

webAI Help Docs

Quick Start Guides

Troubleshooting Guide

Installation/Startup Errors

1. Startup gets stuck on step 6/7. This can be caused by a failed Python installation.
 1. **Solution:** Check your antivirus for a Conda or `_Conda` process. You may need to whitelist this process. It can be confirmed with the following error message in the `runtime.logs` file: `level=ERROR msg="Python setup failed! Will be unable to run flows" error="unable to install python 3.10"`
2. Startup gets stuck on step 6/7 or 7/7. This can be also be caused by rogue Runtime processes.
 1. **Solution:** Check activity monitor for an `agent` or `webai-node` process. Manually close those processes and try running Navigator again.
3. XCode needs to be installed
 1. While the installation of XCode CommandLine Tools will be triggered by our runtime agent upon first run if it is not already installed, you can preemptively install this yourself by following the steps below. If you have already used your machine for software development, it is likely that this is already set up on your machine.
 1. Open Terminal, Cmd+Space -> enter "Terminal"
 2. Run this command: `xcode-select --install`
 3. Gui will open, click "Install"
 4. Accept EULA and let it install. Can take a few minutes
 5. Once it is done, in Terminal run: `xcode-select -p`
 6. You should see `/Library/Developer/CommandLineTools`
 7. You are ready.

Flow/Element Errors

1. The object detection element fails abruptly with no error message and all settings correct.
 1. **Solution:** The dataset is formatted incorrectly. Annotations must be done in COCO JSON format and named `annotations.json`. There must be an `images` folder that contains all of the images.

Navigator terminology

Navigator Terminology:

Canvas - Canvas is the main work space of Navigator. This is the creation area to train, test, and prototype your AI models.

Drawer - The drawer is where the library of Elements are stored.

Element - Elements are the building blocks for creating with Navigator. Elements are a package of code meant to do a function such as train an AI model or show a visual output of a running flow.

Flow - A flow is a series of elements connected together in a particular sequence in order to perform a trained function and provide an output. A basic flow will consist of an input element, AI inference element, and an output element.

Deployment - A deployment gives a user the ability to assign flows to a set of computers and input devices, and decouple a running flow from the Navigator UI allowing the flow to run independently.

AI Terminology:

Object Detector - An object detector is a computer vision algorithm or model that identifies and localizes specific objects within an image or video. It can detect multiple objects simultaneously by drawing bounding boxes around them.

Classifier - A classifier is an algorithm or model that categorizes input data into different classes or categories based on predefined criteria or features.

Large Language Model (LLM) - A large language model (LLM) is a powerful artificial intelligence model that is trained on a large amount of text data to generate human-like text responses. LLMs can understand and generate coherent and contextually relevant text, making them useful for various natural language processing tasks.

AI Model Training - AI model training refers to the process of training and optimizing an artificial intelligence model using a datasets. During training, the model learns patterns, features, and relationships within the data to make accurate predictions or generate desired outputs.

AI Model Inference - AI model inference is the process of using a trained artificial intelligence model to make predictions or generate outputs based on new, unseen data. Inference typically involves applying the trained model to input data and obtaining the corresponding output or prediction.

Bounding Box - A bounding box is a rectangular region that encloses or bounds an object within an image or video. It is commonly used in object detection tasks to visually represent the location and size of detected objects.

Model Confidence - Model confidence refers to the level of certainty or trustworthiness associated with the predictions or outputs generated by an AI model. It indicates the model's belief in the correctness or accuracy of its predictions.

Model Accuracy - Model accuracy is a measure of how well an AI model performs on a given task. It represents the percentage of correct predictions or outputs compared to the total number of predictions made. Higher accuracy indicates better performance.

Computer Vision - Computer vision is a field of artificial intelligence that focuses on enabling computers to gain a high-level understanding of visual data, such as images and videos. It involves the development of algorithms and models to analyze, interpret, and extract meaningful information from visual inputs.

Building a dataset: Object Detector

Now that you have identified that an Object Detector is the right option for you and your use case, you have to collect and prepare your data to train an Object Detector. For the below steps, we will use the example of detecting when someone is not wearing proper protective equipment from the Use Cases & AI Architectures document.

Define the Classes

Start by identifying the classes you want your model to predict.

This could be a larger object you want to detect in a frame or something small. For this example use case of wanting to detect whether people are compliant for wearing the proper protective equipment, you may want to detect if they are wearing a hard hat, high visibility vest, safety glasses, etc. We will focus on detecting people not wearing a high visibility vest for this document.

NOTE: The detail of detection will be depend on the training of the model as well as the resolution and clarity of the images fed to the model. Something larger, like the high visibility vest, will be easier to detect than something like safety glasses.

Collect the Data

Now you will need to collect your data. If you already have a dataset collected, then use this time to review your dataset and make sure it is ready to be annotated.

Before you start capturing images, **you will want to make sure the dataset you are curating represents real-world scenarios for your use case.** This is extremely important for your model performance. If you already have a dataset of images, review them to make sure they represent real-world scenarios.

With collecting data, you will need to take into account numerous variables in order to ensure that your model will have the best data to train on.

- Lighting settings
- Image resolution quality
- Camera angles
- Backgrounds and environments
- If images without the object to identify are needed
- If images with both the object and not the object in the same frame are needed

For the example use case, in order for the model to train properly, we will want images of people wearing a high visibility vest and images of people not wearing a high visibility vest.

Organize the Data

Once you have your data collected, you will want to organize and distribute the images between two folders.

- 80% Training images
 - Create a second folder inside your Training images folder and call it "annotations". This is needed for the next step.
- 20% Testing images

Unlike the Image Classifier dataset, you do not need to worry about organizing separate Validation images as Navigator will automatically "create" your Validation images with the training element. Do this by adjusting the Validation Split slider found in the settings panel for the Object Detection Trainer Element. We recommend starting with 20% Validation Split.

Annotate the Data

In order to be able to train an object detector model, you will need to annotate your training dataset so the model knows what to learn and what it is looking for in the image.

To annotate your training images, you will need an annotation tool that allows you to annotate using bounding boxes and can export the annotations in COCO JSON format. We recommend using the MAC application RectLabel Pro.

You will draw a bounding box around the objects in the frame of your training images that you want to assign a class to. For this example, you will want two classes, people wearing a high visibility vest and people not wearing a high visibility vest, to ensure the model knows the difference. You will want to draw your bounding boxes as close to the edge of the object you want to detect. This will ensure the model trains on just the object you want to detect and not anything in between the edge of the item and the edge of the bounding box.

Once you have annotated your training images, then you can move onto training your Object Detector.

Building a dataset: Image Classifier

Now that you have identified that an Image Classifier is the right option for you and your use case, you have to collect and prepare your data to train an Image Classifier. For the below steps, we will use the example of classifying art supplies from the Use Cases & AI Architectures document.

Define the Classes

Start by identifying the classes you want your model to predict.

