



# **Build a Virtual Assistant Using Jarvis, Cloud Sync, and NeMo**

NetApp Solutions

NetApp  
June 28, 2023

This PDF was generated from [https://docs.netapp.com/us-en/netapp-solutions/ai/cainvidia\\_jarvis\\_deployment.html](https://docs.netapp.com/us-en/netapp-solutions/ai/cainvidia_jarvis_deployment.html) on June 28, 2023. Always check docs.netapp.com for the latest.

# Table of Contents

- Overview ..... 1
  - Jarvis Deployment ..... 1
  - Customize States and Flows for Retail Use Case ..... 1
  - Connect to Third-Party APIs as Fulfillment Engine ..... 9
  - NetApp Retail Assistant Demonstration ..... 9
  - Use NetApp Cloud Sync to Archive Conversation History ..... 10
  - Expand Intent Models Using NeMo Training ..... 12

# Overview

This section provides detail on the implementation of the virtual retail assistant.

[Next: Jarvis Deployment](#)

## Jarvis Deployment

You can sign up for [Jarvis Early Access program](#) to gain access to Jarvis containers on NVIDIA GPU Cloud (NGC). After receiving credentials from NVIDIA, you can deploy Jarvis using the following steps:

1. Sign-on to NGC.
2. Set your organization on NGC: `ea-2-jarvis`.
3. Locate Jarvis EA v0.2 assets: Jarvis containers are in `Private Registry > Organization Containers`.
4. Select Jarvis: navigate to `Model Scripts` and click `Jarvis Quick Start`
5. Verify that all assets are working properly.
6. Find the documentation to build your own applications: PDFs can be found in `Model Scripts > Jarvis Documentation > File Browser`.

[Next: Customize States and Flows for Retail Use Case](#)

## Customize States and Flows for Retail Use Case

You can customize States and Flows of Dialog Manager for your specific use cases. In our retail example, we have the following four yaml files to direct the conversation according to different intents.

See the following list of file names and description of each file:

- `main_flow.yml`: Defines the main conversation flows and states and directs the flow to the other three yaml files when necessary.
- `retail_flow.yml`: Contains states related to retail or points-of-interest questions. The system either provides the information of the nearest store, or the price of a given item.
- `weather_flow.yml`: Contains states related to weather questions. If the location cannot be determined, the system asks a follow up question to clarify.
- `error_flow.yml`: Handles cases where user intents do not fall into the above three yaml files. After displaying an error message, the system re-routes back to accepting user questions. The following sections contain the detailed definitions for these yaml files.

### main\_flow.yml

```
name: JarvisRetail
intent_transitions:
```

```

jarvis_error: error
price_check: retail_price_check
inventory_check: retail_inventory_check
store_location: retail_store_location
weather.weather: weather
weather.temperature: temperature
weather.sunny: sunny
weather.cloudy: cloudy
weather.snow: snow
weather.rainfall: rain
weather.snow_yes_no: snowfall
weather.rainfall_yes_no: rainfall
weather.temperature_yes_no: tempyesno
weather.humidity: humidity
weather.humidity_yes_no: humidity
navigation.startnavigationpoi: retail # Transitions should be context
and slot based. Redirecting for now.
navigation.geteta: retail
navigation.showdirection: retail
navigation.showmappoi: idk_what_you_talkin_about
nomatch.none: idk_what_you_talkin_about
states:
  init:
    type: message_text
    properties:
      text: "Hi, welcome to NARA retail and weather service. How can I
help you?"
    input_intent:
      type: input_context
      properties:
        nlp_type: jarvis
        entities:
          intent: dontcare
# This state is executed if the intent was not understood
dont_get_the_intent:
  type: message_text_random
  properties:
    responses:
      - "Sorry I didn't get that! Please come again."
      - "I beg your pardon! Say that again?"
      - "Are we talking about weather? What would you like to know?"
      - "Sorry I know only about the weather"
      - "You can ask me about the weather, the rainfall, the
temperature, I don't know much more"
    delay: 0
    transitions:

