Advanced InstructLab Workshop: Training a custom LLM for the Parasol Insurance Company

Contents

Parasol™ Alex Garcia

Received Claims

Claim 1 | Claim 2 | Claim 3

Subject:

Claim for Recent Car Accident - Policy Number: ABC12345

Body:

Dear XYZ Insurance Company,

I hope this email finds you well. I am writing to report a recent car accident in which I was involved, and I would like to initiate a claim under my policy with your company. My policy number is ABC12345, and my name is John Smith.

Accident Details:

Date and Time: The accident occurred on October 15, 2023, at approximately 2:30 PM.

Location: The accident took place at the intersection of Elm Street and Maple Avenue, near Smith Park, in Springfield, Illinois. The exact coordinates are 39.7476° N, 89.6960° W.

Circumstances:

1. Weather Conditions: On the day of the accident, the weather was overcast with light rain. The road was wet.

2. Traffic Conditions: Traffic was moderate at the time, with vehicles traveling in all directions. I was driving at the posted speed limit of 35 mph.

3. Vehicle Involved: I was driving my Honda Accord at the time of the accident. The other party involved was driving a Ford Escape.

4. Sequence of Events: While I was proceeding through the intersection with a green light, the other vehicle, which was coming from the north, ran a red light and collided with the front passenger side of my vehicle. I had no time to react to avoid the collision.

5. Injuries: Thankfully, there were no serious injuries, but both vehicles sustained significant damage. The police were called to the scene, and a report was filed. The officer's badge number is 12345, and I can provide a copy of the accident report upon request.

6. Witness Information: There were a few witnesses to the accident, and I have their contact information. Their names are Sarah Johnson, Mark Williams, and Lisa Anderson.

7. Photos: I have taken several photos of the accident scene, including the damage to both vehicles, the traffic signals, and road conditions. I can provide these photos to assist with the claim.

8. Other Party's Information: The driver of the other vehicle provided me with their insurance information and contact details, which I can share with you.

Claim Request:

I would like to request that you initiate a claim under my policy for the damages to my vehicle. I would appreciate it if you could provide me with the next steps and the claim process. Please let me know what documentation or information you require from me to process this claim efficiently.

I understand that accidents happen, and I trust in your company's ability to assist me during this challenging time. Your prompt attention to this matter is greatly appreciated.

Please contact me at (555) 123-4567 or john.smith@email.com to discuss this further or to request any additional information you may need.

Thank you for your assistance in this matter, and I look forward to your prompt response.

Sincerely,

John Smith

Attached images:

Advanced InstructLab Workshop: Training a custom LLM for the Parasol Insurance Company

Contents

1. Introduction to the Lab

Thanks for taking the time to learn about and use InstructLab. During this hands-on exercise, you will learn what InstructLab is and how you can use it to provide value to your company. InstructLab is a core component of RHEL AI, and can be used to contribute to and improve a Large Language Model (LLM).

RHEL AI consists of several core components:

1. RHEL Image Mode
2. The Granite Large Language Model(s)
3. InstructLab for model alignment
4. Support and indemnification from Red Hat

InstructLab is a fully open-source project from Red Hat, and the MIT-IBM Watson AI Lab, that introduces Large-scale Alignment for chatBots (LAB). The project's innovation helps during the instruction-tuning phase of LLM training. However, to fully understand the benefit of this project, you need to be familiar with some basic concepts of what an LLM is and the difficulty and cost associated with training a model.

1.1. What is a Large Language Model?

A large language model (LLM) is a type of artificial intelligence (AI) model that uses deep learning techniques to understand and generate human-like text based on input data. These models are designed to analyze vast amounts of text data and learn patterns, relationships, and structures within the data. They can be used for various natural language processing (NLP) tasks, such as:

- **Text classification**: Categorizing text based on its content, such as spam detection or sentiment analysis.
- **Text summarization**: Generating concise summaries of longer texts, such as news articles or research papers.
- **Machine translation**: Translating text from one language to another, such as English to French or German to Chinese.

- **Question answering**: Answering questions based on a given context or set of documents.
- **Text generation**: Creating new text that is coherent, contextually relevant, and grammatically correct, such as writing articles, stories, or even poetry.

Large language models typically have many parameters (millions to billions) that allow them to capture complex linguistic patterns and relationships in the data. They are trained on large datasets, such as books, articles, and websites, using techniques like unsupervised pre-training and supervised fine-tuning. Some popular large language models include GPT-4, Llama, and Mistral.

In summary, a large language model (LLM) is an artificial intelligence model that uses deep learning techniques to understand and generate human-like text based on input data. They are designed to analyze vast amounts of text data and learn patterns, relationships, and structures within the data, and can be used for various natural language processing tasks.

