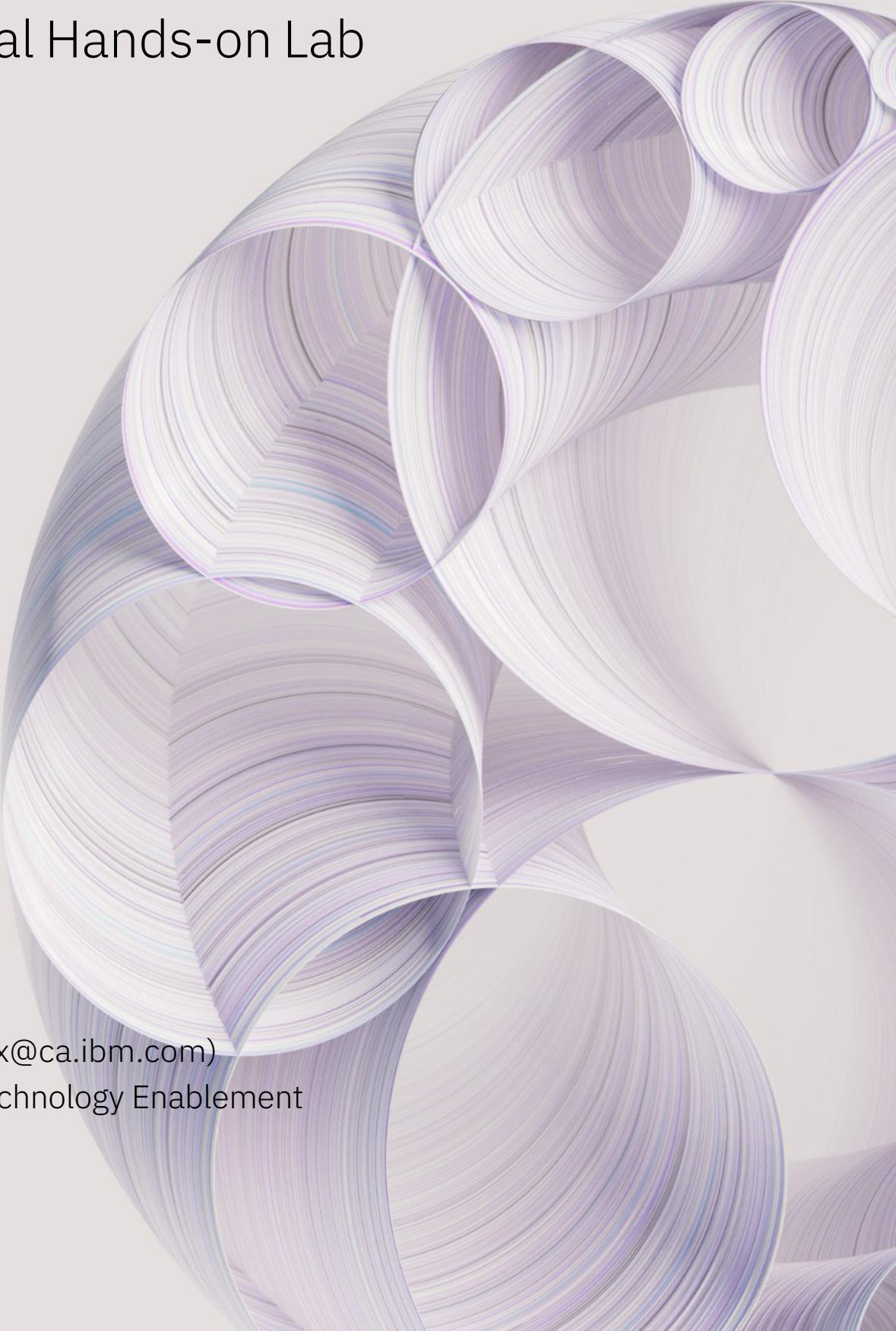


# IBM watsonx.ai

## Technical Hands-on Lab



Felix Lee ([felix@ca.ibm.com](mailto:felix@ca.ibm.com))  
Worldwide Technology Enablement

# Contents

1. Introducing watsonx.ai .....	3
2. About this Lab .....	4
2.1 Disclaimer.....	4
3. Getting Help.....	6
4. Prerequisites & Getting Started .....	7
5. Watsonx.ai console .....	20
6. Watsonx.ai Prompt Lab .....	23
6.1 Basic Navigation .....	23
6.1.1 Open up a new Prompt Lab session .....	26
6.2 Exploring different foundation models and their characteristics.....	27
6.3 Prompt Engineering – updating a zero-shot prompt .....	38
6.4 Prompt Engineering - modifying parameters.....	44
6.5 Working with lists.....	55
6.6 Working with JSON .....	62
6.7 Working with one-shot prompting .....	70
6.8 Saving prompts and prompt sessions .....	76
6.9 Using Jupyter notebooks with prompts.....	90
6.9.1 Getting your API key.....	90
6.9.2 Creating a Jupyter notebook from a prompt .....	94
6.10 Further exploring llama-2 .....	102
6.11 Generating and working with code.....	106
8. Summary .....	118
Appendix A. Troubleshooting .....	120
Appendix B. Generative AI and Foundation Model concepts.....	121
Appendix C. Revision History.....	123

## 1. Introducing watsonx.ai

Watsonx.ai is a core component of watsonx, IBM's enterprise-ready AI and data platform designed to multiply the impact of AI across an enterprise's business.

The watsonx platform has three powerful components: watsonx.ai for new foundation models, generative AI and Machine Learning (traditional AI); and the watsonx.data fit-for-purpose data store that provides the flexibility of a data lake with the performance of a data warehouse; and the watsonx.governance toolkit, that enables AI workflows to be built with responsibility, transparency, and explainability.

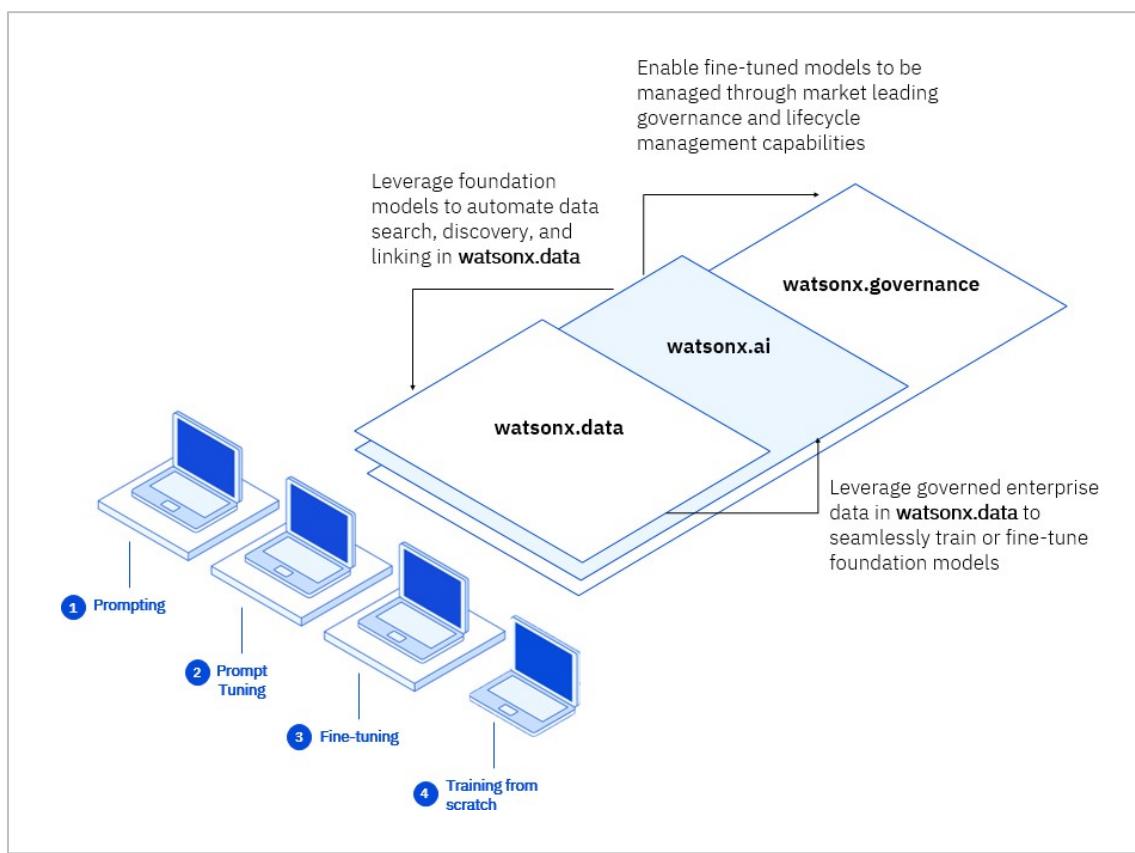


Figure 1: IBM watsonx platform

The watsonx.ai component (the focus of this lab) makes it possible for enterprises to train, validate, tune, and deploy AI models – both traditional AI and generative AI. With watsonx.ai, enterprises can leverage their existing traditional AI investment as well as exploit the innovations and potential of generative AI builds on foundation models to bring advanced automation and AI-infused applications to reduce cost, improve efficiency, scale, and accelerate the impact of AI across their organizations.

