Jarvis and NeMo

GTC China
JARVIS
Platform to develop and deploy conversational AI applications
Designed for sensor fusion

Gaze & Speech
https://www.youtube.com/watch?v=r264lBi1nMU
USE CASES

ACROSS ALL VERTICALS

Online Store

- Provide conversational interface for shopping

Industrial

- Collaborative robots - Robots and humans collaborate in close proximity
- Engineer troubleshooting with the help of AI assistant

Finance

- Call center: Sentiment of customers calling
- Insurance chatbot: "Add a wedding ring to an insurance policy via an image and receive policy price quote"

Energy / Oil & Gas

- Use camera and ask, "what are the safety guidelines for this chemical"?
- Loud environment - virtual assistant using lip reading

Consumer Internet

- Video diarization - Meeting/conversation transcription per person with timestamps
- Content tagging with Image, text, Audio - Recommendation, Ads

In car experience

- Autonomous Driving: Enhanced In-car experience combining visual inputs with speech
CHALLENGES OF CONVERSATIONAL AI

- **Custom models**
  - Cloud services not customizable
  - High costs
  - Data Sovereignty

- **Deployment**
  - Existing software not designed for modern production environments

- **Multiple sensors**
  - Difficult to use multiple sensors efficiently

- **High accuracy**
  - Need state-of-the-art algorithms and models

- **Real Time**
  - Requires low latency for natural interaction
JARVIS BENEFITS

Custom models
Start from base model, train with your data on your infrastructure

Deployment
Micro-service approach
Designed for K8s
Simple APIs, easy to integrate

Multiple sensors
Framework for training and deploying models across modalities
Tools to simplify fusion

High accuracy
Best-in-breed algorithms
Direct access to cutting-edge research

Real Time
End-to-end inference on GPUs optimized to reduce latency
JARVIS WORKFLOW OVERVIEW

Pretrained models + Data for customizing → Fine-Tuning

JARVIS AI Services
- Speech Recognition
- Intent Classification
- Speech Synthesis
- Pose estimation
- Lip activity
- Object detection
- Gaze detection
- Wake word

Client Application
- Jarvsi Core (client)

End users → Multiple sensor input

gRPC, Python client library

Jarvis Core (client)

Sensor Fusion, Dialog Manager, Backend fulfillment

Data for customizing → Fine-Tuning
JARVIS WORKFLOW OVERVIEW

Pretrained models + Data for customizing → Fine-Tuning

Speech Recognition, Intent Classification, Speech Synthesis, Pose estimation, Lip activity, Gaze detection, Object detection, Wake word

JARVIS AI Services

gRPC, Python client library

Client Application

Jarvis Core (client)

End users → Multiple sensor input

Sensor Fusion, Dialog Manager, Backend fulfillment (optional)
Visual Diarization

Multiple speaker transcription based on video and audio streams

**Interaction:** Jupyter notebook with live video stream overlaying gaze detection and lip activity detection and producing a text transcript per person from the audio stream

**Technology of sensor fusion:**
- Video stream
  - Gaze detection to engage the system
  - Lip activity to determine who is speaking
- Audio stream:
  - Transcribe the audio
  - Label transcriptions per individual speaker

**Implementation:**
- Fusion graph via JSON to combine the multiple inference models
- gRPC end points for direct interaction with the inference models
- Jupyter notebook demonstrates Python APIs for interaction

**Model Developer:** Improve the conversational model accuracy via fine-tuning with NeMo

**Developer Operations:** Deploy via docker containers from NGC into Kubernetes (EGX)

Transcription
Driver: Where is a good sushi restaurant?
Passenger: What's the weather in Chicago
Jarvis ASR Service

Jarvis ASR TRTIS pipeline

Audio → Feature Extractor (TRTIS custom backend on GPU) → Jasper (TRT on GPU) → End of Sentence Detector (TRTIS custom backend, N-gram language model) → Greedy or Beam Decoder → BERT-based Punctuator (TRT on GPU) → Text

Jarvis ASR API

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognize</td>
<td>Given audio file as input, return transcript</td>
</tr>
<tr>
<td>StreamingRecognize</td>
<td>Process audio from a file or a microphone as it’s being captured, returning partial transcripts</td>
</tr>
</tbody>
</table>
JARVIS - Conversation Bot Overview
Jarvis - Weather Bot Architecture

Deployment of Jarvis components with simple dialog manager
CONVERSATIONAL AI WORKFLOW

