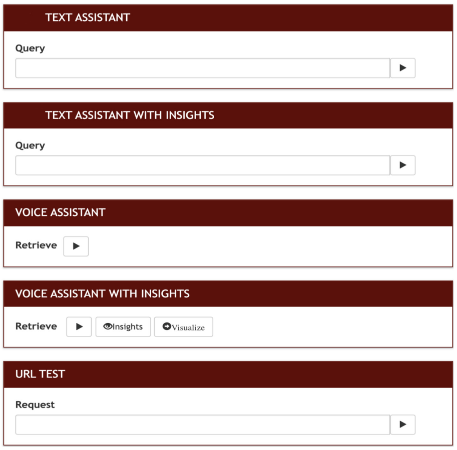
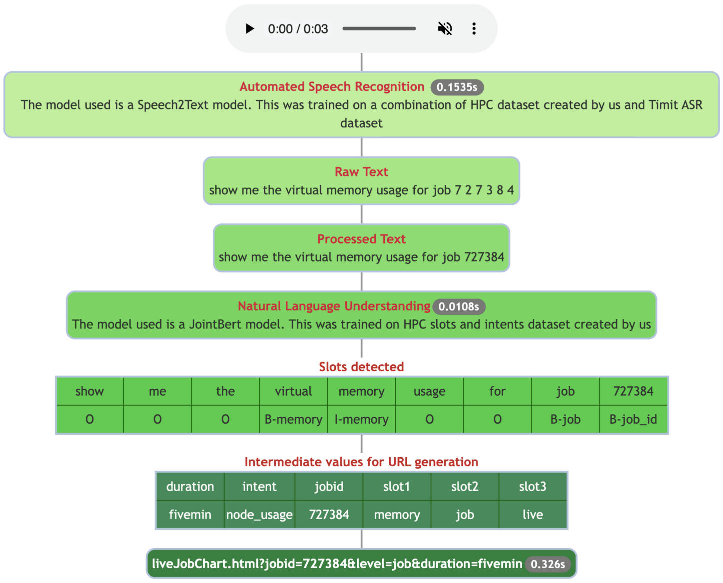


Fig. 12. Screenshot of developed UI for CAI showing various methods of getting the user input in different forms and presenting the option to get flow insight shown in Fig. 13

Figure 12 shows the web UI for the proposed approach where the user can choose to get insights into the flow of CAI by selecting text or voice query box with "insights" then the user gets Fig. 8 that shows the step by step flow with details of each step along with the performance visualization page. We can see that in the figure how CAI converts "show me the virtual memory usage for job 727384" as a voice to outputs for NLU and ASR including the intent and slots and finally make the visualization selection based on intermediate values (dark green). The final URL can be tested in the "URL Test".



Automated Speech Recognition 0.1535s
The model used is a Speech2Text model. This was trained on a combination of HPC dataset created by us and Timit ASR dataset

Raw Text
show me the virtual memory usage for job 7 2 7 3 8 4

Processed Text
show me the virtual memory usage for job 727384

Natural Language Understanding 0.0108s
The model used is a JointBert model. This was trained on HPC slots and intents dataset created by us

Slots detected

show	me	the	virtual	memory	usage	for	job	727384
O	O	O	B-memory	I-memory	O	O	B-job	B-job_id

Intermediate values for URL generation

duration	intent	jobid	slot1	slot2	slot3
fivemin	node_usage	727384	memory	job	live

liveJobChart.html?jobid=727384&level=job&duration=fivemin 0.326s

Fig. 13. Screenshot of the flow of information in proposed approach. The numbers show the latency of ASR and NLU modules and the final number in URL shows end-to-end latency.

6.4 Integrating Other HPC Tools with CAI

In this Section, we describe the changes required to make CAI work with other HPC tools. This can be used as take-away for integrating other HPC tools with CAI. Note that either steps 4 or 5 need modifications to the tool as a tool can be web-based (online) or stand-alone (offline) but not both.