## 2. About this Lab

This IBM watsonx.ai hands-on lab introduces you to some of the core components and capabilities of IBM watsonx.ai. By completing this lab, you will gain the experience needed to demo these capabilities to clients.

Specifically, you will get hands-on experience in the following areas:

- The watsonx.ai web-based user interface (UI) which includes the Prompt Lab, the Structured and Freeform interface, model parameter configuration panels, and model information panels.
- Strengths and weaknesses of different models
- Meaning of different configuration parameters and how they influence a model in generating output
- Zero-shot and one-shot prompting (and by extension few-shot prompting)
- Generating list and JSON files with foundation models
- Code generation and translation
- Saving prompts as well as prompt sessions (to keep the history of all your changes)
- Restoring the prompt to an earlier state via prompt history
- Saving prompts to a Jupyter Notebook and working with the Jupyter Notebook

### 2.1 Disclaimer

Watsonx.ai is being developed and released in an agile manner. In addition to adding new capabilities, the web interface is likely to change over time. Therefore, the screenshots used in this lab may not always look exactly like what you see. You can expect to encounter some of the following:

- Additional foundation models in the library list.
- Changes in the user interface (location of buttons, text for various fields)
- Additional tabs/buttons (especially when the Tuning Studio is rolled out).

These should not affect how the labs work, but have patience and explore.

There are three changes, however, that can affect the results

- Foundation models can be very sensitive to input. If you enter slightly different text than what the exercise is using (even if it is just one single word), the outcome can be different.
- There are ongoing tuning of the models. If the models themselves are updated, then some of the results may vary.
- The sample prompts may change (more can be added, or updated with new text). Since prompt examples are used heavily in this lab, this can affect what you might see. To mitigate that, the prompt example text used will be provided in a gray text box. If you select a prompt example and the text is different, simply copy the given text into various input fields.

**Important notes:**

- Watsonx.ai (as of March 2024) opens up in the **Chat** tab – a new tab in addition to the **Structured** and **Freeform** tabs. This is not discussed in this lab (and the screen captures may not show it). Simply choose **Structured** or **Freeform** as directed by the lab instructions. **Chat** will be covered in a future update.
- New models have been introduced since the 2<sup>nd</sup> re-write of this lab guide. For example:
  - **mixtral-8x7b-instruct-v01-q** – made with AutoGPTQ, which is optimized for GPU consumption.
  - **codellama-34b-instruct-hf** – this is a code-related Large Language Model (LLM) built on top of llama-2.
  - **granite-20b-multilingual** – part of the IBM granite family of models, trained for multilingual tasks.

Please raise questions on the [\*\*#data-ai-demo-feedback\*\*](#) Slack channel (IBMer only). IBM partners can request help at the [\*\*Partner Plus Support\*\*](#) website.

### 3. Getting Help

Lab guide help: If you require assistance in interpreting any of the steps in this lab, please post your questions to the [`#data-ai-demo-feedback`](#) Slack channel (IBMer only). IBM partners can request help at the [Partner Plus Support](#) website.

Watsonx.ai: Assistance with the watsonx.ai product itself is available in the [`#watsonx-ai- feedback`](#) (IBMer) and the [`#watsonx-ai-enablement`](#) Slack channels (IBMer only). Additionally, please refer to the [watsonx.ai documentation](#) as needed.

Additional troubleshooting: See [Appendix A. Troubleshooting](#) at the end of this lab guide for guidance on commonly encountered issues.

## 4. Prerequisites & Getting Started

You can use IBM TechZone to perform the exercises in this document.

### Important note:

**You must use a TechZone account** to perform this lab, or you will run into missing service or out of tokens issues. When you use TechZone, you **will not (and should not)** need to subscribe to any additional services (such as the free Lite Watson Machine Learning service).

In fact, if you add the Lite version of the Watson Machine Learning service by mistake, you will be using that service (instead of the one associated with the TechZone account) and you will likely encounter an “out of token” message.

The detailed setup instructions are as follows.

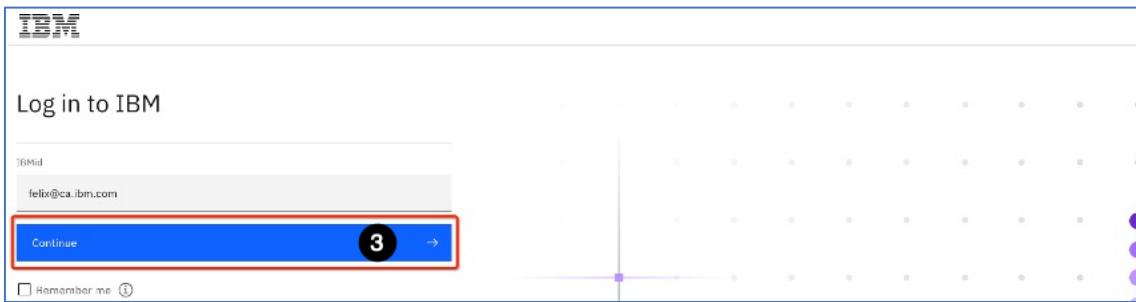
10. Go to the [IBM TechZone](#) website.

11. If you already have an IBMid, skip to Step 3. If you do not have one, click [Create an IBMid](#).



Simply follow the instructions from the subsequent panels to create your IBMid.

12. Provide your IBMid and click **Continue**.



13. Select your Single-Sign-On method. The **w3id Credentials** option is selected in this example:

A screenshot of the "w3id on IBM Security Verify" sign-in page. It displays three options: "w3id Credentials" (selected and highlighted with a red box and the number 4), "QR Code", and "Security Key / Touch ID". Each option includes a brief description and a note about registration. At the bottom, there is a message: "We're enhancing w3id. Learn more".

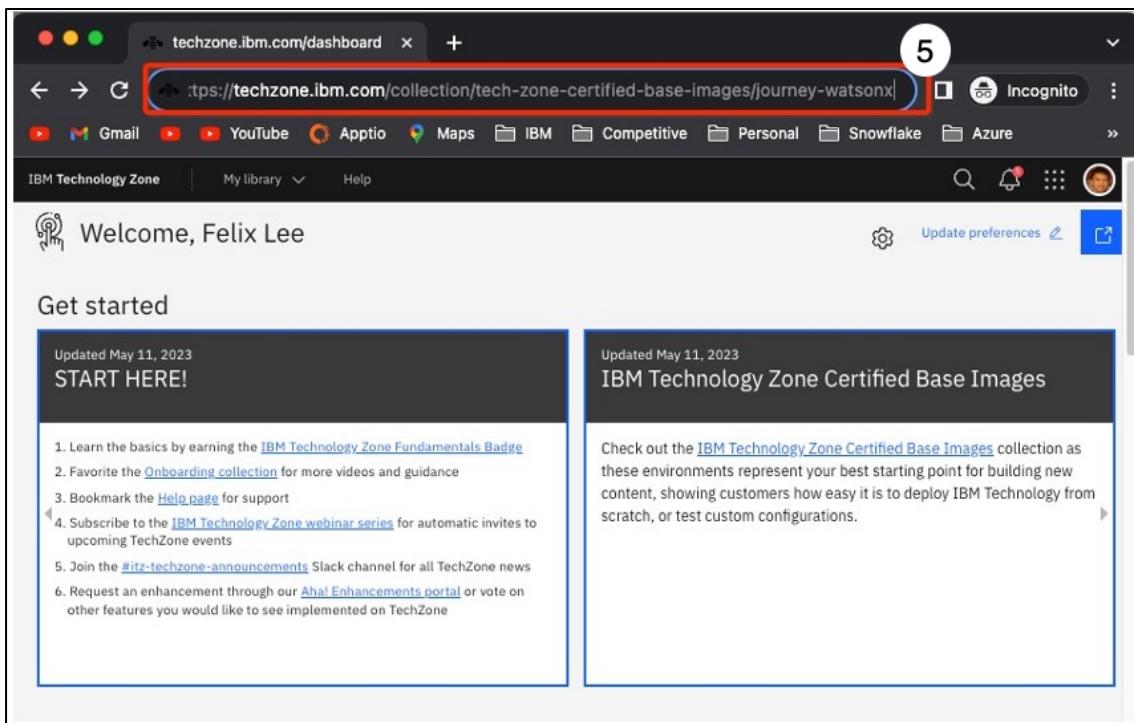
5. Provide the necessary information (w3id and password, QR code, etc.) and sign in.

If this is the very first time you are logging onto the TechZone, you will be brought to the TechZone dashboard. Copy and paste the following HTTP address in the browser's search field (do not use the magnifying glass icon) and press Enter:

<https://techzone.ibm.com/collection/tech-zone-certified-base-images/journey-watsonx>

to be taken to the **TechZone Certified Base Images** page.

If you have been to TechZone before, you might be taken directly to the TechZone Certified Base Images page already. In that case, proceed to Step 6.



6. Your browser will open to the TechZone Certified Base Images page. On the left-hand side, the **watsonx** section should be highlighted. If not, select it.