NOTE

To give you an idea of what an LLM can accomplish, the entire previous section was generated with a simple question against the foundational model you are using in this workshop.

1.2. How are Large Language Models trained?

Large language models (LLMs) are typically trained using deep learning techniques and large datasets. The training process involves several steps:

1. **Data Collection**: A vast amount of text data is collected from various sources, such as books, articles, websites, and databases. The data may include different languages, domains, and styles to ensure the model can generalize well.
2. **Pre-processing**: The raw text data is pre-processed to remove noise, inconsistencies, and irrelevant information. This may include tokenization, lowercasing, stemming, lemmatization, and encoding.
3. **Tokenization**: The pre-processed text data is converted into tokens (words or subwords) that can be used as input and output to the model. Some models use byte-pair encoding (BPE) or subword segmentation to create tokens that can handle out-of-vocabulary words and maintain contextual information.
4. **Pre-training**: The model is trained in an unsupervised or self-supervised manner to learn patterns and structures in the data.
5. **Model Alignment**: (instruction tuning and preference tuning): The process of encoding human values and goals into large language models to make them as helpful, safe, and as reliable as possible. This step is not as compute intensive as some of the other steps.

1.3. How does this relate to InstructLab?

InstructLab leverages a taxonomy-guided synthetic data generation process and a multi-phase tuning framework. This allows InstructLab to significantly reduce reliance on expensive human annotations, making contributing to a large language model easy and accessible. This means that InstructLab can use the LLM to generate data, which is then used to further train the LLM. It also means that the alignment phase becomes most users' starting point for contributing their knowledge. Prior to the LAB technique, users typically had no direct involvement in training an LLM. I know this may sound complicated, but hang in there. You will see how easy this is to use.

As you work with InstructLab, you will see the terms Skills and Knowledge. What is the difference between Skills and Knowledge? A simple analogy is to think of a skill as teaching someone how to fish. Knowledge, on the other hand, is knowing that the best place to catch a Bass is when the sun is setting while casting your line near the trunk of a tree along the bank.

2. Getting started with InstructLab

2.1. Install the ilab command line interface

We created a CLI tool called ilab that implements a local LLM developer experience and workflow. The ilab CLI is written in Python and works on the following architectures:

1. Apple M1/M2/M3 Mac
2. Linux systems
3. Windows 11 within a WSL environment

During this lab, we will be using the ilab command line tools on a Red Hat Enterprise Linux system. We anticipate support for more operating systems in the future. The system requirements to use the command line tool are as follows:

1. C++ compiler
2. Python 3.10 or Python 3.11
3. Approximately 60GB disk space (entire process)
    a. Disk space requirements are dependent on several factors. Keep in mind that we will be generating data to feed to the model while also having the model locally on our system. For example, the model we are working with during this workshop is roughly 5gb in size.

2.2. Configuring ilab

The first thing we need to do is to source a Python virtual environment that will allow us to interact with the InstructLab command line tools. While you have two terminal windows available on your right, let's start with the **upper** window.

1. Navigate to the preset InstructLab folder and activate the python virtual environment by running the following commands:

```
cd ~/instructlab
source venv/bin/activate
```

Your prompt should now look like this

```
(venv) [instruct@bastion instructlab]$
```

2. InstructLab is already pre-installed. From your venv environment, verify ilab is installed correctly by running the ilab command.

```
ilab
```

Assuming that everything has been installed correctly, you should see the following output:

```
Usage: ilab [OPTIONS] COMMAND [ARGS]...


  CLI for interacting with InstructLab.


  If this is your first time running ilab, it's best to start with
  to create the environment.


Options:
  --config PATH  Path to a configuration file.  [default: /home/in
  -v, --verbose  Enable debug logging (repeat for even more verbos
  --version      Show the version and exit.
  --help         Show this message and exit.

Commands:
  config    Command Group for Interacting with...
  data      Command Group for Interacting with...
  model     Command Group for Interacting with...
  system    Command group for all system-related...
  taxonomy  Command Group for Interacting with...

Aliases:
```

```
chat      model chat
generate  data generate
serve     model serve
train     model train
```

Congratulations! You now have everything installed and are ready to dive into the world of LLM alignment!