The first goal when training your model is to prove it out with a smaller set of data, then get more granular as you iterate the model with additional training runs. Each training run will inform what additional data you will need.

- Broad classes, such as Item Use - Drawing supply, Painting supply, Sculpting supply ...
- Granular classes, such as Item Type - Paint Brush, Pencil, Paint, Canvas, Marker, Clay ...
- More Granular classes, such as Paint Brush Type - Flat, Round, Filbert, Fan, Mottler, Oval ...

Collect the Data

Now you will need to collect your data. If you already have a dataset collected, then use this time to review your dataset and make sure it is ready to use.

Before you start capturing images, **you will want to make sure the dataset you are curating represents real-world scenarios for your use case.** This is extremely important for your model performance. If you already have a dataset of images, review them to make sure they represent real-world scenarios.

With collecting data, you will need to take into account numerous variables in order to ensure that your model will have the best data to train on.

- Lighting settings
- Image resolution quality
- Camera angles
- Backgrounds
- Product models
- Product add-on options

Audit the Data

With auditing your dataset, you are checking for formatting, size, and duplicates. Performing this step further ensures your data will give the best opportunity for your AI models performance.

- Start by removing duplicate images and images that are closely related.
 - Duplicate images are considered images that are identical or images that have similar features that aren't distinct from one another.
 - For example, if you have two paint brushes that have the same brush tip and same color body, but are two different brands, then remove one of those images from the dataset.
- Check the format of your images and make sure they are all consistent. If you have a dataset of JPEG's, but you find a couple of PNG's, then you will need to convert those PNG's or remove them from the dataset.
- Ensure your images are roughly the same size for height and width

Structure the Data

Your classes have been defined, images collected, and dataset reviewed. Now you need to structure your dataset to train the Image Classifier.

- Divide and distribute your data into 3 sets:
 - 80% Training images
 - 20% Testing images
- You will need to ensure a balanced distribution of classes within each of the above sets. This is important for optimum model performance and to prevent model bias. For example, using the Item Type classes:
 - 20 images of paint brushes
 - 20 images of pencils
 - 20 images of canvases

- 20 images of paint ...

Organize the Dataset

Last step to building a great Image Classifier dataset is organizing your dataset.

To organize your dataset, you will need to place the images in various folders that correlate with the structure of your dataset as well as the classes you want to train on.

- **Training images set** - Name the main folder as "training" so you can identify that is your training dataset.
 - Inside your training folder, your training images should be organized in folders associated with their class label. For example all Pencil images will be in a folder labeled "pencils".
- **Validation images set** - Follow the same structure as your training images folder except your main folder will be named "validation" to identify this as the validation set.
- **Test images set** - This can be a general folder of images without any structure as this will be used to test the model. Just name the main folder as "test" so you can identify these images as the test set.

Use Cases & AI Architectures: Where do I start?

Identifying Your First Use Case

Before jumping into Navigator, take some time to think about the problems that you think could be solved by AI. The possibilities are limitless, so here are a few to get the creativity flowing.

- You might have years of documentation that takes you hours to sort through to find a single answer. What if you had an expert you could chat with that was trained on these documents and would know the answer to your question in a matter of seconds, saving you hours of time?
- You might be a safety compliance officer and you are hearing reports of some team members not wearing proper protective equipment in designated areas. What if your security cameras could alert you when a person is not wearing the proper safety equipment in that area?
- You might have a database of images that need to be tagged with various features in order to filter the images by type, color, material, etc. What if those images could automatically be tagged, ultimately saving you hours or days worth of manual image processing?

Use Case Solution Planning

Once you have identified a use case, you will want to break your use case down into incremental steps and build up from there.

Using an example from above, let's say you have a dataset of art supply images. You want to be able to easily tag these images, so you decide to train an image classifier with Navigator.

- Start by identifying all the labels/tags you want to filter the images by. This could be very general to very specific. For example:
 - Item Color - Red, Green, Blue, Black, White ...
 - Item Use - Drawing Supply, Painting Supply, Sculpting Supply ...
 - Item Type - Paint Brush, Pencil, Paint, Canvas, Clay ...
- Once your class labels have been identified, start with a broad label to start training and testing your classifier before moving to more specific detailed tagging.
 - In this example, start with the overall Item Use. Train your model on whether the image of the item is Drawing Supply, Painting Supply, Sculpting Supply.
 - Test your trained classifier with a test set of images and measure the accuracy at which it correctly labels the image based on the Item Use.
 - Make adjustments to your model or retrain if needed until the accuracy is acceptable for your use case.
- With your classifier performing as expected for tagging the Item Use, move to training an additional classifier model that would label images by Item Type.
 - Repeat the steps outlined for the Item Use - Train, test, measure, and iterate the model until it performs as expected.
- Before training additional classifiers, build a flow connecting your classifiers in sequence and ensure both models are still providing the correct labels.
- Continue adding additional trained classifier models, if needed, until your images are tagged.

AI Architecture - Which one do I use?

So you have your use case identified as well as a starting point to tackle your solution, but how do you know what AI architecture to use?

Based on your use case and the data you have collected, determine if your use case is a Language (LLM) based solution, or a Computer Vision based solution.

- Large Language Model (LLM) solutions will create an AI subject matter expert you can chat with through text prompts. This AI Expert lives locally on your computer.
 - You need text based data for this solution, such as PDF's, Word Documents (.docx), or Text Documents (.txt)
- Computer Vision solutions create a model to analyze a series of images or video feeds to find an object in the frame or classify the entire image itself.
 - You need image based data for this. The images can be a variety of still images, or it can be pre-recorded videos.

- Computer Vision solutions come in two options depending on you want to solve for your use case.
 - **Classifiers:** A Classifier, such as Resnet, will label an entire image based on the information it is trained on. For example, if you want to identify an image of a Hot Dog vs an image that is Not Hot Dog, then your trained classifier will process the data as it is given and if an image has a Hot Dog in it, then the image is labeled as HOT DOG. If an image does not have a Hot Dog in it, then the image is labeled as NOT HOT DOG.
 - **Object Detector:** An Object Detector is like a Classifier, but has the additional ability to locate the object you trained the model on displaying a bounding box around it. Let's use the previous example of Hot Dog vs Not Hot Dog. The Object Detector will review the image of the hot dog, draw a bounding box around the hot dog and label the bounding box as HOT DOG. With that, the whole image is not being labeled as HOT DOG, but the actual hot dog in the image is being found and labeled as HOT DOG. If no hot dog is found, the image would be labeled as NOT HOT DOG

Prototyping & Hardening for Production

Navigator is a versatile, visual, editor that allows you to easily create, run, and deploy AI Flows. With its lean interface, users can express complex logical flows with it that allow for a high degree of customization.

This article is a guideline to help you get started building your first, simple flows as well as exploring more advanced Flow setups.

Run a first Flow

A great way to start playing around with Navigator is to build a simple Flow that uses your Computer's camera input and sends it directly to an output that you can view on your screen.