The screenshot shows the TechZone Certified Base Images page. On the left sidebar, under the 'Environment' section, the 'watsonx' item is highlighted with a red box and a circled number 6. The main content area features a banner for 'TechZone Certified Base Images' with a 'watsonx' badge. Below the banner, there are three sections: 'Developer Base Image' (docker-based), 'Stand Alone Base Image' (OpenShift-based), and 'SaaS Offerings'. Each section has a brief description and a timestamp. To the right, there are two detailed cards for 'watsonx.data Developer Base Image' and 'watsonx.data Standalone - OCP on VMWare'. The bottom right corner of the page has a feedback icon.

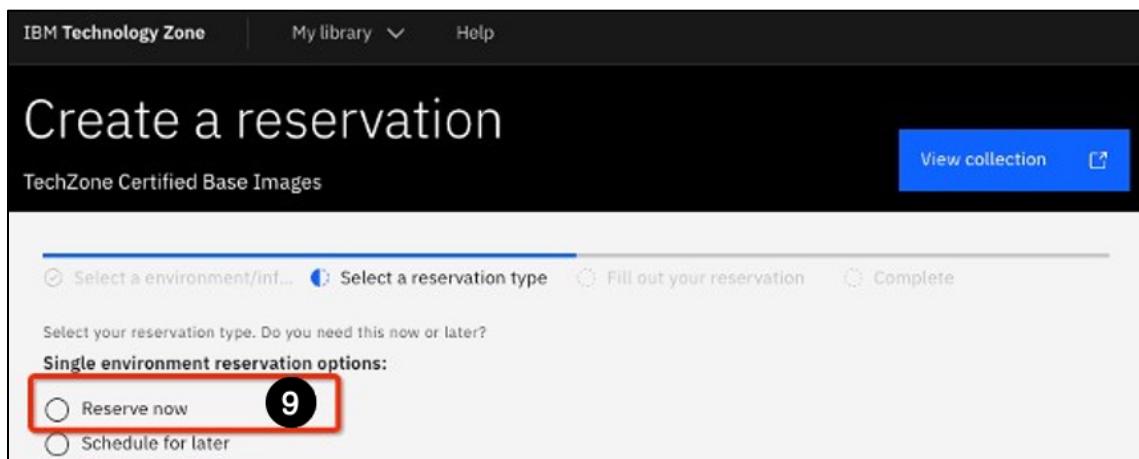
7. Scroll down until you find the **watsonx.ai/.governance SaaS** section (highlighted in red below).

The screenshot shows the IBM Technology Zone interface. On the left, there's a sidebar with categories like Overview, Pre-Installed Software, IBM Cloud Environments, Fyre Environments, Power & Z Systems, Premium - AWS, Azure, ROKS, and **watsonx**. The **watsonx** category is expanded, showing sub-sections like Environments, Metadata, and Comments. In the main content area, there are several tiles. One tile for 'watsonx.ai/.governance SaaS' is highlighted with a red box and has the number '7' in a circle above it. This tile includes details such as 'Apr 21, 2024', 'IBmcloud 2: any, any, any', and a description about dedicated or shared instances of watsonx.ai. Below the tile is a note about accepting an IBM Cloud invitation. Another part of the tile says 'Be sure to accept the IBM Cloud invitation to the correct itz-watsonx-n account on your first reservation of this. Consecutive reservations won't require you to accept the invite.' There are also sections for 'Visibility' and 'Reserve'.

8. Click on Reserve in the **watsonx.ai/.governance SaaS** tile

This screenshot shows a close-up of the reservation interface for the **watsonx.ai/.governance SaaS** tile. It displays the 'Visibility' information ('IBMers, Business Partners') and the prominent 'Reserve' button, which is highlighted with a red box and has the number '8' in a circle above it. To the right of the 'Reserve' button is a sun icon.

9. You will now be on the reservation site for a watsonx.ai environment. Select Reserve now. If you are not ready to start, you can select the **Schedule for later** radio button – the steps are the same.



Now perform the following:

10. Change the **Name** of the reservation to <your id> watsonx.ai L3 SaaS
11. Select the **Practice/Self-Education** tile
12. For the **Purpose description**, enter **Completing L3 Lab**
13. Select your preferred **Geography**. In the example here **AMERICAS** is chosen.

Name  
felix|watsonx.ai|governance SaaS

Purpose

Customer Demo  
Showcasing a pre-built, standardized product use case with minimal customization needed on the environment for a client demonstration. Aligns with 'Custom Demo' in IBM Sales Cloud.

Practice / Self-Education  
Gaining experience with specific technology, product, or solution.

Proof-of-Technology  
Customizing an environment to set up and personalize beyond the pre-built option to showcase a specific customer use case. Typically these environments are needed for a longer period of time. Aligns with 'Proof of Value' in IBM Sales Cloud.

Test  
Need to test a specific function, configuration, or customization.

Please ensure to select the correct purpose as this can **NOT** be updated or changed after this reservation has been created. Review the [Reservation Duration Policy](#) to understand default durations allowed for specific infrastructures based on purpose.

Sales Opportunity number  
Enter an opportunity number  
Providing an [IBM Sales Cloud opportunity number, Gainsight Relationship ID, or a Project Work ID](#) will allow you to extend your reservation date.

Purpose description  
Completing L3 Lab

What are you doing? Why do you need this? What are you trying to accomplish?

Preferred Geography  
any - AMERICAS - any region - any datacenter

**Note:** For any data center (such as Dallas used in this example), multiple IDs are used by TechZone. It is important that you identify the ID TechZone assigns, which may be different from one reservation to another.

For now, continue to the bottom part of the reservation page:

14. Specify the **End date and time**. By default, the end date is 2 days from the current date when you reserve **watsonx.ai/governance SaaS**. You can extend it up to a maximum of 6 days.

Note that this limit may differ among different TechZone images.

15. Select **No** (should be the default) for installing Db2

16. You can optionally provide some additional notes if you choose

17. On the right-hand side at the bottom, accept the **Terms and Conditions**, then click **Submit**.

The screenshot shows a reservation form with the following highlighted areas:

- Step 14:** A red box highlights the "End date and time" section, which includes fields for date (04/28/2024), time (12:20 PM), and location (America/Toronto). Below the date field is a note: "Reservation policy: Recommended 2 months, but can be reserved up to 2 months on this reservation form. Extend later for 5 weeks increments up to 3 months total. Max time 5 months total."
- Step 15:** A red box highlights the "Install DB2?" dropdown menu, which is set to "No".
- Step 16:** A red box highlights the "Notes" section, which contains the placeholder text "Enter any notes you would like to attach to this reservation".
- Step 17:** A red box highlights the "Submit" button at the bottom right of the form.

18. You will get an email sent to your IBMid email address letting you know that your reservation is being provisioned (this should happen within 10-15 minutes). Once the watsonx.ai provisioning is completed, you will receive a second email telling you that it is ready for use. This email looks like the example below.

If you have not done so (in a previous reservation) you will need to click on the **notifications page** link (as highlighted below) to go and accept the invitation.

# IBM Technology Zone

Status Update: Ready

Your reserved environment is ready to use.

**Reservation Name:**  
watsonx Assistant, Discovery or watsonx Discovery, Speech and watsonx.ai

**Reservation ID:**  
[66043a447f56ae001e10b783](#)

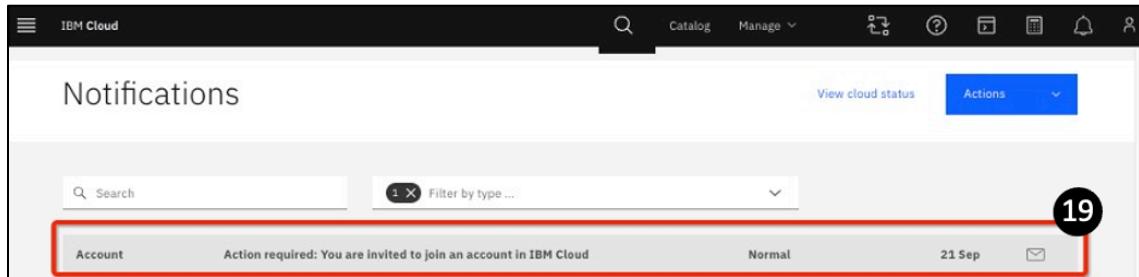
**Step 1 - Getting Access:**  
This environment requires access to our TechZone owned IBM Cloud account where this environment is hosted. Please login to your IBM Cloud account and navigate to your [notifications page](#) to accept the invitation for accesss to the environment.

**Step 2 - Navigate to your Reservation:** 18  
Select "View My Reservations" button to navigate to your reservations.

[View My Reservations](#)

**Step 3 - Leverage Content:**  
- Check the Collection for more information and guidance provided by the content author on how to best leverage this environment:  
<https://techzone.ibm.com/collection/63860f633718a600183c731e>  
- Leverage the [watsonx troubleshooting guide](#) for additional guidance on how to access, create projects, share projects, and more.

19. Clicking on **notifications page** brings you to a login page where you need to provide your IBMid (and may have to log in with your credentials if you have not already done so). Then you will see the Action required item. Click on the item.