2.3. Initialize ilab

Now that we know that the command-line interface `ilab` is working correctly, the next thing we need to do is initialize the local environment so that we can begin working with the model. This is accomplished by issuing a simple init command. Initialize `ilab` by running the following command:

```
ilab config init
```

You should see the following output (press `ENTER` for defaults):

```
Welcome to InstructLab CLI. This guide will help you to setup your env
Please provide the following values to initiate the environment [press
Path to taxonomy repo [/home/instruct/.local/share/instructlab/taxonom
```

NOTE

You may hit `ENTER` for all default settings.

```
Path to your model [/home/instruct/.cache/instructlab/models/merlinite
Generating
 /home/instruct/.config/instructlab/config.yaml
...
Detecting Hardware...
We chose Nvidia 1x L4 as your designated training profile. This is for
This profile is the best approximation for your system based off of th
Is this profile correct? [Y/n]: Y
```

Type `Y` as shown above or press `ENTER` to accept the training profile configuration. **For this lab**, we are using a single NVIDIA L4 GPU as described in the above output.

```
Initialization completed successfully, you're ready to start using
 ilab
. Enjoy!
```

- Several things happen during the initialization phase: A default taxonomy is located on the local file system, and a configuration file (config.yaml) is created in the 'home/instruct/.config/instructlab/' directory.
- The config.yaml file contains defaults we will use during this workshop. After this workshop, when you begin playing around with InstructLab, it is important to understand the contents of the configuration file so that you can tune the parameters to your liking.

2.4. Download the model

With the InstructLab environment configured, you will now download two different quantized (compressed and optimized) models to your local directory. Granite will be used as a model server for API requests, and Merlinite will help create synthetic data to train a new model.

NOTE

We are using quantized models because we are only leveraging a single GPU for this lab. For better performance or production use cases, you would use unquantized models.

Run the `ilab model download` commands as shown below:

First, let's download Granite:

```
ilab model download --repository instructlab/granite-7b-lab-GGUF --
```

One more time, let's pull down Merlinite:

```
ilab model download --repository instructlab/merlinite-7b-lab-GGUF
```

The `ilab model download` commands download models from the HuggingFace Instructlab organization that we will use for this workshop. The output should look like the following:

```
Downloading model from Hugging Face: instructlab/granite-7b-lab-GGUF@m
Downloading 'granite-7b-lab-Q4_K_M.gguf' to '/home/instruct/.cache/ins
INFO 2024-09-10 16:51:32,740 huggingface_hub.file_download:1908: Downl
granite-7b-lab-Q4_K_M.gguf: 100%|| 4.08G/4.08G [00:19<00:00, 207
Download complete. Moving file to /home/instruct/.cache/instructlab/mo
INFO 2024-09-10 16:51:52,562 huggingface_hub.file_download:1924: Downl
```

Now the models are downloaded, we can serve and chat with the Granite model. Serving the model simply means we are going to run a server that will allow other programs to interact with the data similar to making an API call.

2.5. Serving the model

Let's serve the model by running the following command:

```
ilab model serve --model-path /home/instruct/.cache/instructlab/mo
```

As you can see, the serve command can take an optional `--model-path` argument. In this case, we want to serve the Granite model. If no model path is provided, the default value from the config.yaml file will be used.

Once the model is served and ready, you'll see the following output:

```
INFO 2024-09-10 18:12:09,459 instructlab.model.serve:145: Using model
INFO 2024-09-10 18:12:09,459 instructlab.model.serve:149: Serving mode
INFO 2024-09-10 18:12:16,023 instructlab.model.backends.llama_cpp:250:
 {% for message in messages %}
{% if message['role'] == 'user' %}
{{ '◁user▷
' + message['content'] }}
{% elif message['role'] == 'system' %}
{{ '◁system▷
' + message['content'] }}
{% elif message['role'] == 'assistant' %}
{{ '◁assistant▷
' + message['content'] + eos_token }}
{% endif %}
{% if loop.last and add_generation_prompt %}
{{ '◁assistant▷' }}
{% endif %}
{% endfor %}
INFO 2024-09-10 18:12:16,026 instructlab.model.backends.llama_cpp:193:
INFO 2024-09-10 18:12:16,026 instructlab.model.backends.llama_cpp:194:
```

WOOHOO! You just served the model for the first time and are ready to test out your work so far by interacting with the LLM. We are going to accomplish this by chatting with the model.

2.6. Chat with the model

Because you're serving the model in one terminal window, you will have to use a separate terminal window and re-activate your Python virtual environment to run the `ilab chat` command and communicate with the model you are serving.

1. In the **bottom** terminal window, issue the following commands:

```
cd ~/instructlab
source venv/bin/activate
```

Your prompt should now look like this

```
(venv) [instruct@bastion instructlab]$
```

2. Now that the environment is sourced, you can begin a chat session with the ilab chat command:

```
ilab model chat -m /home/instruct/.cache/instructlab/models/granite
```

You should see a chat prompt like the example below.

```
╭──────────────────────────────────────────────────────────────────────────────
│ Welcome to InstructLab Chat w/ GRANITE-7B-LAB-Q4_K_M.GGUF (type /h f
>>>
```

3. At this point, you can interact with the model by asking it a question. Example: What is OpenShift in 20 words or less?

```
What is OpenShift in 20 words or less?
```

Wait, wut? That was AWESOME!!!!! You now have your own local LLM running on this machine. That was pretty easy, huh?