Once you log in, click on the Element Drawer at the bottom left of the Canvas and search for the Camera Element. The Camera Element is a placeholder for any video input. Navigator assigns provisions and assigns your default Computer Camera on Startup, so you should already see a "My Webcam" label on the Element. This tells you that the Camera Element will use your Computer's camera as an input and provide it as its output for the next Element in your Flow.

Go back to the Element Drawer and add "Output Preview" to Canvas and place it to the right side of the Camera Element. Output Preview will simply display the input that is handed to it. Click and drag the connector dot on the right side of the Camera element to the connector dot on the left side of the Output Preview.

Now run the flow by clicking the Play button in the top right corner of the Canvas. Navigator will first install required dependencies on your machine*, if needed, and then run the Flow. Once the red Pause button appears instead of the spinner you know that the Flow is running. At the same time your built-in Camera should activate and the Output Preview Element will show an “Open” button. Click the Open button in the Output Preview Element to open a preview window that displays the video feed from the Camera Element.

To end the Flow, simply click the Pause Button in the upper right corner of the Canvas. Congratulations, you ran your first Flow!

NOTE: If dependencies need to be installed, it may take a moment for the model to start. The following model runs will start faster.

Run a first Inference

The previous example illustrates how to connect Elements and run a Flow in Navigator. But the real power of Navigator and the underlying Runtime is to execute AI Flows on your Hardware. In the following, we will build a simple Inference Flow that will detect Objects within the frame of your Camera.

Start with the Flow that you built in the previous step. We will keep the setup the same and add Inference in between. Open the Element Drawer, search for “Object Detector”, and drag it onto Canvas. This Inference Element runs YoLo under the hood to detect objects on frames. It takes frames as input and produces annotated frames (bounding box, probability, object label) as output. Rewire the Camera Element to connect to the Object Detector, and connect the Object Detector to the Output Preview input.

Make sure that the Camera Element has your Camera assigned and the other two Elements have your Computer assigned.

You can now run the Flow. Once the Open button shows up on the Output Preview, you can open it and watch YoLo detect objects in real time.

Deployment: How to implement your AI model

Navigator's Canvas is a great tool to test out your AI Flows and quickly iterate on them. Running a flow is coupled to the Navigator application though, which means that the moment you close Navigator or shut down your machine, any flow that might still be running will be stopped.

In an actual production setup you will probably want to decouple running a flow from Navigator. This can be achieved via “Deployments”, a feature that allows you to assign flows to a set of computers and input devices.

Where do my Deployments run?

Navigator is the interactive UI that allows you to create and manage your AI Flows, as well as deploy them to a given Hardware-setup. The execution of these Flows is handled by the webAI Runtime Agent. Within your local network, one of these Agents takes the role of the Controller. For as long as this Controller is running, the Flows are being executed.

Currently, your Navigator installation will start Runtime for you in the background and default to Controller, which in turn means that Deployments are coupled to the lifetime of Navigator. We will soon give you the tools to set this up specifically to your needs.

How to create a Deployment?

Designing your Flow

In order to create a Deployment, you start by designing your AI Flow on Canvas. Canvas will automatically provision and assign your computer and your local camera to any Element you pull onto Canvas, to provide a local test environment.

Saving a Version of your Flow

Once you are happy with your Flow and it is ready to be deployed, you need to save a Version of it. A Flow Version is a snapshot of the Flow in its current configuration that you can refer to later on when creating Deployments.

- Click on the context menu of your project on the left (three dots) and then select “Save Version for Deployment”.
- You can review all Versions you saved when clicking on the context menu of the project and then selecting “Show Flow Versions”.
- You can also rename and delete previously saved Versions in the Versions Panel.

Setting up your Devices

As mentioned above, Navigator will auto-assign your computer and your camera to all Elements pulled onto it. For a Deployment, you will likely want to use different Devices to deploy your AI Flow to. In order to make the Devices available for Deployments, as well as for Canvas, you need to provision them on the Devices page.

Navigator differentiates between two kinds of Devices, Input Devices and Computers. Input Devices are currently limited to Cameras, but will soon be extended with additional options. Computers are computing devices that run the webAI Runtime Agent to power your AI Flows.

To provision your Devices, click on the Button with the same name in Navigator. Provisioned Devices are available to all Projects you are working on.

Creating a Deployment

Deployments are associated with the currently selected Project. This means, all Deployments you create will be organized within that Project and have access to the Flow Versions of that Project only. You can access the Deployments screen by clicking on the drop-down menu next to the run button. This allows you to toggle between Canvas and Deployments.

- To create a new Deployment, click on the Create button.
- In the process you can give a name and a description, and you will have to select a Version that you would like to deploy.
- Based on this Version, you need to assign a set of provisioned Devices to the Elements that are part of the Flow. Input types like “Camera” require a provisioned Input Device, Elements like “Person Detector” require a provisioned Computer to run on.
- Once you have assigned all Elements to their corresponding provisioned Devices, you can “Build” the Deployment.

All currently running Deployments are listed in the table along with their current status. It is possible to rename, delete, stop, and restart them. You can also perform these actions in bulk by selecting multiple Deployments.

AI Models: How to create a computer vision model

If you want to create a vision based solution for your use case, then training a computer vision model is what you need. Follow the below steps to begin training your own LLM that lives locally, on your device.

How to train a computer vision model

1. Start by creating a new Canvas
2. Drag either Object Detection or Classification Trainer Element onto the Canvas
3. Go to settings of the Training Element. For the training data path, select the location of the training data that will be used. Note: For object detection training, all images must be in the same folder and formatted in the **COCO JSON** format.
4. Select the output artifact path of where you would like the training artifacts to be saved.
5. All other settings can be left as the default.
6. Click the Run button on the Canvas. Note: The first time this flow runs, it may take a moment due to downloading dependencies.
7. Once the training is completed, you will have a selectable computer vision inference model.

How to use your trained computer vision inference element

1. To use your newly trained inference model, create a new canvas.

2. Then drag the inference element of the type of computer vision model you trained. This will be either an Object Detection Inference Element or a Classifier Inference Element.
3. Once on the canvas, open the inference element's settings and select either your trained artifact from the dropdown or file selector, not both.
4. Now you will want to build a flow by dragging an input element onto the canvas, either Camera Element or Media Loader Element, and connecting it to the input side of your inference element. Then drag an output element, Output Preview, onto the canvas and connect it to the output side of your inference element.

NOTE: If you are using a Camera Element, you must first have an input device configured under the Devices tab.

AI Models: How to create a local expert LLM

If you want to create your own subject matter expert to solve your use case, then training your own local LLM is your best bet. Follow the below steps to begin training your own LLM that lives locally, on your device.

LLM Dataset Generation

Before you can generate your dataset, make sure you have your documents you want to use ready. Gather all relevant documents you want your model to be built off of. These documents must be in one folder and in the following formats - PDFs, text, and docx.