20. The following page opens up. Click **Join now**.

Action required: You are invited to join an account in IBM Cloud

**Update time:** 21 Sep 2023, 4:26 PM local time

**Severity:** Normal

Hi felix@ca.ibm.com,

Data Steward invited you to join the following IBM Cloud account: Techzone Outcomes  
(Account ID: feb0a088323a45db90b8dd694b314c53).

[Join now.](#)

20

The link to join the account expires in 30 days.

21. You will now be sent back to the IBM Cloud login page. Use your IBMid to log in. You are asked to join the TechZone account. Accept the Terms and Conditions, then click Join Account.

IBM Cloud

Catalog Cost estimator Docs

Join IBM Cloud

You're almost there. Complete your IBM Cloud account details so that you can get started.

Account information

felix@ca.ibm.com

Personal information

Felix Lee  
United States

Account notice

I accept the product Terms and Conditions of this registration form.

Build your journey to public cloud  
Build, deploy, and manage solutions in IBM's public cloud.

Join Account 21

22. When you log in, **ensure you are using the right account**. Assuming that you have accepted the invitation, you should have an itz-watsonx -n account. Go to the menu bar and click on the  icon to slide open the list of accounts available to you, and ensure that you select the account that was assigned to you from your current TechZone reservation.

- a) You can always check your reservation to find the right account. To do so, log on to <https://techzone.ibm.com>. The dashboard page will open.

The screenshot shows the IBM Technology Zone dashboard. At the top, there's a navigation bar with links for 'IBM Technology Zone', 'My library', and 'Help'. Below the navigation is a user profile section with a welcome message 'Welcome, Felix Lee' and icons for search, notifications, and preferences. The main content area has a title 'Get started'. It features two cards: one for 'watsonx Demo Hub' (updated Oct 26, 2023) and another for 'Welcome to IBM Technology Zone!' (updated Oct 26, 2023). Both cards contain descriptive text and links to more information.

- b) Select the **My Library** pull-down and click on **My reservations**

This screenshot shows the same dashboard as above, but with the 'My library' dropdown menu open. The menu includes options like 'My created collections', 'My bookmarked searches', 'My favorites', and 'My reservations'. The 'My reservations' option is highlighted with a red box and a circled '22b'.

- c) The **My reservations** page opens. Select the correct reservation tile (you may have more than one). In the example below it is the **watsonx.ai/.governance SaaS** tile. Click on it to open up the detailed reservation information page.

- d) Scroll down and find **Cloud Account** which provides the correct account that TechZone has assigned for this reservation. As shown in the example, it is **ITZ-WATSONX**.

Purpose	Opportunity ID(s)
Purpose Practice / Self-Education	
Opportunity Product(s)	Opportunity description L3 material update testing
Customer(s)	
Environment	
Note: Optimized by <a href="#">IBM Turbonomic</a>	
Reservation ID 65d8a9a3945cf30011ec4ffd	Type IBM Cloud
Request method watsonx-ai-saas	Transaction ID 4c271485-74c0-41e8-8b72-2b612caae525
Cloud Account ITZ-WATSONX	Geo americas
Region us-south	Datacenter dal10
Customer data false	Environment watsonxai-vhuxxrw8
Idle runtime limit 10800	Timeout action

- e) In the screenshot below, the account **itz-watsonx** is being used as an example.

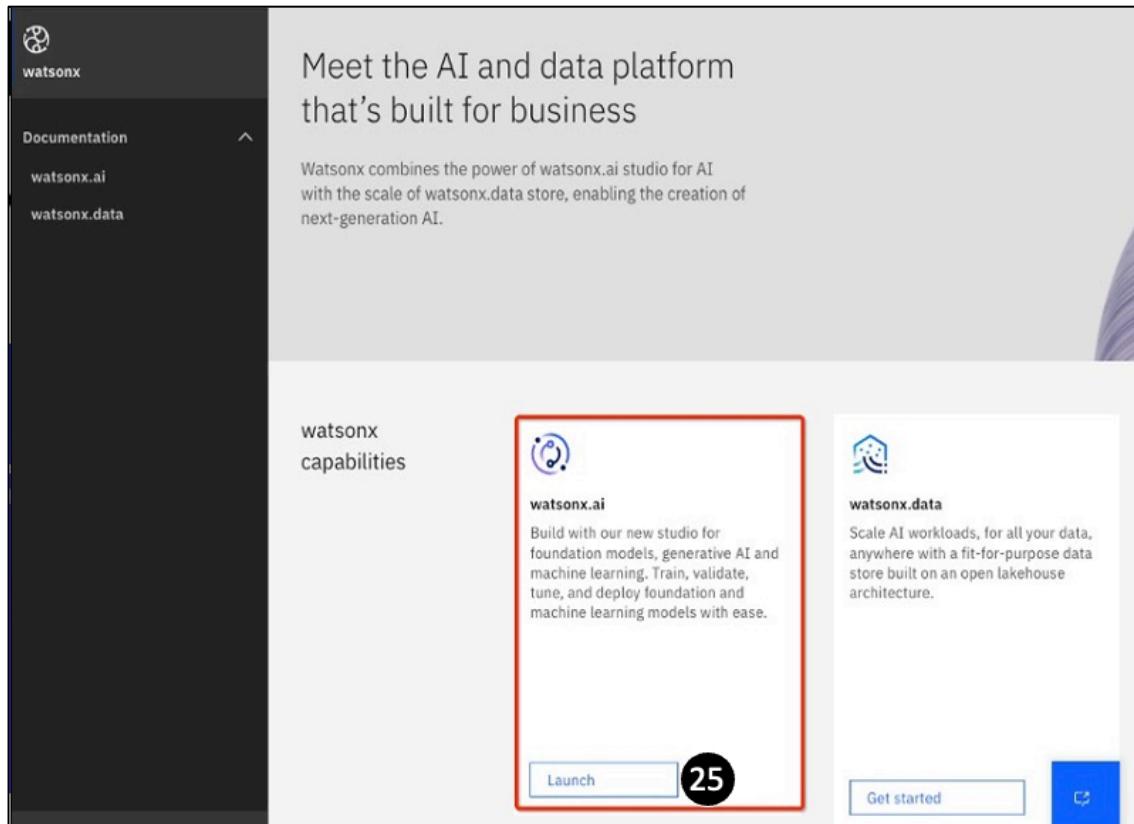
The screenshot shows the IBM Cloud dashboard. At the top, there's a navigation bar with 'IBM Cloud', a search icon, 'Catalog', 'Manage', and other icons. Below the navigation is a sidebar with various icons. The main area is titled 'Dashboard'. Under 'For you', there are three cards: 'Build' (Explore IBM Cloud with this selection of easy starter tutorials and services), 'Build with Watson' (Chatbots, insights, recognizers, and more. Explore the AI platform for business.), and 'Use Watson Assistant' (Watson Assistant lets you build conversational interfaces into any application, device, or channel.). To the right, a search bar says 'Search...'. Below it is a list of resource names: 446198 - IBM - SKI Hosting, IBM Canada, 1556061 - Intel, 2071092 - ITZ-SQUAD, 2636727 - itz-watsonx, 2701381 - itz-watsonx-2, 2701257 - itz-watsonx-3 (this item is highlighted with a red box and has a circled number '22e' next to it), 2701299 - itz-watsonx-4, 2701255 - itz-watsonx-5, and 3333366 - TechZone Outcomes.

23. In the search field, enter **watsonx**.

24. The search will return a list similar to the following. Click **watsonx**.

The screenshot shows the IBM Cloud catalog search results for 'watsonx'. The search bar at the top says 'watsonx' and has a circled number '23'. Below the search bar, it says '0 resource results found'. The 'Catalog Results' section shows three items: 'watsonx Service' (highlighted with a red box and circled number '24'), 'watsonx.data Service', and 'NeuralSeek Service'. Below the catalog results, there are two links: 'Search "watsonx" in Support Cases' and 'Search "watsonx" in Docs'.

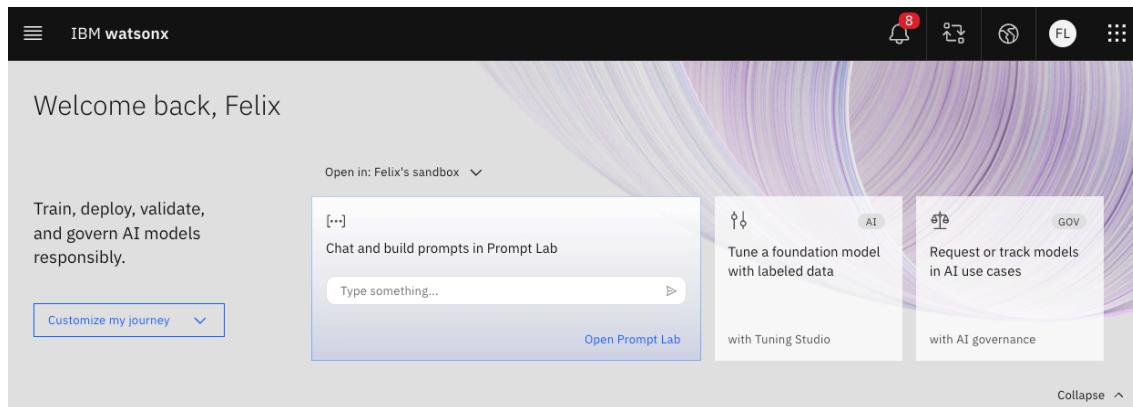
25. You will now see the AI and data platform page. Click **Launch** from the watsonx.ai tile.