```

```

    next_state: input_intent
idk_what_you_talkin_about:
  type: message_text_random
  properties:
    responses:
      - "Sorry I didn't get that! Please come again."
      - "I beg your pardon! Say that again?"
      - "Are we talking about retail or weather? What would you like to
know?"
      - "Sorry I know only about retail and the weather"
      - "You can ask me about retail information or the weather, the
rainfall, the temperature. I don't know much more."
    delay: 0
  transitions:
    next_state: input_intent
error:
  type: change_context
  properties:
    update_keys:
      intent: 'error'
  transitions:
    flow: error_flow
retail_inventory_check:
  type: change_context
  properties:
    update_keys:
      intent: 'retail_inventory_check'
  transitions:
    flow: retail_flow
retail_price_check:
  type: change_context
  properties:
    update_keys:
      intent: 'check_item_price'
  transitions:
    flow: retail_flow
retail_store_location:
  type: change_context
  properties:
    update_keys:
      intent: 'find_the_store'
  transitions:
    flow: retail_flow
weather:
  type: change_context
  properties:

```

```

        update_keys:
            intent: 'weather'
    transitions:
        flow: weather_flow
temperature:
    type: change_context
    properties:
        update_keys:
            intent: 'temperature'
    transitions:
        flow: weather_flow
rainfall:
    type: change_context
    properties:
        update_keys:
            intent: 'rainfall'
    transitions:
        flow: weather_flow
sunny:
    type: change_context
    properties:
        update_keys:
            intent: 'sunny'
    transitions:
        flow: weather_flow
cloudy:
    type: change_context
    properties:
        update_keys:
            intent: 'cloudy'
    transitions:
        flow: weather_flow
snow:
    type: change_context
    properties:
        update_keys:
            intent: 'snow'
    transitions:
        flow: weather_flow
rain:
    type: change_context
    properties:
        update_keys:
            intent: 'rain'
    transitions:
        flow: weather_flow

```

```

snowfall:
  type: change_context
  properties:
    update_keys:
      intent: 'snowfall'
  transitions:
    flow: weather_flow
tempyesno:
  type: change_context
  properties:
    update_keys:
      intent: 'tempyesno'
  transitions:
    flow: weather_flow
humidity:
  type: change_context
  properties:
    update_keys:
      intent: 'humidity'
  transitions:
    flow: weather_flow
end_state:
  type: reset
  transitions:
    next_state: init

```

## retail\_flow.yml

```

name: retail_flow
states:
  store_location:
    type: conditional_exists
    properties:
      key: '{{location}}'
    transitions:
      exists: retail_state
      notexists: ask_retail_location
  retail_state:
    type: Retail
    properties:
    transitions:
      next_state: output_retail
  output_retail:
    type: message_text
    properties:

```

```

        text: '{{retail_status}}'
    transitions:
        next_state: input_intent
ask_retail_location:
    type: message_text
    properties:
        text: "For which location? I can find the closest store near you."
    transitions:
        next_state: input_retail_location
input_retail_location:
    type: input_user
    properties:
        nlp_type: jarvis
        entities:
            slot: location
            require_match: true
    transitions:
        match: retail_state
        notmatch: check_retail_jarvis_error
output_retail_acknowledge:
    type: message_text_random
    properties:
        responses:
            - 'ok in {{location}}'
            - 'the store in {{location}}'
            - 'I always wanted to shop in {{location}}'
        delay: 0
    transitions:
        next_state: retail_state
output_retail_notlocation:
    type: message_text
    properties:
        text: "I did not understand the location. Can you please repeat?"
    transitions:
        next_state: input_intent
check_retail_jarvis_error:
    type: conditional_exists
    properties:
        key: '{{jarvis_error}}'
    transitions:
        exists: show_retail_jarvis_api_error
        notexists: output_retail_notlocation
show_retail_jarvis_api_error:
    type: message_text
    properties:
        text: "I am having troubled understanding right now. Come again on