1. Start by creating a new Canvas. Then open the Elements Drawer and drag the LLM Dataset Generator element onto the Canvas.
2. Open the LLM Dataset Generator Element settings and adjust the following settings:
 1. **Topic:** This can be anything you would like.
 2. **References folder path:** Using the "Select Directory" button, choose the folder where your documents are located.
 3. **Output folder path:** Using the "Select Directory" button, choose the folder where you would like to save the output of the dataset generation
 4. **Dataset size:** Add the number of topics you want your dataset to train with.

NOTE: We recommend starting with 5 for testing and getting familiar with the process of dataset generation. This generates a list of five topics and is quicker for training, but it will not produce as accurate of a model as a larger dataset size.

The higher the dataset size, the more accurate your dataset and trained model will be. However, the larger the dataset size, the longer it will take to generate your dataset. It can take several hours to generate large dataset, so be patient.

3. Next, enter your GPT, Claude, and Gemini API keys.
4. Now you can now hit run. Dependencies will be installed the first time this flow is run, so it may take a while for them to install.

LLM Model Training

Now that you have generated your LLM Dataset, you can train your LLM Model.

1. Start by creating a new Canvas. Drag the LLM Trainer Element onto the Canvas.
2. Open the LLM Trainer Element settings and make the following adjustments
 1. **Dataset Folder Path:** Using the “Select Directory” button, choose the folder where you saved your LLM dataset during LLM Dataset Generation.
 2. **Artifact Save Path:** Using the “Select Directory” button, choose the folder where you would like to save your trained adapter.
 3. **Base Model Assets Path:** Using the “Select Directory” button, choose the folder where you would like to save your base model.
 4. **Evaluator API Key:** Add a Groq, OpenAI, Claude, or Gemini API key to enable the Faithfulness and Relevancy benchmarks in your training metrics. If you need a free API key, you can [generate one for Groq here](#).
 5. **Batch Size:** 4 is recommended for testing
3. Leave all other settings as the default.
4. You can now hit run. This process may take a while, so be patient.

LLM Model Inference

You have generated your LLM Dataset and you have trained your LLM Model. Now we can use our LLM Inference Model and interact with our trained expert.

1. Start by creating a new Canvas and drag the LLM Chat element onto the Canvas.
2. Open the LLM Chat Element setting and and make the following adjustments:
 1. **Max token:** 256 is recommended for testing
 2. **Model Storage Path:** Using the “Select Directory” button, choose the folder where you saved your base model during LLM Model training.

3. **Model Adapter Folder Path:** Using the “Select Directory” button, choose the same folder where you saved trained adapter during LLM model training
3. Leave all other settings as the default
4. Now drag the Prompt API and Response API elements to the canvas
5. Connect the Prompt API to the LLM chat input. Then connect the output of the LLM chat to the Response API. The flow on your canvas should look like this.
6. You can now hit run.

NOTE: Dependencies will be installed the first time this flow is run and may take a while.

Element Library

Navigator comes with a range of stock elements so you can start creating right away. Here is the list of all the stock elements, what they do, and how to use them. New elements are added regularly, and you can make your own to customize your solution for production.

Navigator Pro Elements

Inputs

- Camera - connect a webcam or network camera (IP or RTSP) as a data feed
- Media Loader - upload images or videos to test models and see the inference outputs. Not necessary for training or dataset generation
- Prompt API - Send LLM prompts to the LLM Chat or Document QnA elements
- LLM Dataset Generator - Standalone element that processes text documents and prepares them for training an LLM with the option to augment that data with information from popular LLMs.
 - Supports Word Docs, PDFs, and .txt files

Outputs

- Response API - Connect to the output end of the LLM Chat or Document QnA elements to get responses back from the API

Training

All Training elements need to be in their own Project to train effectively. Specify where the trained model should be stored on your computer using the Output Artifact Path setting.

- LLM Trainer - Train your own custom LLM with this element. Select a supported model, connect your dataset from the LLM Dataset Generator, and click Run. For more info on how to build an LLM Dataset, read our deep dive here.

Inference

- Document QnA - Add documents for a model to query and give answers from, then connect it to stock open source, or tuned custom models. Also known as RAG (Retrieval Augmented Generation)
- LLM Model Chat - Load your custom tuned LLM (from the LLM Trainer) and connect it to data inputs and outputs.
- Text Reader - A pre-trained Optical Character Recognition model. Connect it to a camera feed or the Media Loader as an input to read text in images and videos.

Enterprise Elements

The Enterprise plan includes our Vision suite in addition to the LLM suite available with Navigator Pro. To start a trial of the Enterprise Plan, contact our team [HERE](#).

Outputs

- Image Regions: Specify Regions of Interest in images or videos for the model to focus on. Regions of interest are a portion of an image you want to filter or operate on in some way.
- Zone Counter - Similar to Output Preview, but keeps count of the objects
- Output Preview - preview window to show images or camera feeds with inference and bounding boxes.

Training

All Training elements need to be in their own Project to train effectively. Specify where the trained model should be stored on your computer using the Output Artifact Path setting.

- Deep Detection Lite Trainer - webAI proprietary object detector model that requires less data to reach similar accuracy levels as other object detector models. Upload a folder of images with a COCO JSON annotations file. File name of the annotations should be `annotations.json`. For more info on how to build an Object Detection Dataset, read our deep dive here.

- Image Classification Trainer - Train a ResNet Classification Model. Upload a folder of images organized into sub-folders labeled with the class names, define where you want your Model to be saved, and hit Run. For more info on how to build an Image Classification Dataset, read our Deep Dive here.
- Object Detection Trainer - Train a YOLOv8 object detector. Upload a folder of images with a COCO JSON annotations file. File name of the annotations should be `annotations.json`. For more info on how to build an Object Detection Dataset, read our deep dive here.

Inference

- Deep Detection Lite Inference - webAI proprietary object detector inference model. Connect a data source and output element such as Output Preview to see the inference on your images or video.
- Image Classification Inference - understand and categorize images under a specific label. Requires data inputs, a trained classification model, and an output element such as Output Preview to see the inference.
- Object Detection Inference - Detect and locate objects in images or videos using models you've trained with the Object Detection Trainer. Connect to a data source and an output element such as Output Preview to see the inference.
- Object Detector - A pre-trained object detection model running the YOLOv8 model. Connect it to a camera feed and an output element such as Output Preview to see bounding boxes around objects

Other

- Barcode Reader - A pre-trained Barcode Reader vision model. Select the Barcode type, connect a data source such as a Camera and an output like Output Preview to read barcodes.
- Class Filter - A whitelist filter element for Object Detectors. Connect it as the output to an Object Detector or Object Detection Inference element, open the settings and type the objects you'd like to have detected. Connect an output element like Output Preview to see the inference and bounding boxes.
- Object Tracker - a pre-trained Object Detection model that tracks objects across a screen. Tracking differs from Detection in that the model has the concept of object permanence.