26. If you do not see **Launch**, enter the following into the URL:

- o Dallas: <https://dataplatform.cloud.ibm.com>
- o Frankfurt: <https://eu-de.dataplatform.cloud.ibm.com>
- o London: <https://eu-gb.dataplatform.cloud.ibm.com>

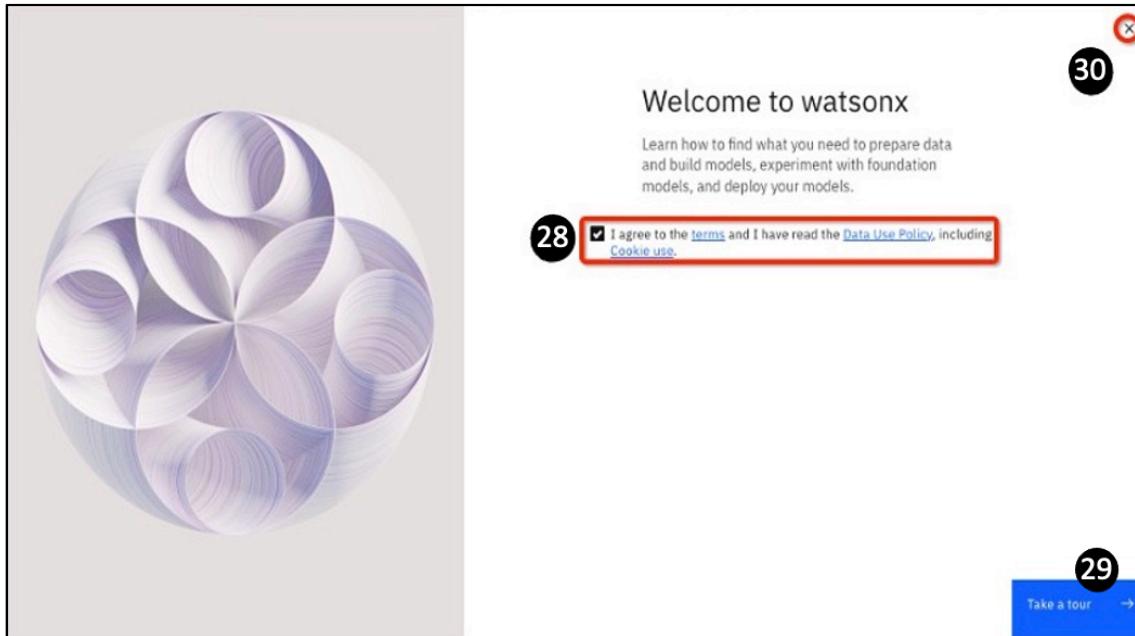
27. You are now in the watsonx.ai console and ready to begin this lab.



28. You might see the following page. Select the checkbox to agree to the terms.

29. You can click on the **Take a tour →** button on the bottom-right if you are interested in taking a tour of watsonx.ai.

30. For this lab, simply click on the **x** on the upper right to close this window.

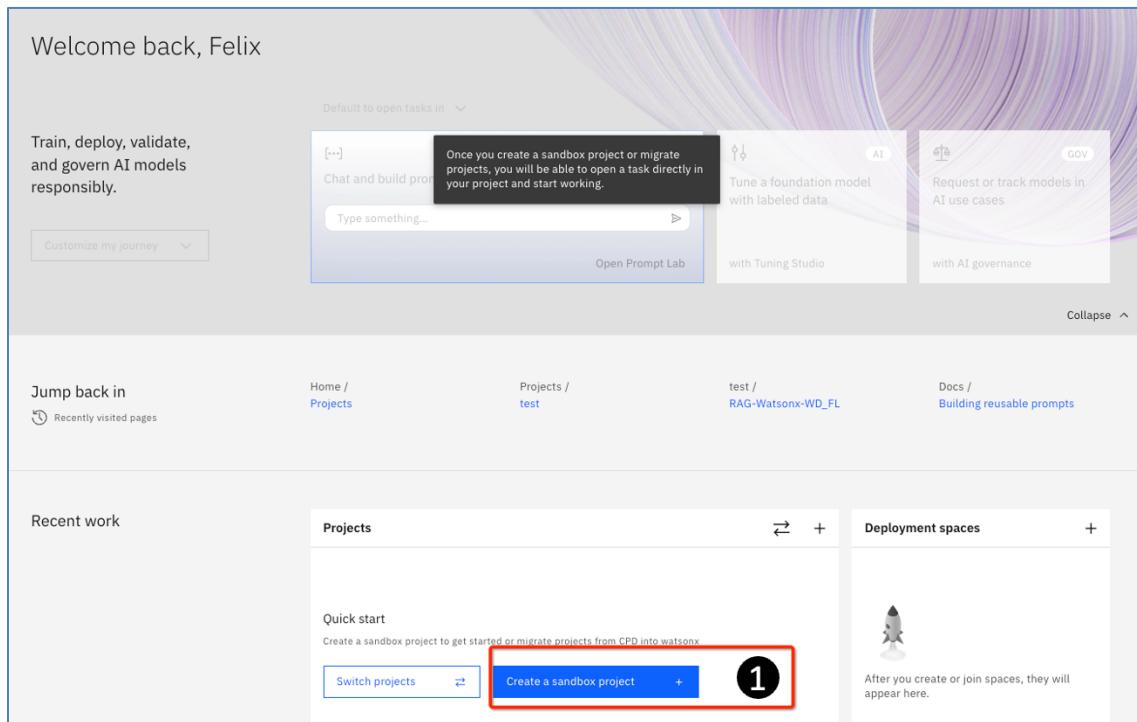


You are now ready to explore the watsonx.ai console.

## 5. Watsonx.ai console

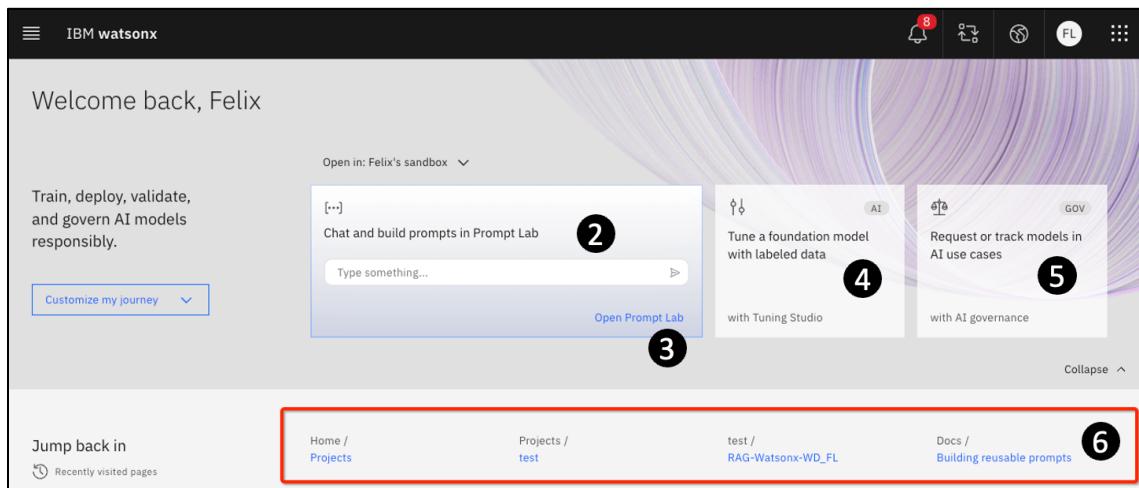
The watsonx.ai console contains various tiles. You must first create a project before performing any tasks in this lab.

1. Click the **Create a sandbox project** button in the **Projects** tile. This will create a project with all the necessary services for this lab. This can take tens of seconds.