3. Integrating AI into an Insurance Application

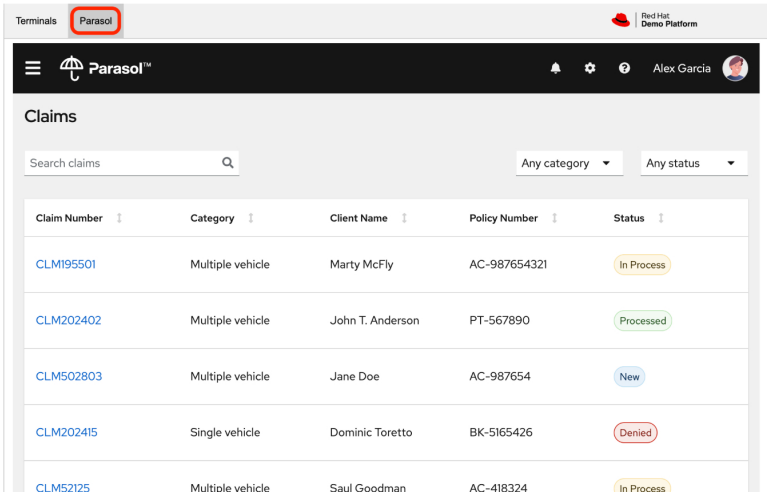
The previous section showed you the basics of how to interact with InstructLab. Now let's take things a step further by using InstructLab with an example application. We will use InstructLab to leverage the Granite LLM, add additional data in the form of knowledge and/or skills, train the model with new knowledge and enable it to answer questions effectively. This is done in the context of Parasol, a fictional company that processes insurance claims.

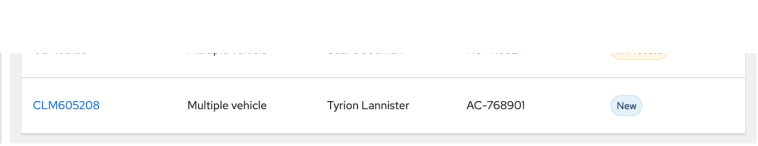
Parasol has a chatbot application infused with AI (the Granite model) to provide repair suggestions for claims submitted. This would allow Parasol to expedite processing of various claims on hold. But at the moment, the chatbot does not provide effective repair suggestions. Using historical claims data that contain different repairs performed under different conditions, we show how users can add this knowledge to the Granite model, train it on the additional knowledge and improve its recommendations.

3.1. Using the Parasol Application

Let's start by taking a look at the current experience a claims agent has when interacting with the chatbot.

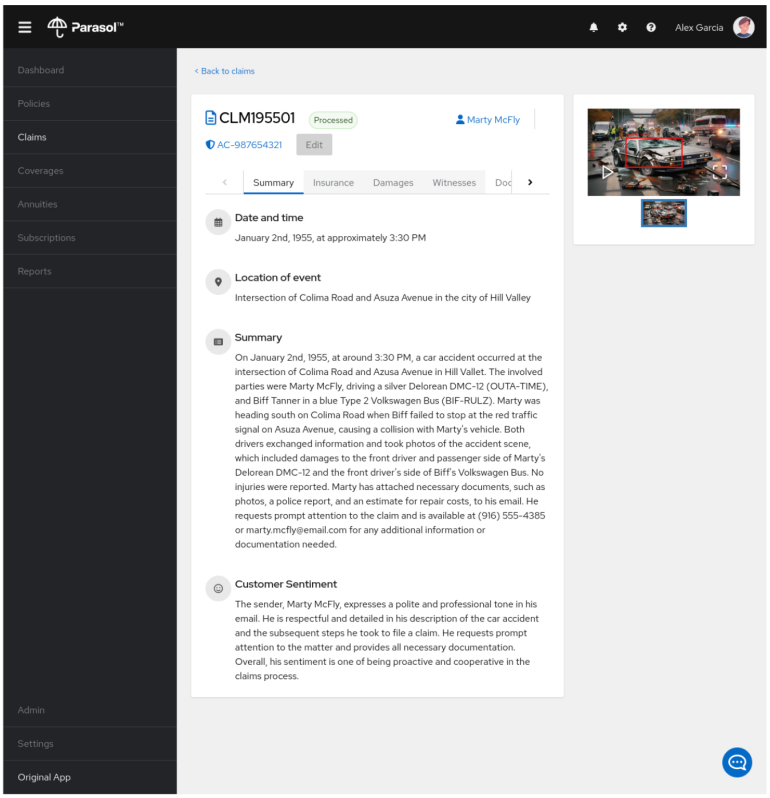
1. While you may currently be in the **Terminals** view, switch to **Parasol** (in the top bar above the upper terminal window) to see the Parasol company's claims application in your browser.