Supported LLM Base Models

Navigator supports a variety of open source LLMs, and is adding more all the time. Below is the current list of supported models with links to their Hugging Face model pages where you can learn more about each model's strengths and weaknesses. Change the model you are training [here](#)

For most use cases that you expect to train or run inference on consumer hardware, we recommend using a 7B parameter model. This is the current sweet spot between model performance and size for consumer devices. All LLM features in Navigator work well with 7B parameter models.

Full list of supported Models

[TinyLlama/TinyLlama-1.1B-Chat-v1.0](#)

[gradientai/Llama-3-8B-Instruct-262k](#)

[mlx-community/Meta-Llama-3-8B-Instruct # 4 bit & 8 bit](#)

[mlx-community/gemma-1.1-2b-it-4bit](#)

[mlx-community/Mixtral-8x22B-Instruct-v0.1-4bit](#)

[mlx-community/WizardLM2-8x22B-4bit-mlx](#)

[mlx-community/Meta-Llama-3-70B-Instruct-4bit](#)

[mlx-community/Phi-3-mini-4k-instruct # 4 bit & 8 bit](#)

[mlx-community/Phi-3-mini-128k-instruct # 4 bit & 8 bit](#)

[mlx-community/OpenELM-270M-Instruct](#)

[mlx-community/OpenELM-450M-Instruct](#)

[mlx-community/OpenELM-1_1B-Instruct # 4 bit & 8 bit](#)

[mlx-community/Qwen1.5-1.8B-Chat-4bit](#)

[mlx-community/Qwen1.5-0.5B-Chat-4bit](#)

[mlx-community/Qwen1.5-7B-Chat-4bit](#)

[mlx-community/Qwen1.5-72B-Chat-4bit](#)

[Maykeye/TinyLLama-v0](#)

Templates

Templates: Document QnA

Want use LLMs to query or summarize documents or other information privately and securely? The Document QnA is the fastest way to get started. Follow the steps below to upload your documents and get going.

1. Open the Document QnA Element settings and make the following adjustments.
2. **Trained Artifact:** If you have previously trained a custom LLM you can select it here. It will override the Base Model Architecture and Quantization Rank settings.
3. **Base Model Architecture:** Select any of the supported models, or leave this as the default.
4. **References Folder Path:** Upload a folder of containing the documents you want to interact with. It can be any mix of Word Docs, PDFs, or .txt files.
 1. Even if you have only 1 document it needs to be in a folder.
5. **Base Model Storage Path:** Select where you want the base LLM model to be stored on your machine.
6. **Model System Prompt:** This default prompts your model to act as an assistant. You can leave this setting alone, or play with it to change how the model responds.
 1. For example: ask the model to talk like a Pirate or Rock Star, speak in a casual tone, or speak in like a poet.
7. **Leave all other settings as default.**
8. **Press Run**
9. **Start chatting!**

If you want a deeper dive into all of the settings to really customize your Document QnA flow, read our guide here.

Templates: Local Expert

To use this template, you'll need a Trained Model. If you haven't created one yet, you can

- [Download a pre-made one here](#)
 - [Create your own using this guide](#)
1. Start by creating a new Canvas and drag the LLM Chat element onto the Canvas.
 2. Open the LLM Chat Element setting and and make the following adjustments:
 3.
 1. **Model Dropdown:** Select a model you have trained yourself. If you haven't trained a model, download a pre-made one at the link above and add it to the *Model Adapter Folder Path*.

2. **Model Storage Path:** Using the “Select Directory” button, choose the folder where you want the base model to be saved.
3. **Model Adapter Folder Path:** Use this to upload trained models from us, or models shared by others.
4. Leave all other settings as the default
5. You can now hit run.

NOTE: Dependencies will be installed the first time this flow is run and may take a while.

Templates: Custom LLM Chatbot

To use this template, you'll need a Trained Model. If you haven't created one yet, you can

- [Download a pre-made one here](#)
 - [Create your own using this guide](#)
6. Start by creating a new Canvas and drag the LLM Chat element onto the Canvas.
 7. Open the LLM Chat Element setting and and make the following adjustments:
 8.
 1. **Model Dropdown:** Select a model you have trained yourself. If you haven't trained a model, download a pre-made one at the link above and add it to the *Model Adapter Folder Path*.
 2. **Model Storage Path:** Using the “Select Directory” button, choose the folder where you want the base model to be saved.
 3. **Model Adapter Folder Path:** Use this to upload trained models from us, or models shared by others.
 9. Leave all other settings as the default
 10. You can now hit run.

NOTE: Dependencies will be installed the first time this flow is run and may take a while.

Templates: Train an LLM

LLM Model Training

Now that you have generated your LLM Dataset, you can train your custom LLM Model.

1. Open the LLM Trainer Element settings and make the following adjustments

2. **Base Model Architecture:** The default model is great for creating a model quickly. Want to train something tailored to a specific idea? Check out our Supported LLM Base Models page to learn more about all the models in Navigator
3. **Dataset Folder Path:** Using the “Select Directory” button, choose the folder where you saved your LLM dataset during LLM Dataset Generation.
4. **Artifact Save Path:** Using the “Select Directory” button, choose the folder where you would like to save your trained adapter.
5. **Base Model Assets Path:** Using the “Select Directory” button, choose the folder where you would like to save your base model.
6. **Evaluator API Key:** Add a Groq, OpenAI, Claude, or Gemini API key to enable the Faithfulness and Relevancy benchmarks in your training metrics. If you need a free API key, you can [generate one for Groq here](#).
7. **Batch Size:** 4 is recommended for testing
8. Leave all other settings as the default.
9. You can now hit run. This process may take a while, so be patient.

For detailed breakdown on LLM Training check out our in-depth article [here](#).

Templates: LLM Dataset Generation

LLM Dataset Generation

Before you can generate your dataset, make sure you have your documents you want to use ready. Gather all relevant documents you want your model to be built off of. These documents must be in one folder and in the following formats - PDFs, text, and docx. You can have a mix of any and all types in the same folder.

1. Open the LLM Dataset Generator Element settings and adjust the following settings:
2. Topic: This can be anything you would like
3. References folder path: Using the “Select Directory” button, choose the folder where your documents are located.
4. **Dataset size:** Add the number of topics you want your dataset to train with.

NOTE: We recommend starting with 5 for testing and getting familiar with the process of dataset generation. This generates a list of five topics and is quicker for training, but it will not produce as accurate of a model as a larger dataset size.

The higher the dataset size, the more accurate your dataset and trained model will be. However, the larger the dataset size, the longer it will take to generate your dataset. It can take several hours to generate large dataset, so be patient.

5. Next, enter your Groq, GPT, Claude, or Gemini API keys. You can add as many as you like, but Groq or two others are required. [You can get a free Groq key here](#).