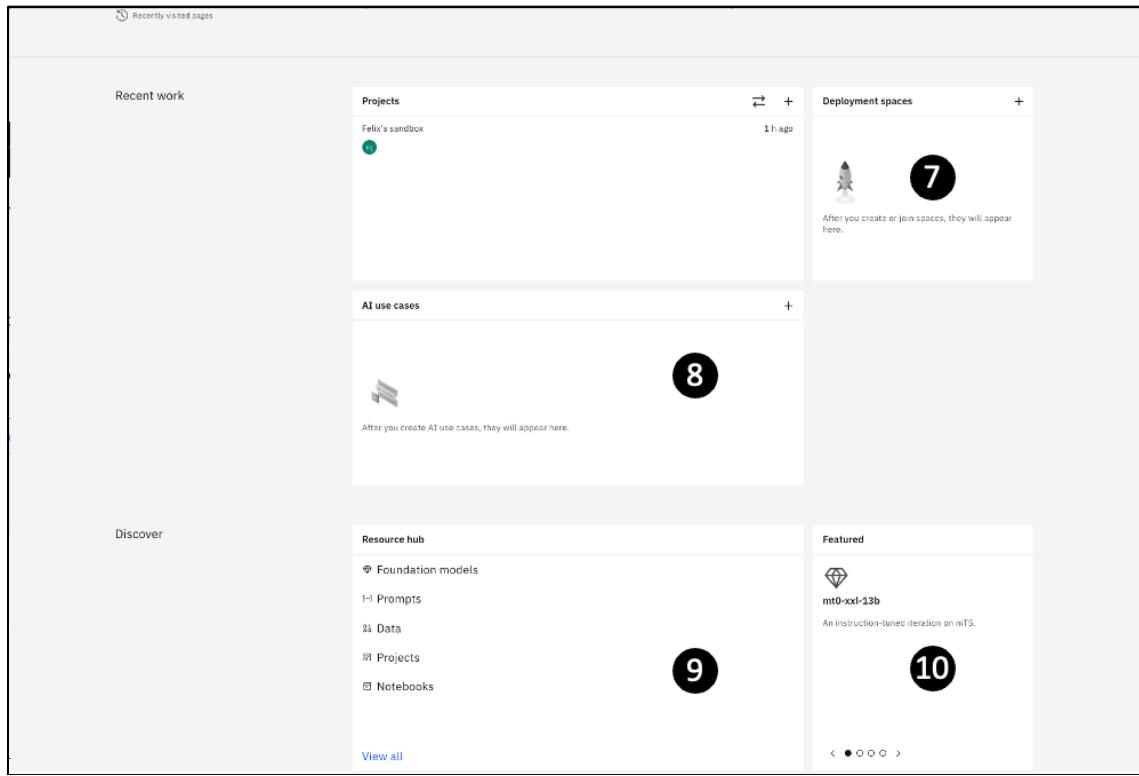


Here are the various regions of the console:

2. Allows you to use the watsonx.ai **Chat** feature. This is covered in another lab.
3. Takes you to the **Prompt Lab** page. You can perform various prompt engineering tasks: experiment and work with different models, test your prompts, and work on zero-shot, one-shot, or few-shot prompting. You can also change various parameters to control how conservative or creative the generative AI model will be. This is the focus of this lab.
4. Takes you to the **Tuning Studio**. This is covered in another L3 Lab.
5. Takes you to watsonx.governance features such as model tracking with AI use cases.
6. Quick links to recently visited pages.



7. Scrolling down further, this shows a list of **Deployment spaces** (required to deploy any prompt-tuned model).
8. List your **AI use cases** once you have created them.
9. Collection of samples – This is an important area if you are new to watsonx.ai and generative AI. Here you will find examples and useful information on:
  - a. Foundation models that are available to use in watsonx.ai.
  - b. Prompts
  - c. Data
  - d. Projects
  - e. Notebooks
10. Model highlights – watsonx.ai will highlight various foundation models. It also provides details of the intended usage of the model. For open models from Hugging Face, you can also link to additional information from Hugging Face.



In this lab, you will focus on the prompt lab (item 2 above). The Prompt Lab provides rich capabilities for working with prompts and different foundation models. This lab assumes that you are familiar with various generative AI and foundation model concepts covered in the following Your Learning assets:

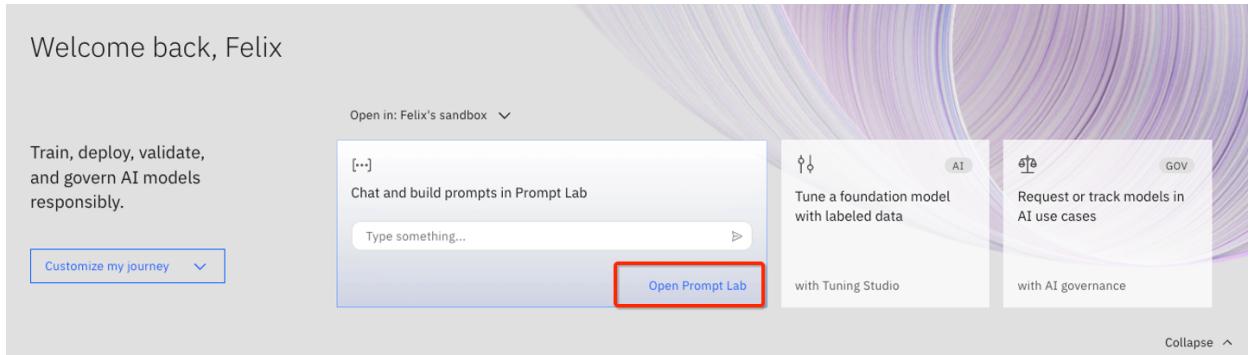
- [AI Fundamentals Level 1](#)
- [AI for Business Level 1](#)

Topics that will be covered include:

- Basic navigation of the Prompt Lab user interface
- Exploring different foundation models and their strengths/weaknesses
- Exploring simple zero-shot prompting and one-shot prompting
- Perform prompt engineering – changing the prompt input, or various parameters
- Create text and formatted output
- Various ways of saving your prompt information
- Creating a Jupyter notebook and running it

## 6. Watsonx.ai Prompt Lab

From the watsonx.ai console, click on **Open Prompt Lab**.



This brings you to the Prompt Lab.

### 6.1 Basic Navigation

The Prompt Lab provides many capabilities. This lab only examines a subset.

The window that opens is the default view of the Prompt Lab. There are 3 icons on the left. You can hover over each one to get some information about each:

- Provides a list of prompt samples – most useful for beginners
- […] List of saved prompts
- ⌚ History of prompts used in the current session

The other important areas of the Prompt Lab (looking at a subset for now) include:

1. Watsonx.ai provides AI guardrails. By default, it is off. You turn this on to prevent potential harmful input and output text (such as hate, abusive, or prejudiced wordings). You might have to start working with watsonx.ai before you can turn this on.
2. You might have to start working with watsonx.ai before you can turn this on.  
The ability to toggle between the Structured prompt or Freeform prompt editors.
  - a. **Chat:** If you are logging on Watsonx.ai for the first time this would be the default tab on the Prompt Lab page. This is not covered in this lab.

- b. **Structured:** The focus of this lab. Select this tab. This is recommended for beginners.
  - c. Experienced users may opt for Freeform prompting.
3. You can change to different foundation models by clicking on the drop-down menu.
4. Allows users to provide an instruction which will become part of your prompt. While it is optional, you should be careful what you place here because it consumes tokens.

Keep in mind that prompting is not the same as asking a question. A foundation model is not “answering” but rather generating output based on the input. Anything that is included can be used by the model.

5. You have the option to provide a sample input here.
6. This is where you provide the sample output (corresponding to the input from item 4).

Foundation models can sometimes be thought of as probability machines – they generate output based on what might be the best extension of the given input (prompt). However, unless a user is an expert in writing prompts, a foundation model may not understand the intent of the prompt or the type/structure desired for the output.

A user can “teach” a foundation model by providing a sample input and output (referred to as a “shot”). These instruct the model on how to best respond to a query/prompt. Types of shots include:

- Zero-shot prompting – no input/output sample provided
- One-shot prompting – a single input/output sample provided
- Few-shot prompting – multiple input/output samples provided

One-shot or few-shot prompting makes it easier to teach a model how to optimally respond. Zero-shot can still be effective when one has good skills in prompt engineering.

Notes:

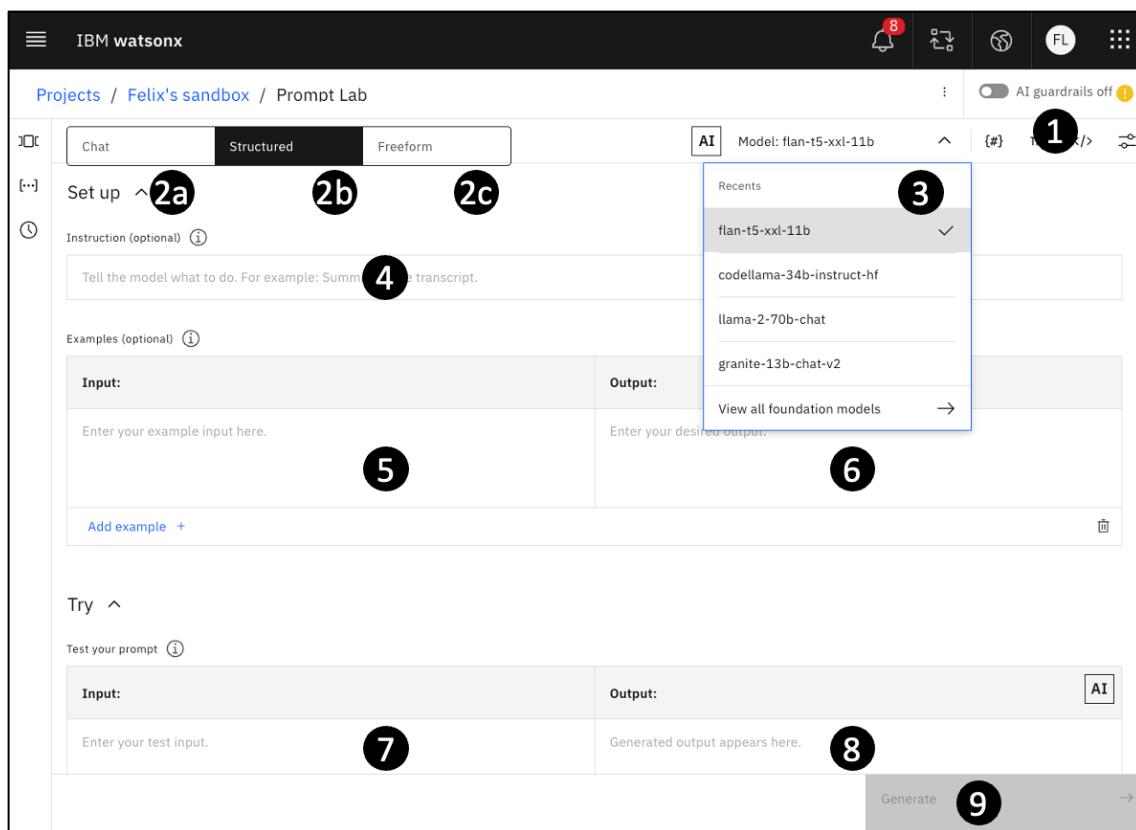
- Prompts are tokenized before being passed into a model, and typically foundation model usage costs are calculated based on the number of tokens. Using few-shot prompting frequently can become expensive.