Pretrained models + Data for customizing → Fine-Tuning

JARVIS AI Services:
- Speech Recognition
- Intent Classification
- Speech Synthesis
- Pose estimation
- Gaze detection
- Lip activity
- Object detection
- Wake word

Client Application

gRPC, Python client library

Jarvis Core (client)

End users + Multiple sensor input

Client Application:

Sensor Fusion, Dialog Manager, Backend fulfillment

Fine-Tuning

Data for customizing

Jarvis Core (client)

Pretrained models

End users

Multiple sensor input
CONVERSATIONAL AI WORKFLOW

NeMo

Pretrained models + Data for customizing → Fine-Tuning

JARVIS AI Services
- Speech Recognition
- Intent Classification
- Speech Synthesis
- Pose estimation
- Gaze detection
- Lip activity
- Object detection
- Wake word

gRPC, Python client library

Client Application

Jarvis Core (client)

End users + Multiple sensor input

Sensor Fusion, Dialog Manager, Backend fulfillment

Data for customizing

Fine-Tuning
NEMO: TRAINING CONVERSATIONAL AI MODELS

- Open source deep learning Python toolkit for training speech and language models
- High performance training on NVIDIA GPUs
  - Uses TensorCores
  - Multi-GPU
  - Multi-Node
- Based on concept of Neural Module - reusable high level building block for defining deep learning models
- PyTorch backend (TensorFlow on Roadmap)

---

Pretrained Models per module

Neural Modules Collection Libraries

- Voice Recognition
- Natural Language
- Speech Synthesis

Neural Modules Core
- Mixed Precision, Distributed training, Semantic checks

Optimized Framework
- Accelerated Libraries CUDA, cuBLAS, cuDNN etc…
NEMO COLLECTIONS

pip install nemo_asr

nemo_asr
(Speech Recognition)
- Jasper acoustic model
- QuartzNet acoustic model
- RNN with attention
- Transformer-based
- English and Mandarin tokenizers and dataset importers

pip install nemo_nlp

nemo_nlp
(Natural Lang Processing)
- BERT pre-training & finetuning
- GLUE tasks
- Language modeling
- Neural Machine Translation
- Intent classification & slot filling
- ASR spell correction
- Punctuation
- English and Mandarin dataset importers

pip install nemo_tts

nemo_tts
(Speech Synthesis)
- Tacotron 2
- WaveGlow
- English and Mandarin output and datasets importers
NEMO EXAMPLE: JASPER ASR

Audio To Text Data Layer

Audio Preprocessing

Jasper Encoder

Jasper Decoder For CTC

Greedy CTC Decoder

CTC Loss

Logging Callback

Train Action

(invoke)
**NEMO EXAMPLE: JASPER ASR**

```
import nemo
import nemo_asr

data_layer = nemo_asr.AudioToTextDataLayer(...)
data_preprocessor = nemo_asr.AudioPreprocessing()
arpa_encoder = nemo_asr.JasperEncoder(...)


audio, audio_len, text, text_len = data_layer()hist, hist_len = data_preprocessor(audio, audio_len)encoded, encoded_len = jasper_encoder(hist, hist_len)

log_probs = jasper_decoder(encoder_output=encoded)predictions = greedy_decoder(log_probs=log_probs)loss = ctc_loss(log_probs, targets=text, encoded_len, text_len)
```

```
from nemo_asr.helpers import monitor_asr_train_progress
train_callback = nemo.core.SimpleLossLoggerCallback(tensors=[loss, predictions, transcript, transcript_len],print_func=lambda x: monitor_asr_train_progress(x, labels=labels))

neural_factory = nemo.core.NeuralModuleFactory()neural_factory.train(tensors_to_optimize=[loss],callbacks=[train_callback],optimizer="novograd",optimization_params={"lr": 1e-2})
```
# ASR COMPARISONS