6. Now you can now hit run. Dependencies will be installed the first time this flow is run, so it may take a while for them to install.
7. The output will be a folder with the name `dataset_[your_topic_name]_[timestamp]`
8. This folder is what you connect to the Dataset Folder Path in the LLM Trainer Element in the Training step.

For a complete deep dive into Dataset Generation, See the LMM Dataset Generator article.

Elements

Create your own elements - Element SDK

Code for elements that are executed by

[Runtime](<https://gitlab.com/iris-core/systems-senior/webai-2.0-infrastructure>)

and can be configured in

[Navigator](<https://gitlab.com/iris-core/ant-senior/navigator>).

Elements are python programs that receive and/or emit data from/to other elements they are connected to in the context of a flow.

Folder structure

```
```text
```

```
|— elements | Elements
```

```
└— wheels | Shared Python libraries
```

```
```
```

Downloading pre-built elements

In CI/CD, we package up all available elements for you to download [here](https://gitlab.com/iris-core/systems-senior/runtime/webai-elements/-/artifacts). See `builder` instructions below on how to import into a running Runtime agent.

Setup

1. Set up [Navigator](https://gitlab.com/iris-core/ant-senior/navigator) and [Runtime](https://gitlab.com/iris-core/systems-senior/webai-2.0-infrastructure) according to the instructions in their own repos.
1. Ensure no elements are installed at this point by checking the corresponding UI section in Navigator.
1. Start the runtime agent, as it must be running for elements to be installed.
1. Install all elements (in [development mode](#development-mode)) by running `make install BUILDER=/path/to/builder` from this repo's folder.
 - The `BUILDER` environment variable should point to the runtime's `builder` executable.
 - To install elements in development mode (see more about this below), replace `install` with `dev-install`. This is useful for actively developing elements.

See all available commands by running `make`.

Installing elements individually

Note the runtime agent must be running for elements to be able to be installed.

To install an element individually, run:

1. ``/path/to/builder package /path/to/element``
2. ``/path/to/builder import /path/to/package/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx.zip``

Note the ``.zip`` package path should be logged by the first command.

Development mode

To install an element in development mode, use the alternative ``dev-import`` command (instead of ``package`` + ``import`` commands):

```
`path/to/builder dev-import path/to/element`
```

Installing elements in development mode means using the ``builder dev-import`` command from runtime, which installs elements in a way where changes to the code are immediately reflected in the next flow execution.

An exception to this are changes to element configuration in ``publish.json``, such as inputs/outputs or settings - which require the element to be reinstalled to have an effect.

Elements

Creating a new element [(script available here)](/scripts/generate_new_element.sh)

A new element can be **automatically generated** from the command line at root with `./scripts/generate_new_element.sh`. You may need to run `chmod +x ./scripts/generate_new_element.sh` in order to make the script executable.

When running this script, simply type a space separated name for your new element. The script will:

1. create the proper element directory structure (see below)
2. generate the required files (see below)
3. automatically generate boilerplate in each of the required files

To **manually** create a new element, create a new `hyphen-separated` directory under `[./elements/](/elements)`. For example, if your element is called "new element," the directory should be named `elements/new-element/`.

The files required are:

- `setup.py`

- `requirements.txt` specifying package dependencies, including the `[element framework](#element-framework)`
- `publish.json` that can be `[generated by the element framework](#publishjson)`.

We expect this requirement to be removed in the future.

- `__init__.py` inside a `snake_case` subdirectory. For example:
`new-element/new_element/__init__.py`. This subdirectory is the entry point into the element's code. Other python scripts for the element should be included in this same subdirectory.

Renaming an existing element `[(script available here)](/scripts/rename_existing_element.sh)`

You can rename an existing element from the command line at root with

`./scripts/rename_existing_element.sh`. You may need to run

`chmod +x ./scripts/rename_existing_element.sh` in order to make the script executable.

When running the script, the CLI will prompt you to select an existing element to rename. Once you do this, it will prompt you to provide input for the new name. After confirming, the script will rename all instances of the following input string variants (see 1-4 in the example below) in the hyphen-separated parent element directory, as well as the underscore_separated element subdirectory. The script renames the directories as well.

...

Example:

Selected element to rename: "media loader"

User input for new name: "photo packer"

1. lowercase space separated: "media loader" -> "photo packer"
2. capitalized space separated: "Media Loader" -> "Photo Packer"
3. lowercase hyphen separated: "media-loader" -> "photo-packer"
4. lowercase underscore separated: "media_loader" -> "photo_packer"
5. hyphen separated parent directory: "media-loader/" -> "photo-packer/"
6. underscore separated subdirectory: "media_loader" -> "photo_packer/"

...

Keep in mind, there may be some exceptions of different string variants that the script does not cover. For example, "Bounding Box UI" does not fall under the 2nd variant, "capitalized space separated" (technically, it would be Bounding Box Ui). In cases like this, it is useful to do a manual passthrough after running the script to verify any lingering instances of the element's former name have been properly converted to the new name.

publish.json

Elements currently require some duplication of information between their code

and a `publish.json` file. We expect the need to maintain `publish.json` files alongside elements to be removed in the near future.

To generate a `publish.json` file for an element, ensure the element framework wheel is installed in the current environment and run
`python -m framework generate --path <path to element>` (`--path` can be omitted if the command is run from the element's folder).

Element framework

All elements depend on the element framework, which is the library that manages element lifecycles.

The wheel for the element framework is available in

[this read-only registry](<https://gitlab.com/iris-core/systems-senior/webai-2.0-infrastructure/-/packages>),

and it must be included in every element's `requirements.txt` with the following

(replacing `X.X.X` with the appropriate version number):

```
``text
--extra-index-url https://gitlab.com/api/v4/projects/49121232/packages/pypi/simple
framework==X.X.X
...

```

Development environment

VSCode

This repository includes VSCode workspace settings and a set of recommended extensions (in the `[`.vscode` folder](/.vscode)`).

Type checking

We use pyright for type checking. For the best experience, ensure your editor is respecting ``pyrightconfig.json`` (VSCode provides this functionality through the `[python](https://marketplace.visualstudio.com/items?itemName=ms-python.python)` extension).

Formatting

``black`` and ``isort`` are used to format python code. VSCode extension recommendations and workspace settings for these are included in this repo.

Python virtual environments

When working on an element, it is useful to set up a development environment with proper type checking & auto completion. Doing so requires installing

element dependencies into a virtual environment, and pointing your editor to this virtual environment. To do so:

1. ``cd`` into your element's folder under ``/elements``.
1. Create and activate a virtual environment with ``python -m venv venv && source venv/bin/activate``. Note this virtual environment will only be used for development environment purposes, and is not made use of during element execution.
1. Install the element's dependencies into the newly created & activated virtual environment with ``pip install -r requirements.txt``.

The final step is to point your editor to this virtual environment. For VSCode, this means:

1. Running "Python: Select Interpreter" from the ``CMD + SHIFT + P`` command palette.
1. Selecting "Enter interpreter path...".
1. Pasting the path to the ``venv`` folder. This can be easily obtained by right clicking the folder in the VSCode files explorer and selecting "Copy Path".