1. If the tool uses tool-specific abbreviations and terminologies then the new samples containing those words need to be added to HPC-ASR and HPC-NLU dataset following step taken in Sect. 4.2 to add more intents and queries. Otherwise, step 2 can be skipped.
2. Use updated dataset to fine-tune ASR Sect. 4.3 and NLU Sect. 4.4 models to recognize new words in the queries.
3. Update interface layer of CAI to accommodate tool-specific variables and visualization types to provide the visualization mapper (Sect. 4.5) knowledge of different visualizations supported by the tool.
4. Web-based tool modifications: Tool needs to implement a unique URL for each visualization chart so that CAI can customize the charts through URL based on user's input.
5. Offline tool modification: The tool needs to parse the URL to get the values and pass them to the existing Data Access Object to fetch and visualize.

7 Related Work

Several studies [21,24,28] exist in literature that use an end-to-end based approach to convert the voice directly to intent and slots, combining ASR and NLU into one model however the trade-off is discussed in Sect. 6.1. Another approach is to combine ASR and NLU models to understand the context of speech sample. Several state-of-the-art ASR models [10,12,30] have been proposed in literature that provide good performance for publicly available dataset and common words found in day-to-day conversation. However, we need to fine-tune these ASR models to recognize technical terms found computer science and HPC. Similarly, NLU models [13,17,27,31,32] are trained for publicly available datasets like Air Travel Information System (ATIS) [16] and SNIPS [14]. Hence, to develop a system for HPC profiling tool, we need to generate our own dataset and retrain models from scratch to get better accuracy. To the best of our knowledge, this is the first work that develops a conversational AI-based interface for HPC profiling tools.

8 Conclusion and Future Work

In this paper, we explored the challenges associated with designing a conversational (speech/text) interface for HPC tools. We used state-of-the-art AI models for speech and text and adapted it for use in the HPC arena by retraining them on new HPC datasets we created. We demonstrated that our proposed model, retrained with an HPC specific dataset, delivers higher accuracy than the existing state-of-the-art pre-trained language models. We also created an interface to convert speech/text data to commands for HPC tools and show how users can utilize the proposed interface to gain insights quicker leading to better productivity. We also deployed and tested CAI and the CAI enhanced OSU INAM on a state-of-the-art production HPC system and evaluated the ability of CAI to correctly interpret speech/text input from multiple different volunteer users and display the correct visualization output from the HPC tool. To the best of our knowledge, this is the first effort aimed at designing a conversational interface using state-of-the-art AI techniques to enhance the productivity of novice and advanced users of HPC tools alike.

As part of future work, we plan on releasing various components developed as part of this paper including 1) the HPC-ASR and HPC-NLU datasets, 2) the retrained ASR and NLU models, 3) CAI, and 4) the enhanced OSU INAM profiling tool with support for CAI. We also plan to extend CAI to other popular profiling tools.

Acknowledgement. This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, #2112606, and XRAC grant #NCR-130002.