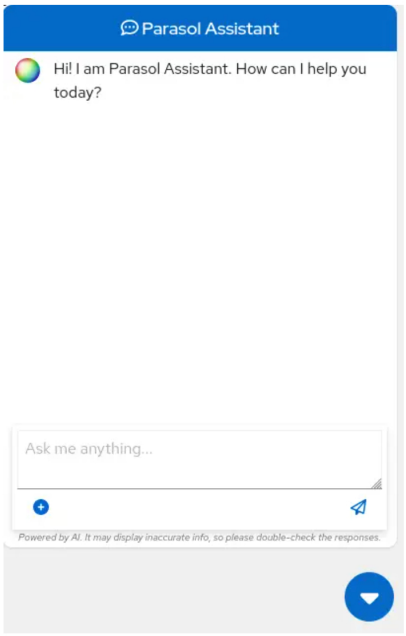
As a claims agent, you can navigate and view the existing claims by clicking on the claim number on the screen.

2. For this lab we will be investigating **CLM195501** which is a claim that has been filed by Marty McFly, let's click on this claim now.



You can read the details of the claim on this page.

3. Once you read the claim, click on the chatbot using the small blue icon in the bottom right of the page.



IMPORTANT

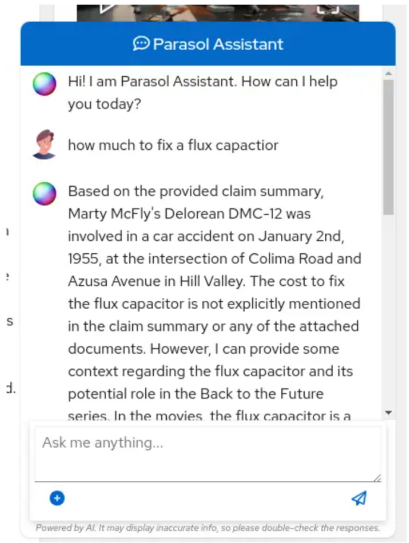
This chatbot is backed by the Granite model you served earlier, so if you killed that running process you will need to restart it in your terminal by running the following: `ilab model serve --model-path /home/instruct/.cache/instructlab/models/granite-7b-lab-Q4_K_M.gguf`

Let's imagine as a claims agent you'd like to know how much it might cost to repair a flux capacitor on Marty's DeLorean.

4. Ask the chatbot the following question:

```
How much does it cost to repair a flux capacitor?
```

You should see something similar to the following. Note that LLMs by nature are non-deterministic. This means that even with the same prompt input, the model will produce varying responses. So, your results may vary slightly.



What we've already started to do is provide contextual information about the claim in each conversation with the LLM using Prompt Engineering. But unfortunately, the chatbot doesn't know how much it costs to repair a flux capacitor, nor will it have any domain-specific knowledge for our organization.

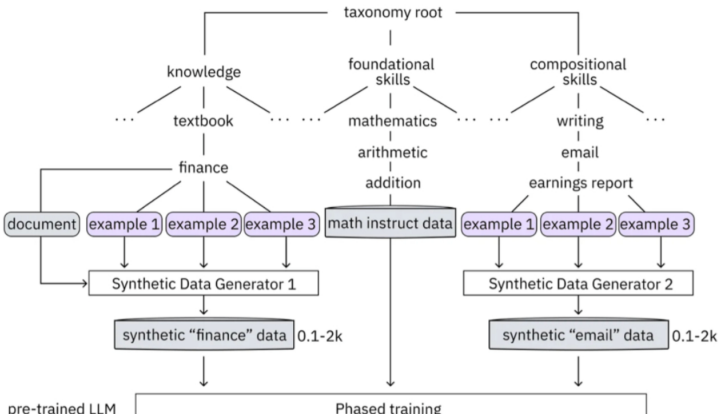
With InstructLab and RHEL AI, we can change that by teaching the model!

3.2. Understanding the Taxonomy

InstructLab uses a novel synthetic data-based alignment tuning method for Large Language Models (LLMs.) The "lab" in InstructLab stands for **L**arge-scale **A**lignment for Chat **B**ots.

The LAB method is driven by taxonomies, which are largely created manually and with care.

InstructLab crowdsources the process of tuning and improving models by collecting two types of data: **knowledge** and **skills**, in the new InstructLab open source community. These submissions are collected in a taxonomy of YAML files to be used in the synthetic data generation process. To help you understand the directory structure of a taxonomy, please refer to the following image.



We are now going to leverage the taxonomy model to teach the model knowledge about a specific vehicle we cover and its details, from our organization's collection of public (and private) internal data.