It may be useful to have a project-level virtual environment as well, which avoids frequent switching - but keep in mind that it makes it harder to verify whether an element's ``requirements.txt`` is correct.

In a project-level virtual environment, the element framework can be installed

with

```
`pip install framework --upgrade --extra-index-url  
https://gitlab.com/api/v4/projects/49121232/packages/pypi/simple`,
```

which does not require creating a `requirements.txt`.

Object Detector

A [YOLOv8 object detector](#) that can identify [91 different objects](#). For in-depth information on the data used to train the model, you can [find that here](#).

Connect to inputs such as a Camera or Media Loader and outputs such as Output Preview or Zone Counter.

If you want to identify certain objects but not others, connect the Class Filter element.

Class Filter

Limit the number of classes an object detector will identify by creating a white-list of the classes you want to detect. Simply open the settings and list the class names you want to detect separated by commas.

Object Tracker

Tracks objects that have been identified with an object detector or classifier across a video feed.

Connect with inputs like a camera and outputs such as Output Preview or Zone Counter.

Barcode Reader

Scan barcodes with this pre-trained Barcode reader. Connect inputs such as camera feeds or the Media Loader and outputs such as Output Preview or Zone Counter.

Reads the following barcode types:

- EAN_13
- EAN_8
- UPC_A
- UPC_E
- QR_CODE

Zone Counter

The Zone Counter keeps track of the number of objects detected within certain zones. Use this instead of Output Preview as the output element for object detection or classification models.

Text Reader

Need to process/read text from an image or video feed? This element is a pre-trained Optical Character Recognition (OCR) model.

Connect it to input elements such as Cameras or the Media Loader and outputs such as Output Preview to see the inference.

Object Detection Inference

You've trained an Object Detector, now its time to start doing inference.

1. Add the Object Detector Inference element to canvas and open the settings.
2. Select the model from the drop down, or add upload it via the Model Artifact directory.
3. Adjust the Minimum Confidence Threshold selector to your desired level to for an inference to be registered.
4. Connect the element to inputs or outputs such as a Camera or Media Loader and Output Preview.

Image Classification Trainer

Classifiers are computer vision models that determine whether an object is present in a frame or not (think Hot Dog, No Hot Dog). Not sure that's the right model for your solution? Check out this [Guide to Use Cases and Model Architectures](#).

You will also need a dataset ready to train. If you don't have one, check out our guide for [choosing free datasets or creating your own](#).

Now that you're ready to train, drag the Image Classification Trainer element to canvas and open up the settings:

Training Run Name: Give your training run a name so you can reference it later.

Training Data Path: Select the top level folder your training data is stored in.

Validation Split: Set your validation split. This is the group of images that will be used to give the model feedback on how the training is going.

Model Backbone: Select your Model Backbone - each is good at its own thing. We currently support:

ResNet 50
ResNet 50 V2
ResNet 101
ResNet 101 V2
ResNet 152
ResNet 152 V2

You can find additional documentation on computer vision models [here](#).

Adjust the following settings or leave as Default

Number of Head Layers: Additional layers that are tacked on the end of the model to train (more increases training time)

Number of Head Layer Units: How wide of a head layer you would like to train, if there is more than one head layer element creates multiple head layers starting at the number of head layer units and decreasing by a factor of 2 (512 head layer units and 2 head layers -> head of 512->256->num_classes)

Number of Epochs: One Epoch is a complete pass of the entire training dataset through the training algorithm.

Freeze Backbone: Trains only the head model when set to True.

Early Stopping Patience: Number of Epochs to wait if the loss doesn't change meaningfully before stopping training early

Seed: Deterministic seed. Uses this for the random number generators to start them at similar values

Model Optimizer: Learn more about the optimizers [HERE](#)

Image Classification Inference

You've trained a classifier, now its time to start doing inference.

1. Add the Image Classification Inference element to canvas and open the settings.
2. Select the model from the drop down, or add upload it via the Model Artifact directory. 1.
3. Adjust the Minimum Confidence Threshold selector to your desired level to for an inference to be registered.

4. Connect the element to inputs or outputs such as a Camera or Media Loader and Output Preview.

Output Preview

Output Preview is the Navigator-native preview window to see the inference on your vision specific models.

Drag the element to canvas to the end of the flow and connect the last element to the endpoint on the left of the element. Most common inference elements using Output Preview are:

Object Detector
Object Detection Inference
Object Tracker
Classifier Inference
Class Filter
Barcode Reader
Text reader

Open the settings to select the Minimum level of Confidence for bounding boxes being drawn in the output preview.

Media Loader

The Media Loader is the way to upload images and videos into Navigator to be processed and inferred. Drag the Media Loader to canvas and open the settings.

Select the folder of images or videos to upload. Supported file formats:

- .jpg
- .png
- .jpeg
- .npy
- .raw
- .mp4
- .avi
- .mov

Frame Rate: How fast the image or video frames are processed. The “0” Default setting processes images as 1 per second and videos to the FPS they are set at in the file.

Stay Alive: After all the files are uploaded and processed, this element will shut down. If you need it to stay alive for your flow, select this setting.

Camera

Camera elements are how you connect a range of cameras to your models and solutions.

Navigator supports:

Computer cameras: Built-in cameras, or USB port connected cameras

Network cameras: Accessible through a URL such as an IP or RTSP Camera

To connect a camera, drag the element to canvas and select which camera you'd like to use in the settings. Rename the camera if you like.

To add an additional camera:

Open the Devices Menu. If you have not added the computer where the camera is connected to the Device Registry yet, you will need to connect it before adding the camera. [Learn how to Add A Device Here.](#)

Select 'Input Devices' and 'Add Input Device'

Select which type of camera you're connecting and click next.

For Computer Cameras: give it a name, select the computer its attached to, and input the ID linked with the camera.

For Network Cameras: give it a name, add the URL where the camera is located, select the computer that should be analyzing the video.

LLM Dataset Generator

The LLM Dataset Generator is the first step to creating a custom LLM. Whether you're creating a local expert or a custom model to use with the Document QnA element

Before you can generate your dataset, make sure you have your documents you want to use ready. Gather all relevant documents you want your model to be built off of. These documents must be in one folder and in the following formats - PDFs, text, and docx.

1. Start by creating a new Canvas. Then open the Elements Drawer and drag the LLM Dataset Generator element onto the Canvas.
2. Open the LLM Dataset Generator Element settings and adjust the following settings:

1. **Topic:** This can be anything you would like.
2. **References folder path:** Using the “Select Directory” button, choose the folder where your documents are located.
3. **Output folder path:** Using the “Select Directory” button, choose the folder where you would like to save the output of the dataset generation
4. **Dataset size:** Add the number of topics you want your dataset to train with.