- As a related concept, the larger the foundation model the more resources (cores and memory) and costs are required to operate them.

7. The Try section is where you enter your prompt/query into the foundation model.
8. This is where the generated output from a foundation model will be displayed.
9. Click the Generate button for the model to generate the output.

Other controls available will be discussed later in the lab.

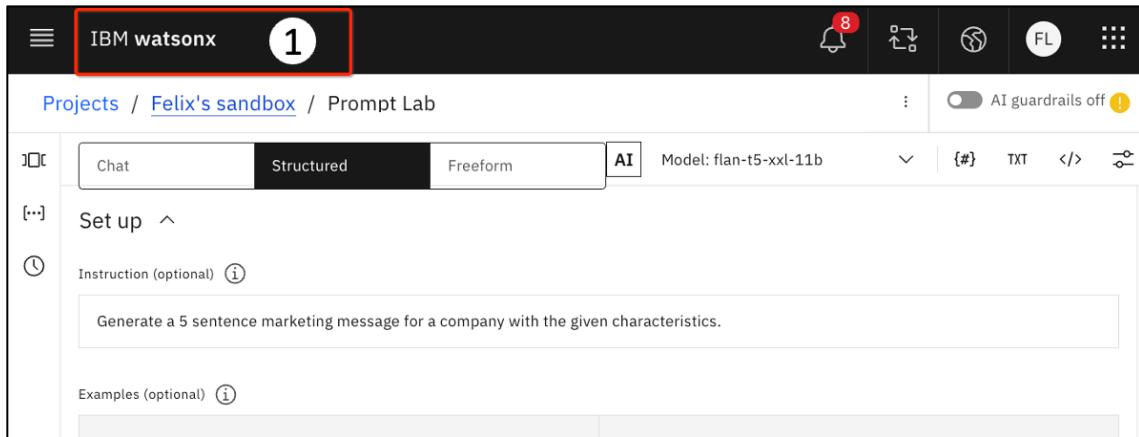
**Important note:** watsonx.ai (as of March 2024) opens up in the **Chat** tab – a new tab in addition to the **Structured** and **Freeform** tabs. This is not discussed in this lab (and the screen captures do not show it). Simply choose **Structured** or **Freeform** as directed by the lab instructions. **Chat** will be covered in a future update.



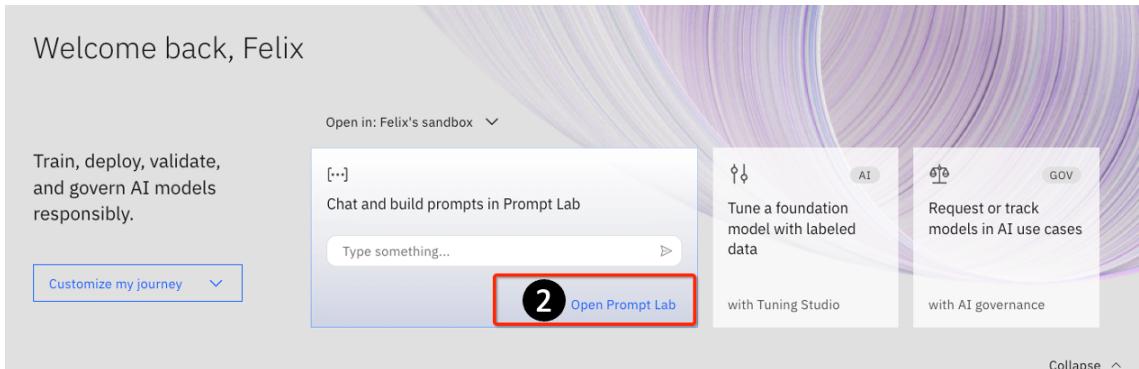
### 6.1.1 Open up a new Prompt Lab session

Many of the exercises in this lab begin with a step: Open a new Prompt Lab session. Here are the steps to do so.

1. Click on the **IBM Watsonx** in the upper left corner of the console.



2. You are back to the watsonx.ai welcome screen. Click **Open Prompt Lab**.



This takes you back to the Prompt Lab.

Note: you can certainly explore the Prompt Lab on your own; however, since watsonx.ai “remembers” (for example, if you pick a different model or set different inference parameters), exploring and testing outside of the scope of the lab might impact your prompt output. You can always do a reset by opening up a new Prompt Lab session.

## 6.2 Exploring different foundation models and their characteristics

IBM watsonx.ai is constantly updating the list of models available. The following models are available as of May 2024. Note that depending on the Geo region you reserved, you will get different lists. This list is also constantly changing.

 <b>granite-13b-chat-v2</b> The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks. Provider: IBM Type: InstructLab	 <b>mt0-xxl-13b</b> An instruction-tuned iteration on mT5. Provider: BigScience Type: Provided model	 <b>codellama-34b-instruct-hf</b> Code Llama is an AI model built on top of Llama 2, fine-tuned for generating and discussing code. Provider: Code Llama Type: Provided model
 <b>flan-t5-xxl-3b</b> A pretrained T5 - an encoder-decoder model pre-trained on a mixture of supervised / unsupervised tasks converted into a text-to-text format. Provider: Google Type: Provided model	 <b>flan-t5-xxl-11b</b> flan-t5-xxl is an 11 billion parameter model based on the Flan-T5 family. Provider: Google Type: Provided model	 <b>flan-ul2-20b</b> flan-ul2 is an encoder decoder model based on the T5 architecture and instruction-tuned using the Fine-tuned Language Net. Provider: Google Type: Provided model
 <b>merlinite-7b</b> Merlinite-7b is a Mistral-7b-derivative model trained with the LAB methodology, using Mixtral-8x7b-Instruct as a teacher model. Provider: Mistral AI, tuned by IBM Type: InstructLab	 <b>mixtral-8x7b-instruct-v01-q</b> Mixtral-8-7b-instruct-v01-gptq model is made with AutoGPTQ, which mainly leverages the quantization technique to 'compress' the model weights from FP16 to 4-bit INT and performs 'decompression' on-the-fly before... Provider: Mistral AI, tuned by IBM Type: Provided model	 <b>granite-13b-instruct-v2</b> The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks. Provider: IBM Type: Provided model
 <b>granite-20b-multilingual</b> The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks. Provider: IBM Type: InstructLab	 <b>granite-7b-lab</b> The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks. Provider: IBM Type: InstructLab	 <b>llama-2-13b-chat</b> Llama-2-13b-chat is an auto-regressive language model that uses an optimized transformer architecture. Provider: Meta Type: Provided model
 <b>llama-2-70b-chat</b> Llama-2-70b-chat is an auto-regressive language model that uses an optimized transformer architecture. Provider: Meta Type: Provided model	 <b>llama-3-70b-instruct</b> Llama-3-70b-instruct is an auto-regressive language model that uses an optimized transformer architecture. Provider: Meta Type: Provided model	 <b>llama-3-8b-instruct</b> Llama-3-8b-instruct is an auto-regressive language model that uses an optimized transformer architecture. Provider: Meta Type: Provided model
 <b>mixtral-8x7b-instruct-v01</b> The Mixtral-8x7B Large Language Model (LLM) is a pretrained generative Sparse Mixture of Experts. Provider: Mistral AI Type: Provided model		

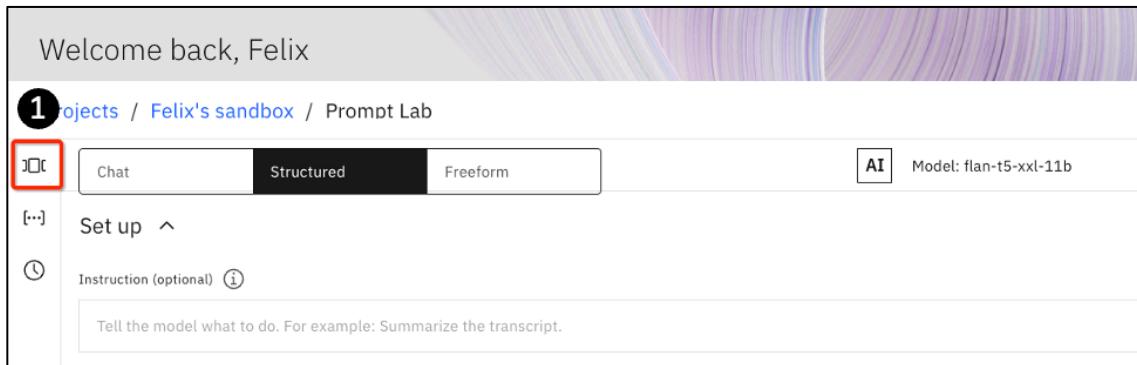
### Notes:

- The yellow icon  marks model(s) that are being deprecated. As of May 2024, the mixtral-8x7b-instruct-v01-q model is marked for deprecation. You will not be using this model in this lab.