```



```
that?"
  transitions:
    next_state: input_intent
```

## weather\_flow.yml

```
name: weather_flow
states:
  check_weather_location:
    type: conditional_exists
    properties:
      key: '{{location}}'
    transitions:
      exists: weather_state
      notexists: ask_weather_location
  weather_state:
    type: Weather
    properties:
    transitions:
      next_state: output_weather
  output_weather:
    type: message_text
    properties:
      text: '{{weather_status}}'
    transitions:
      next_state: input_intent
  ask_weather_location:
    type: message_text
    properties:
      text: "For which location?"
    transitions:
      next_state: input_weather_location
  input_weather_location:
    type: input_user
    properties:
      nlp_type: jarvis
      entities:
        slot: location
      require_match: true
    transitions:
      match: weather_state
      notmatch: check_jarvis_error
  output_weather_acknowledge:
    type: message_text_random
    properties:
```

```

    responses:
      - 'ok in {{location}}'
      - 'the weather in {{location}}'
      - 'I always wanted to go in {{location}}'
    delay: 0
  transitions:
    next_state: weather_state
output_weather_notlocation:
  type: message_text
  properties:
    text: "I did not understand the location, can you please repeat?"
  transitions:
    next_state: input_intent
check_jarvis_error:
  type: conditional_exists
  properties:
    key: '{{jarvis_error}}'
  transitions:
    exists: show_jarvis_api_error
    notexists: output_weather_notlocation
show_jarvis_api_error:
  type: message_text
  properties:
    text: "I am having troubled understanding right now. Come again on
that, else check jarvis services?"
  transitions:
    next_state: input_intent

```

## error\_flow.yml

```
name: error_flow
states:
  error_state:
    type: message_text_random
    properties:
      responses:
        - "Sorry I didn't get that!"
        - "Are we talking about retail or weather? What would you like to know?"
        - "Sorry I know only about retail information or the weather"
        - "You can ask me about retail information or the weather, the rainfall, the temperature. I don't know much more"
        - "Let's talk about retail or the weather!"
      delay: 0
    transitions:
      next_state: input_intent
```

[Next: Connect to Third-Party APIs as Fulfillment Engine](#)

## Connect to Third-Party APIs as Fulfillment Engine

We connected the following third-party APIs as a Fulfillment Engine to answer questions:

- [WeatherStack API](#): returns weather, temperature, rainfall, and snow in a given location.
- [Yelp Fusion API](#): returns the nearest store information in a given location.
- [eBay Python SDK](#): returns the price of a given item.

[Next: NetApp Retail Assistant Demonstration](#)

## NetApp Retail Assistant Demonstration

We recorded a demonstration video of NetApp Retail Assistant (NARA). Click [this link](#) to open the following figure and play the video demonstration.

# NetApp NARA



Hi, welcome to NARA retail and weather service. How can I help you?

Write your message...

Submit

System replied. Waiting for user input.

Unmute System Speech

Next: [Use NetApp Cloud Sync to Archive Conversation History](#)

## Use NetApp Cloud Sync to Archive Conversation History

By dumping conversation history into a CSV file once a day, we can then leverage Cloud Sync to download the log files into local storage. The following figure shows the architecture of having Jarvis deployed on-premises and in public clouds, while using Cloud Sync to send conversation history for NeMo training. Details of NeMo training can be found in the section [Expand Intent Models Using NeMo Training](#).



Next: Expand Intent Models Using NeMo Training

# Expand Intent Models Using NeMo Training

NVIDIA NeMo is a toolkit built by NVIDIA for creating conversational AI applications. This toolkit includes collections of pre-trained modules for ASR, NLP, and TTS, enabling researchers and data scientists to easily compose complex neural network architectures and put more focus on designing their own applications.

As shown in the previous example, NARA can only handle a limited type of question. This is because the pre-trained NLP model only trains on these types of questions. If we want to enable NARA to handle a broader range of questions, we need to retrain it with our own datasets. Thus, here, we demonstrate how we can use NeMo to extend the NLP model to satisfy the requirements. We start by converting the log collected from NARA into the format for NeMo, and then train with the dataset to enhance the NLP model.

## Model

Our goal is to enable NARA to sort the items based on user preferences. For instance, we might ask NARA to suggest the highest-rated sushi restaurant or might want NARA to look up the jeans with the lowest price. To this end, we use the intent detection and slot filling model provided in NeMo as our training model. This model allows NARA to understand the intent of searching preference.

## Data Preparation

To train the model, we collect the dataset for this type of question, and convert it to the NeMo format. Here, we listed the files we use to train the model.