NOTE: We recommend starting with 5 for testing and getting familiar with the process of dataset generation. This generates a list of five topics and is quicker for training, but it will not produce as accurate of a model as a larger dataset size.

The higher the dataset size, the more accurate your dataset and trained model will be. However, the larger the dataset size, the longer it will take to generate your dataset. It can take several hours to generate large dataset, so be patient.

3. Next, enter your Groq, GPT, Claude, or Gemini API keys. You can add as many as you like, but 1 is required. [You can get a free Groq key here.](#)
4. Now you can now hit run. Dependencies will be installed the first time this flow is run, so it may take a while for them to install.
5. The output will be a folder with the name `dataset_[your_topic_name]_[timestamp]`
6. This folder is what you connect to the Dataset Folder Path in the LLM Trainer Element in the Training step.

NEXT: Train your custom LLM with the LLM Trainer Element

LLM Trainer

Now that you have generated your LLM Dataset, you can train your LLM Model. This article is a deep dive into the LLM Trainer and all its settings. To get a short training overview, check out [Templates: Train an LLM](#)

1. Start by creating a new Canvas. Drag the LLM Trainer Element onto the Canvas.
2. Open the LLM Trainer Element settings and make the following adjustments
 1. **Dataset Folder Path:** Using the “Select Directory” button, choose the folder where you saved your LLM dataset during LLM Dataset Generation.

2. **Artifact Save Path:** An optional setting to create a backup save of the adapter file outside of Navigator's built in Artifact Registry.
3. **Base Model Assets Path:** Using the "Select Directory" button, choose the folder where you would like to save your base model.
4. **Evaluator API Key:** Add a Groq, OpenAI, Claude, or Gemini API key to enable the Faithfulness and Relevancy benchmarks in your training metrics. You will still get [BLEU](#) and [ROUGE](#) scores without adding an API key

If you need a free API key, you can [generate one for Groq here.](#)

Model Selection

Navigator supports a variety of open source LLMs, and is adding more all the time. Change the model you are training here

For most use cases that you expect to train or run inference on consumer hardware, we recommend using a 7B parameter model. This is the current sweet spot between model performance and size for consumer devices. All LLM features in Navigator work well with 7B parameter models.

See the full list of supported models [HERE](#)

Advanced Settings

1. **Dataset Utilization Rate:** Percentage that sets how much of the dataset to use.
2. **Batch size:** Number of entries of the dataset to train at one time. Lower numbers use less memory, higher numbers train faster. We recommend starting with 4
3. **Learning Rate:** Determines the step size or rate at which the model weights (parameters) are updated during training. Lower means a smaller step and likely slower convergence, but is more stable compared to a higher learning rate
4. **Quantization Rank:** Reduces the number of bits used to train each parameter. Think of this like compressing an audio file, or lower resolution images or videos. Smaller Quantization Ranks use less memory but lose accuracy.
5. **Seed:** Starting point for random number generation, usually used for repeatability in experiments

Training Metrics

To see the training metrics of current and past runs, click the “Canvas” Dropdown at the top of the menu, then select “Run History”

From here you can see the Summary and Report pages for each of your runs

Hover over the tool tip at the end of each score to learn more about what each one means for your model.

NEXT: Add your model to the LLM Chat Element

LLM Chat

You have generated your LLM Dataset and you have trained your LLM Model. Now we can use our LLM Chat and interact with the trained expert.

1. Start by creating a new Canvas and drag the LLM Chat element onto the Canvas.
2. Open the LLM Chat Element setting and make the following adjustments:
 1. Select a Trained Artifact from the drop down. This is where the models you’ve trained with the LLM Trainer are stored. If you don’t have a trained artifact, you will chat with the pre-trained base model.
 2. **Base Model Architecture:** This is the core model you’ll be fine tuning. Select any model from the dropdown.
 3. **Model System Prompt:** This default prompts your model to act as an assistant. You can leave this setting alone, or play with it to change how the model responds.
3. For example: ask the model to talk like a Pirate or Rock Star, speak in a casual tone, or speak in plain language.

If you have brand voice guidelines, this is the place to add them.

4. **Max token:** 256 is recommended for testing
5. **Model Storage Path:** Using the “Select Directory” button, choose the folder where you want to save the base model for the chat.
6. **Model Adapter Folder Path:** A backup to the built in model artifact registry. If you have a model that is not in your registry, you can plug it in here.
7. **Max Tokens:** A setting limiting the number of tokens from the LLMs output. By default this is loaded from the Trained Artifact
8. **Temperature:** A balance between predictability and creativity. Lower settings

prioritize learned patterns, giving more deterministic outputs. Higher temperatures encourage creativity and diversity.

Connect the Chat Element with the Prompt API input and the Response API output to chat with your model.

Document QnA

Document QnA (also known as RAG, Retrieval-Augmented Generation) is an element to query and interact with documents. Once you load them in, you can ask questions of the whole document repository and the model will respond with answers based on the documents. The model will also provide citations from where it got its information in your documents.

This article is a deep dive into the Document QnA Element. To get started quickly, check out [Templates: Internal Document Expert](#)

1. Start by creating a new Canvas and drag the Document QnA element onto the Canvas.
 1. Open the Document QnA settings and make the following adjustments:
 1. If you have a trained model you'd like to use, select the Trained Artifact from the drop down. This is where the models you've trained with the LLM Trainer are stored. This is an optional step for Document QnA.
 2. Base Model Architecture. This is the base model you will be chatting with. For a full list of supported models and their strengths and weaknesses, see our Supported LLM Base Models.
 3. **Embedding/Retrieval Model:** Embedding models are similar to LLMs in the sense that each has a different size and capability. The default (mixedbread) offers a great balance of speed and performance but here is a list of all of them:
https://www.sbert.net/docs/sentence_transformer/pretrained_models.html
 4. **Model System Prompt:** This default prompts your model to act as an assistant. You can leave this setting alone, or play with it to change how the model responds.
 2. For example: ask the model to talk like a Pirate or Rock Star, speak in a casual tone, or speak in plain language.

If you have brand voice guidelines, this is the place to add them.

5. **Temperature:** A balance between predictability and creativity. Lower settings prioritize learned patterns, giving more deterministic outputs. Higher

temperatures encourage creativity and diversity.

6. **Model Storage Path:** Using the “Select Directory” button, choose the folder where you want to save the base model for the chat.

7. **Max Tokens:** A setting limiting the number of tokens from the LLMs output. By default this is loaded from the Trained Artifact

8. **Model Adapter Folder Path:** A backup to the built in model artifact registry. If you have a model that is not in your registry, you can plug it in here.

9. **Document Chunk Size:** How big of a chunk each document gets broken up into, i.e. how many words (500 words for instance is default)

10. **Document Overlap:** How many words to overlap per chunk.

11. **Number of Docs per Query:** Number of chunks to retrieve per query of the database

Connect the Document QnA element to the Prompt API and Response API to interact with your model.