Navigate back to the **Terminals** view. In the terminal window where you are running chat, enter `exit` to quit the chat session.

1. Navigate to the taxonomy directory.

```
cd /home/instruct/.local/share/instructlab
tree taxonomy | head -n 10
```

You should see the taxonomy directory listed as shown below:

```
taxonomy
├── CODE_OF_CONDUCT.md
├── compositional_skills
│   ├── arts
│   ├── engineering
│   ├── geography
│   ├── grounded
│   │   ├── arts
│   │   ├── engineering
│   │   ├── geography
```

Now, we need to create a directory where we can place our files.

2. Create a directory to add new knowledge, demonstrating how to properly use the taxonomy structure to add knowledge with InstructLab.

```
mkdir -p /home/instruct/.local/share/instructlab/taxonomy/knowledge
```

3. Add new capabilities to our model through new knowledge.

The way the taxonomy approach works is that we provide a file, named `qna.yaml`, that contains a sample data set of questions and answers. This data set will be used in the process of creating many more synthetic data examples, enough to fully influence the model's output. The important thing to understand about the `qna.yaml` file is that it must follow a specific schema for InstructLab to use it to synthetically generate more examples.

The `qna.yaml` file is placed in a folder within the `knowledge` subdirectory of the taxonomy directory. It is placed in a folder with an appropriate name that is aligned with the data topic, as you will see in the below command.

4. Instead of having to type a bunch of information in by hand, simply run the following command to copy this example `qna.yaml` file to your taxonomy directory:

```
cp -av ~/files/backToTheFuture/qna.yaml /home/instruct/.local/share
```

5. You can then verify the file was correctly copied by issuing the following command which will display the first 10 lines of the file:

```
head /home/instruct/.local/share/instructlab/taxonomy/knowledge/par
```

During this workshop, we don't expect you to type all of this information in by hand - we are including the content here for your reference.

It's a YAML file that consists of a list of Q&A examples that will be used by the trainer model to teach the student model. There is also a source document which is a link to a specific commit of a text file in git, where we've included that a flux capacitor costs an affordable $10,000,000.

To help you understand the qna file format, we have included an excerpt of the file below. Feel free to view the entire file on the system with the following command:

```
cat /home/instruct/.local/share/instructlab/taxonomy/knowledge/para
```

```
version: 3
domain: time_travel
created_by: Marty McFly
seed_examples:
  - context: |
      The DeLorean DMC-12 is a sports car manufactured by John DeLorea
      for the American market from 1981 to 1983. The car features gull
      It gained fame for its appearance as the time machine in the "Ba
    questions_and_answers:
      - question: |
          When was the DeLorean manufactured?
        answer: |
          The DeLorean was manufactured from 1981 to 1983.
      - question: |
          Who manufactured the DeLorean DMC-12?
        answer: |
          The DeLorean Motor Company manufactured the DeLorean DMC-12.
      - question: |
          What type of doors does the DeLorean DMC-12 have?
        answer: |
          Gull-wing doors.
document_outline: |
  Details and repair costs on a DeLorean DMC-12 car.
document:
  repo: https://github.com/gshipley/backToTheFuture.git
  commit: 8bd9220c616afe24b9673d94ec1adce85320809c
  patterns:⑥
    - data.md
```

1. **`version`**: The version of the qna.yaml file, this is the format of the file used for SDG (SDG = Synthetic Data Generation). The value must be the number 3.
2. **`created_by`**: Your GitHub username.
3. **`domain`**: Specify the category of the knowledge.
4. **`seed_examples`**: A collection of key/value entries.
    a. **`context`**: A chunk of information from the knowledge document. Each qna.yaml needs five context blocks and has a maximum word count of 500 words.
    b. **`questions_and_answers`**: The parameter that holds your questions and answers
        i. **`question`**: Specify a question for the model. Each qna.yaml file needs at least three question and answer pairs per context chunk with a maximum word count of 250 words.
        ii. **`answer`**: Specify the desired answer from the model. Each qna.yaml file needs at least three question and answer pairs per context chunk with a maximum word count of 250 words.
5. **`document_outline`**: Describe an overview of the document you're submitting.
6. **`document`**: The source of your knowledge contribution.
    a. **`repo`**: The URL to your repository that holds your knowledge markdown files.
    b. **`commit`**: The SHA of the commit in your repository with your knowledge markdown files.
    c. **`patterns`**: A list of glob patterns specifying the markdown files in your repository. Any glob pattern that starts with *, such as *.md, must be quoted due to YAML rules. For example, *.md.

Now, it's time to verify that the seed data is curated properly.