References

1. Horovod: Distributed training framework for TensorFlow. https://github.com/uber/horovod
2. MPIP: Lightweight, Scalable MPI Profiling. http://www.llnl.gov/CASC/mpip/
3. MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. http://mvapich.cse.ohio-state.edu/features/. Accessed 13 March 2022
4. Prometheus exporter. https://github.com/prometheus/node_exporter
5. The future of conversational AI (2021). https://www2.deloitte.com/us/en/insights/focus/signals-for-strategists/the-future-of-conversational-ai.html. Accessed 13 March 2022
6. Nvidia Nsight Developer Tools (2022). https://developer.nvidia.com/tools-overview. Accessed 13 March 2022
7. The impact of voice assistants (2022). https://www.pwc.com/us/en/services/consulting/library/consumer-intelligence-series/voice-assistants.html. Accessed 13 March 2022
8. Kousha, P., et al.: Accelerated real-time network monitoring and profiling at scale using OSU INAM. In: Practice and Experience in Advanced Research Computing (PEARC 2020) (2020)
9. Agelastos, A., et al.: The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications, pp. 154–165. SC 2014, IEEE Press, Piscataway, NJ, USA (2014). https://doi.org/10.1109/SC.2014.18, http://dx.doi.org/10.1109/SC.2014.18
10. Amodei, D., et al.: Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. CoRR abs/1512.02595 (2015). http://arxiv.org/abs/1512.02595
11. Barth, B., Evans, T., McCalpin, J.: Tacc stats. https://www.tacc.utexas.edu/research-development/tacc-projects/tacc-stats
12. Baevski, A., Zhou, H., Mohamed, A., Auli, M.: Wav2vec 2.0: a framework for self-supervised learning of speech representations (2020). https://arxiv.org/abs/2006.11477
13. Castellucci, G., Bellomaria, V., Favalli, A., Romagnoli, R.: Multi-lingual intent detection and slot filling in a joint bert-based model. arXiv preprint arXiv:1907.02884 (2019)
14. Coucke, A., et al.: Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. CoRR abs/1805.10190 (2018). http://arxiv.org/abs/1805.10190
15. Garofolo, J., et al.: TIMIT Acoustic-Phonetic Continuous Speech Corpus (1993). 11272.1/AB2/SWVENO, https://hdl.handle.net/11272.1/AB2/SWVENO
16. Hemphill, C.T., Godfrey, J.J., Doddington, G.R.: The ATIS spoken language systems pilot corpus, pp. 96–101. HLT 1990, Association for Computational Linguistics, USA (1990). https://doi.org/10.3115/116580.116613
17. Hosseini-Asl, E., McCann, B., Wu, C., Yavuz, S., Socher, R.: A simple language model for task-oriented dialogue. CoRR abs/2005.00796 (2020). https://arxiv.org/abs/2005.00796
18. HPCToolkit: (2019). http://hpctoolkit.org/. Accessed 13 March 2022
19. Kousha, P., et al.: INAM: Cross-Stack Profiling and Analysis of Communication in MPI-Based Applications. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3437359.3465582

20. Kudo, T., Richardson, J.: Sentencepiece: a simple and language independent subword tokenizer and detokenizer for neural text processing. arXiv preprint arXiv:1808.06226 (2018)
21. Lugosch, L., Ravanelli, M., Ignoto, P., Tomar, V.S., Bengio, Y.: Speech model pre-training for end-to-end spoken language understanding (2019)
22. Malony, A.D., Shende, S.: Performance technology for complex parallel and distributed systems. In: Kotsis, G., Kacsuk, P. (eds.) Proceedings of DAPSYS 2000, pp. 37–46 (2000)
23. OSU InfiniBand Network Analysis and Monitoring Tool. http://mvapich.cse.ohio-state.edu/tools/osu-inam/
24. Palogiannidi, E., Gkinis, I., Mastrapas, G., Mizera, P., Stafylakis, T.: End-to-end architectures for ASR-free spoken language understanding. In: ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 7974–7978 (2020). https://doi.org/10.1109/ICASSP40776.2020.9054314
25. Panayotov, V., Chen, G., Povey, D., Khudanpur, S.: Librispeech: an ASR corpus based on public domain audio books. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5206–5210. IEEE (2015)
26. Paszke, A., et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library (2019)
27. Qin, L., Che, W., Li, Y., Wen, H., Liu, T.: A stack-propagation framework with token-level intent detection for spoken language understanding. arXiv preprint arXiv:1909.02188 (2019)
28. Serdyuk, D., Wang, Y., Fuegen, C., Kumar, A., Liu, B., Bengio, Y.: Towards end-to-end spoken language understanding. CoRR abs/1802.08395 (2018). http://arxiv.org/abs/1802.08395
29. Sergeev, A., Balso, M.D.: Horovod: fast and easy distributed deep learning in TensorFlow. CoRR abs/1802.05799 (2018). http://arxiv.org/abs/1802.05799
30. Wang, C., Tang, Y., Ma, X., Wu, A., Okhonko, D., Pino, J.: Fairseq s2t: fast speech-to-text modeling with fairseq (2020). https://arxiv.org/abs/2010.05171
31. Wen, T., et al.: A network-based end-to-end trainable task-oriented dialogue system. CoRR abs/1604.04562 (2016). http://arxiv.org/abs/1604.04562
32. Wu, D., Ding, L., Lu, F., Xie, J.: Slotrefine: a fast non-autoregressive model for joint intent detection and slot filling. CoRR abs/2010.02693 (2020). https://arxiv.org/abs/2010.02693