### **dict.intents.csv**

This file lists all the intents we want the NeMo to understand. Here, we have two primary intents and one intent only used to categorize the questions that do not fit into any of the primary intents.

```
price_check
find_the_store
unknown
```

### **dict.slots.csv**

This file lists all the slots we can label on our training questions.

```
B-store.type
B-store.name
B-store.status
B-store.hour.start
B-store.hour.end
B-store.hour.day
B-item.type
B-item.name
B-item.color
B-item.size
```

B-item.quantity  
B-location  
B-cost.high  
B-cost.average  
B-cost.low  
B-time.period\_of\_time  
B-rating.high  
B-rating.average  
B-rating.low  
B-interrogative.location  
B-interrogative.manner  
B-interrogative.time  
B-interrogative.personal  
B-interrogative  
B-verb  
B-article  
I-store.type  
I-store.name  
I-store.status  
I-store.hour.start  
I-store.hour.end  
I-store.hour.day  
I-item.type  
I-item.name  
I-item.color  
I-item.size  
I-item.quantity  
I-location  
I-cost.high  
I-cost.average  
I-cost.low  
I-time.period\_of\_time  
I-rating.high  
I-rating.average  
I-rating.low  
I-interrogative.location  
I-interrogative.manner  
I-interrogative.time  
I-interrogative.personal  
I-interrogative  
I-verb  
I-article  
O

## train.tsv

This is the main training dataset. Each line starts with the question following the intent category listing in the file dict.intent.csv. The label is enumerated starting from zero.

## train\_slots.tsv

```
20 46 24 25 6 32 6
52 52 24 6
23 52 14 40 52 25 6 32 6
...
```

## Train the Model

```
docker pull nvcr.io/nvidia/nemo:v0.10
```

We then use the following command to launch the container. In this command, we limit the container to use a single GPU (GPU ID = 1) since this is a lightweight training exercise. We also map our local workspace /workspace/nemo/ to the folder inside container /nemo/.

```
NV_GPU='1' docker run --runtime=nvidia -it --shm-size=16g \
                    --network=host --ulimit memlock=-1 --ulimit
stack=67108864 \
                    -v /workspace/nemo:/nemo\
                    --rm nvcr.io/nvidia/nemo:v0.10
```

Inside the container, if we want to start from the original pre-trained BERT model, we can use the following command to start the training procedure. data\_dir is the argument to set up the path of the training data. work\_dir allows you to configure where you want to store the checkpoint files.

```
cd examples/nlp/intent_detection_slot_tagging/
python joint_intent_slot_with_bert.py \
    --data_dir /nemo/training_data\
    --work_dir /nemo/log
```

If we have new training datasets and want to improve the previous model, we can use the following command to continue from the point we stopped. checkpoint\_dir takes the path to the previous checkpoints folder.

```
cd examples/nlp/intent_detection_slot_tagging/
python joint_intent_slot_infer.py \
    --data_dir /nemo/training_data \
    --checkpoint_dir /nemo/log/2020-05-04_18-34-20/checkpoints/ \
    --eval_file_prefix test
```



## Inference the Model

We need to validate the performance of the trained model after a certain number of epochs. The following command allows us to test the query one-by-one. For instance, in this command, we want to check if our model can properly identify the intention of the query where can I get the best pasta.

```
cd examples/nlp/intent_detection_slot_tagging/  
python joint_intent_slot_infer_b1.py \  
--checkpoint_dir /nemo/log/2020-05-29_23-50-58/checkpoints/ \  
--query "where can i get the best pasta" \  
--data_dir /nemo/training_data/ \  
--num_epochs=50
```

Then, the following is the output from the inference. In the output, we can see that our trained model can properly predict the intention `find_the_store`, and return the keywords we are interested in. With these keywords, we enable the NARA to search for what users want and do a more precise search.

```
[NeMo I 2020-05-30 00:06:54 actions:728] Evaluating batch 0 out of 1  
[NeMo I 2020-05-30 00:06:55 inference_utils:34] Query: where can i get the  
best pasta  
[NeMo I 2020-05-30 00:06:55 inference_utils:36] Predicted intent:      1  
find_the_store  
[NeMo I 2020-05-30 00:06:55 inference_utils:50] where      B-  
interrogative.location  
[NeMo I 2020-05-30 00:06:55 inference_utils:50] can        O  
[NeMo I 2020-05-30 00:06:55 inference_utils:50] i          O  
[NeMo I 2020-05-30 00:06:55 inference_utils:50] get        B-verb  
[NeMo I 2020-05-30 00:06:55 inference_utils:50] the        B-article  
[NeMo I 2020-05-30 00:06:55 inference_utils:50] best       B-rating.high  
[NeMo I 2020-05-30 00:06:55 inference_utils:50] pasta     B-item.type
```

[Next: Conclusion](#)

## Copyright information

Copyright © 2023 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b) (FEB 2014).

## Trademark information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.