6. Validate your taxonomy

InstructLab allows you to validate your taxonomy files before generating additional data. You can accomplish this by using the **`ilab taxonomy diff`** command as

data. You can accomplish this by using the `ilab taxonomy diff` command as shown below:

NOTE

Make sure you are still in the virtual environment indicated by the (venv) on the command line. If not, source the `venv/bin/activate` file again.

```
ilab taxonomy diff
```

You should see the following output:

```
knowledge/parasol/claims/qna.yaml
Taxonomy in /home/instruct/.local/share/instructlab/taxonomy is valid
```

3.3. Generating Synthetic Data

Okay, so far so good. Now, let's move on to the AWESOME part. We are going to use our taxonomy, which contains our `qna.yaml` file, to have the LLM automatically generate more examples. The generate step can often take a while and is dependent on your hardware and the amount of synthetic data that you want to generate.

InstructLab will generate X number of additional questions and answers based on the samples provided. To give you an idea, it takes 7 minutes when running the default full synthetic data generation pipeline at a scale factor of 30. This can take around 15 minutes using Apple Silicon and depends on many factors. You could customize the scale factor or run a simple pipeline to take less time or if you have lesser hardware, but it is not recommended as it will not generate the optimal output.

However, for the purpose of this workshop we will only generate a small amount of additional samples to give you a sense of how it works.

NOTE

If needed, stop serving the Granite model by typing `CTRL` + `C` in the terminal within which it is running.

We will now run the command (in the second, **bottom** Terminal) to generate the synthetic data. The merlinite model will serve as the **teacher** model:

```
ilab data generate --model /home/instruct/.cache/instructlab/models/
```

After running this command, the magic begins!

NOTE

You will see an `AssertionError` thrown before the SDG process begins. This does not impact the process so please continue without worry.

InstructLab is now synthetically generating data based on the seed data you provided in the `qna.yaml` file.

You will see output on your screen indicating the data is being generated as shown below:

```
INFO 2024-10-21 02:01:23,450 instructlab.sdg.llmblock:51: LLM server s
INFO 2024-10-21 02:01:23,450 instructlab.sdg.pipeline:197: Running blo
INFO 2024-10-21 02:01:23,450 instructlab.sdg.pipeline:198: Dataset({
    features: ['icl_document', 'document', 'document_outline', 'domain
    num_rows: 10
})
```

This will take several minutes to complete.

Once the process completes and we have generated additional data, we can use the `ilab model train` command to incorporate this dataset with the model.

If you are curious to view the data generated, the SDG process creates a jsonl file located in the `/home/instruct/.local/share/instructlab/datasets` directory named knowledge_train_msgs[TIMESTAMP].jsonl

TIP

JSONL files consist of multiple JSON objects, each on its own line.

Feel free to explore. You must input your exact file name in the following command:

```
cat /home/instruct/.local/share/instructlab/datasets/knowledge_train_m
```

NOTE

Using a scale factor of 5 is generally not enough synthetic data to effectively impact the knowledge or skill of a model. However, due to time constraints of this workshop, the goal is to simply show you how this works using real commands. You would typically want to use a scale factor of 30 which is the default value to train the model effectively.

Once the new data has been generated, the next step is to train the model with the updated knowledge. This is performed with the `ilab model train` command.

NOTE

Training using the newly generated data is a time and resource intensive task. Depending on the number of epochs desired, internet connection for safetensor downloading, and other factors, it can take many hours and is highly dependent on the hardware used.

4. Enhancing a LLM with InstructLab

Due to the time constraints of this lab, we will not actually be training the model! This would require a full-scale synthetic data generation process and a training run that could take many hours. You probably have smoewhere else you need to be, so we are going to show you the end results without making you wait.

1. We have provided a model that has already been through this process in your demo system. First, if you have any processes running in either terminal window, type `CTRL` + `C` to exit. In order to serve the newly trained model you can now run the following in the **upper** command window:

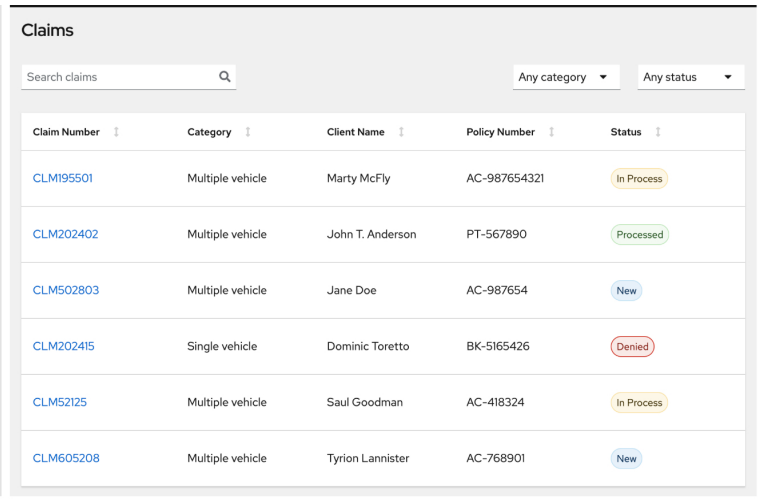
```
ilab model serve --model-path /home/instruct/files/summit-connect-
```