**English LibriSpeech dataset %WER**

<table>
<thead>
<tr>
<th>Model</th>
<th>Language Model</th>
<th>Test-Clean</th>
<th>Test-Other</th>
<th>Params, M</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepSpeech 2</td>
<td>5-gram</td>
<td>5.33</td>
<td>13.25</td>
<td>&gt;70</td>
</tr>
<tr>
<td>wav2letter++</td>
<td>ConvLM</td>
<td>3.26</td>
<td>10.47</td>
<td>208</td>
</tr>
<tr>
<td>Listen-Attend-Spell (with SpecAugment)</td>
<td>RNN</td>
<td>2.5</td>
<td>5.8</td>
<td>360</td>
</tr>
<tr>
<td>Jasper 10x5</td>
<td></td>
<td>3.77</td>
<td>11.08</td>
<td>333</td>
</tr>
<tr>
<td></td>
<td>6-gram</td>
<td>3.19</td>
<td>9.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transformer-XL</td>
<td>2.86</td>
<td>8.17</td>
<td></td>
</tr>
<tr>
<td>QuartzNet 15x5</td>
<td></td>
<td>3.90</td>
<td>11.28</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>6-gram</td>
<td>2.96</td>
<td>8.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transformer-XL</td>
<td>2.69</td>
<td>7.25</td>
<td></td>
</tr>
</tbody>
</table>
DOMAIN SPECIFIC ASR

Jupyter notebook transfer learning tutorial

- Start with pretrained base QuartzNet model
- Fine tune with WSJ data (newspaper read aloud)
- Add custom language model to base model
- Add custom language model to fine-tuned model for best performance

- Achieves Word Error Rate of < 2.5!

TRANSFER LEARNING CUSTOMER STORY

S&P Global produces transcriptions of earnings calls - 10,000 hours of high quality data

Scribe application works with ASR models

Recognizes domain specific financial jargon

Additional language models provide meta tags, punctuation

GTC Talk: https://events.rainfocus.com/widget/nvidia/gtcdc19/catalog-short?search=nemo
KENSHO ASR RESULTS

- QuartzNet trained on domain specific financial data outperformed all leading ASR models
- Fine tuning was faster and had higher accuracy than training from scratch

Evaluation WER vs Iteration

- fine-tune
- from-scratch
JARVIS AND NEMO TOGETHER
DEPLOYING JARVIS AI SERVICES TO K8S

One Helm Chart to Deploy

Stage 1: Create TensorRT engines
A. Get PyTorch checkpoints
   - Download from NGC or use local checkpoint
B. Create TRT plans
   - Convert PyTorch checkpoint to TRT
C. Create TRT engines
   - Generate optimized engines

Stage 2: Launch TRTIS
A. Configuring TRTIS
   a. Setup model directory structure and server configuration
B. Launching TRTIS
JARVIS PARTNERSHIP

BENEFITS & EXPECTATIONS

Partnership Benefits

Cutting-edge Technology
- Opportunity to collaborate with NV Eng to accelerate development

Performance
- SW optimizes on GPUs by design
- Prioritization of feature requests

Capability Growth
- Proof-of-concept to solve more challenges

Collaboration

Try and Implement
- API on preferred model/data
- Validation of performance metrics

Help us improve
- Provide Feedback
- File bugs and issues
- Feature requests

Success together
- Your business growth
- Testimonial & Success case study

NVIDIA Confidential, please do not distribute
Try NeMo today!

Register for Jarvis Early Access (January)

https://nvidia.github.io/NeMo/zh/index.html

> pip install nemo_toolkit nemo_asr nemo_nlp nemo_tts

https://developer.nvidia.com/nvidia-jarvis
Backup
Jarvis NLP Service

Jarvis NLP TRTIS pipeline

Text → Tokenizer (TRTIS custom backend on CPU & GPU) → BERT encoder (TRT on GPU) → Sentence classifier (TRT on GPU) → Token classifier (TRT on GPU) → Detokenize (TRTIS custom backend on CPU) → Labels

Jarvis NLP API

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassifyText</td>
<td>Given text input, return a class label and score</td>
</tr>
<tr>
<td>ClassifyTokens</td>
<td>Given text input or array of tokens, return a class label and score per token</td>
</tr>
<tr>
<td>TransformText</td>
<td>Given input text, return output text</td>
</tr>
</tbody>
</table>

Jarvis NLP Provided Models

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalyzeSentiment</td>
<td>Run sentiment detection on input and return label/score</td>
</tr>
<tr>
<td>AnalyzeEntities</td>
<td>Given text input, return named entities (NER)</td>
</tr>
<tr>
<td>Punctuate</td>
<td>Take text without punctuation (e.g. ASR output) and add periods, commas, question marks</td>
</tr>
</tbody>
</table>
Jarvis TTS Service

Jarvis TTS TRTIS pipeline

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesize</td>
<td>Given text input, return audio of spoken version as a single audio clip</td>
</tr>
<tr>
<td>SynthesizeOnline</td>
<td>Given text input, return audio of spoken version as an audio stream</td>
</tr>
</tbody>
</table>