It may take some seconds to start, but you should see the following which should look familiar to you:

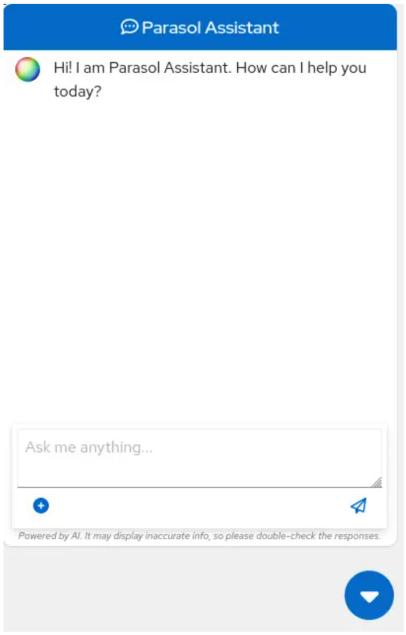
```
INFO 2024-10-20 17:24:33,497 instructlab.model.serve:136: Using model
INFO 2024-10-20 17:24:33,497 instructlab.model.serve:140: Serving mode
INFO 2024-10-20 17:24:34,492 instructlab.model.backends.llama_cpp:232:
 {% for message in messages %}
{% if message['role'] == 'user' %}
{{ '◁user▷
' + message['content'] }}
{% elif message['role'] == 'system' %}
{{ '◁system▷
' + message['content'] }}
{% elif message['role'] == 'assistant' %}
{{ '◁assistant▷
' + message['content'] + eos_token }}
{% endif %}
{% if loop.last and add_generation_prompt %}
{{ '◁assistant▷' }}
{% endif %}
{% endfor %}
INFO 2024-10-20 17:24:34,495 instructlab.model.backends.llama_cpp:189:
INFO 2024-10-20 17:24:34,495 instructlab.model.backends.llama_cpp:190:
```

4.1. Verifying the Application

Now for the moment of truth. You've added knowledge, generated synthetic data, and retrained the model. **Refresh your browser window** where you were viewing Marty McFly's claim in the Parasol insurance application



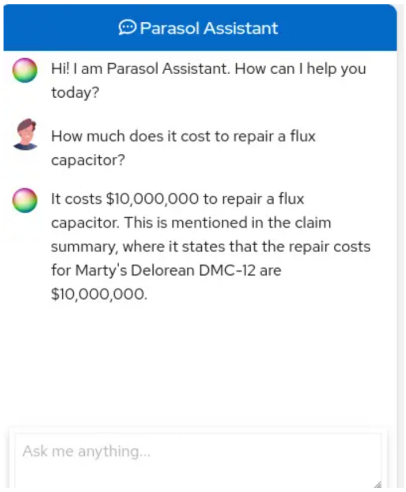
Click on the blue chatbot icon in the bottom right corner of the screen to open the chatbot. If you already have it open you will need to start a new session by pressing the small `+` button on the bottom left-hand corner of the chat window.

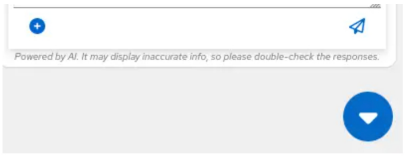


1. Let's ask the chatbot the same question with the newly trained model and see if the response has improved.

```
How much does it cost to repair a flux capacitor?
```

You should see something similar to the following (keep in mind that your output may look different due to the nature of large language models):





CONGRATULATIONS! You just trained a chatbot for Parasol insurance and will make every claims agent's life a little better!

5. Conclusion

Woohoo young padawan, mission accomplished. Breathe in for a bit. We're proud of you, and I dare say you're an AI Engineer now. You're probably wondering what the next steps are, so let me give you some suggestions.

Start playing with both skill and knowledge additions. This is to give something "new" to the model. You give it a chunk of data, something it doesn't know about, and then train it on that. How could InstructLab-trained models help at your company? Which friend will you brag to first?

As you can see, InstructLab is pretty straightforward and most of the time you spend will be on curating new taxonomy content. Again, we're so happy you made it this far, and remember if you have questions we are here to help, and are excited to see what you come up with!

Please visit the official project github at www.github.com/instructlab and check out the community repo to learn about how to get involved with the upstream community! Also, learn more about RHEL AI here (which includes support for InstructLab, idemification for model output for the included Granite large language models, and a platform to run AI your own way on the hybrid cloud).