- There are open-source models as well as IBM models. The granite models are created by IBM
  - granite-13b-chat-v2
  - granite-13b-instruct-v2
  - granite-20b-multilingual
  - granite-7b-lab

You will now work with various models and prompts.

1. Click on the  icon to slide open a list of sample prompts on the left (1).



Watsonx.ai provides sample prompts grouped into categories like:

- Summarization
- Classification
- Generation
- Extraction
- Question Answering
- Code
- Translation

These are the 7 main use cases for generative AI (there is no sample prompt yet for the Retrieval Augmented Generation use case). For this first exercise, you will look at a **Marketing email generation** example in the **Summarization** section.

2. Scroll down on the **Sample prompts** list to find the **Summarization** section. You can also scroll up and down and see what other examples are provided with watsonx.ai

The screenshot shows the Watsonx.ai Prompt Lab interface. On the left, there's a sidebar with 'Sample prompts' and a message about improving prompts. The main area has tabs for Chat, Structured, Freeform, and AI (selected). The Model dropdown shows 'flan-t5-xxl-11b'. The 'Set up' section contains an instruction: 'Generate a 5 sentence marketing message for a company with the given characteristics.' Below it, a box says 'More samples available when you scroll down'. The 'Try' section has a 'Test your prompt' input field.

3. Select the **Marketing email generation** sample prompt. Once you select a sample prompt, the selected sample is highlighted in a **blue** rectangle.
4. Notice that the **flan-t5-xxl-11b** model was automatically selected. Watsonx.ai selects the model that is most likely to provide the best performance. However, this is not a guarantee, and in this part of the lab, you will explore different models and observe their completion.

The screenshot shows the Watsonx.ai Prompt Lab interface. The 'Marketing email generation' sample prompt is highlighted with a red box (step 3). The 'Model' dropdown at the top right is also highlighted with a red box and shows 'flan-t5-xxl-11b' (step 4). The rest of the interface is similar to the previous screenshot.

This prompt asks the foundation model to generate a 5-sentences marketing message based on the following characteristics (click on the **Details** field in the Try section to expand the box and see the full text of this example).

Details
<p>Characteristics:</p> <p>Company - Golden Bank</p> <p>Offer includes - no fees, 2% interest rate, no minimum balance</p> <p>Tone - informative</p> <p>Response requested - click the link</p> <p>End date - July 15</p>

If you cannot find this prompt example, or if the contents have changed, you can enter:

- For **Instruction**

Generate a 5 sentence marketing message for a company with the given characteristics.

- For **Details** under the **Try** section

Characteristics:

Company - Golden Bank

Offer includes - no fees, 2% interest rate, no minimum balance

Tone - informative

Response requested - click the link

End date - July 15

5. Click **Generate** to see the email this foundation model will generate based on the prompt and its details. The following output is generated:

Email
<p>We are offering a new checking account with no fees and a higher interest rate. If you are interested, click on the link and fill out the application. The deadline is July 15, so hurry! Contact us or visit us at our website if you have any questions.</p>

This does not look to be a very good draft as there are missing pieces. There is no mention of Golden Bank, the 2% interest rate offer, and no minimum balance. It does inform the reader of the pending deadline of July 15. Pointing to the company website is not a bad idea as the bank would have more details there.

Note the following:

- In this exercise, you did not pass in any sample input/output. This is an example of zero-shot prompting.
- This LLM did not “hallucinate” beyond reasonable information:
  - The completion says “higher interest rate” – the 2% would likely qualify as the bank would not have mentioned it in the instruction. However, it would have been better to explicitly mention the 2%.
  - The LLM did not make up any phone number or URL for the customer to use – which is good as no explicit instructions were given on those.
- All the text in the Details section is part of the prompt. Look at the bottom of the console (to the left of the Generate button) to see the information:

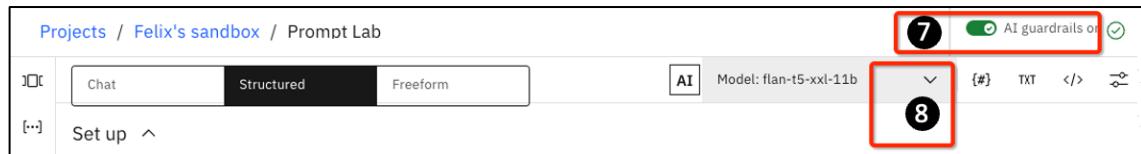
```
Stop reason: End of sequence token encountered
Tokens: 62 input + 57 generated = 119 out of 4096 | Seed: 111
Time: 2.1 seconds
```

- You might see a different execution time than 2.1 seconds. These are not dedicated environments, so performance will likely vary.
  - In this exercise, the input (including the instruction) consumed 62 tokens while the output consumed another 57 generated tokens.
  - A token is an abstract entity that the model uses to represent input and output. In this example, the instructions and details add up to fewer than 40 words. This means that **the number of words does not map 1-1 to the number of tokens**. The ratio of words to tokens varies anywhere from 1:1.3 to roughly 1:3<sup>1</sup> (depends on the length of words, keep in mind that punctuations etc. also consume tokens). It is also model-dependent.
- Different models have different token limits. For example, the limit for the mpt-7b-instruct2 model is 2048 tokens. For the **granite-13b-instruct-v2** has an 8192 limit.

---

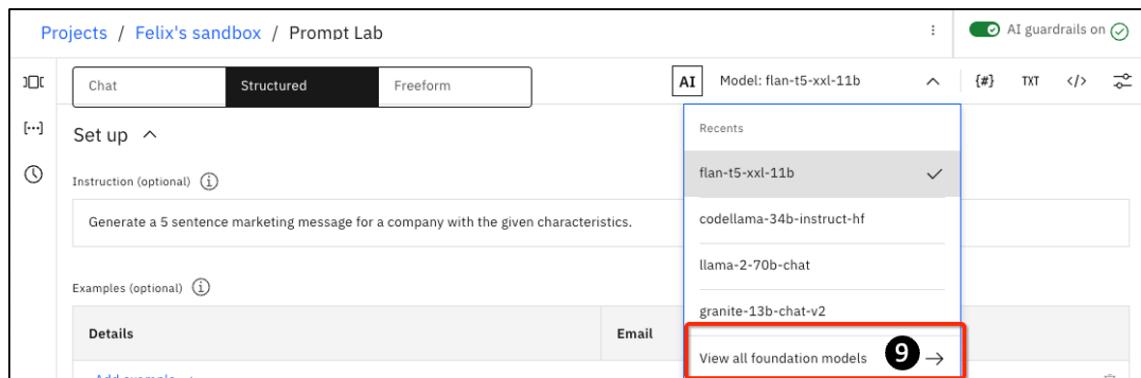
<sup>1</sup> See <https://ec.yourlearning.ibm.com/w3/join/event/10388387> (IBMer).

6. There is a monthly limit on the total number of tokens available to you, depending on your subscription.
7. Turn on **API guardrails**. This provides checking for Hate, Abuse, Profanity (HAP) content. You will see it in action later.
8. Now you will try a different foundation model. Click the **model drop-down box** on the right.



If this is your first time using the watsonx.ai Prompt Lab, you will see only the flan-t5- xxl-11b model listed. If you have used other models, you will see them in the drop-down.

9. Regardless of what is shown in the drop-down, click **View all foundation models**.



This will bring up a list of models as shown below:

<b>granite-13b-chat-v2</b> <small>The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks.</small> Provider: IBM	Type: InstructLab	<b>mt0-xxl-13b</b> <small>An instruction-tuned iteration on mT5.</small> Provider: BigScience	Type: Provided model
<b>flan-t5-xxl-11b</b> <small>flan-t5-xxl is an 11 billion parameter model based on the Flan-T5 family.</small> Provider: Google	Type: Provided model	<b>flan-ul2-20b</b> <small>flan-ul2 is an encoder-decoder model based on the T5 architecture and instruction-tuned using the Fine-tuned Language Net.</small> Provider: Google	Type: Provided model
<b>granite-13b-instruct-v2</b> <small>The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks.</small> Provider: IBM	Type: Provided model	<b>granite-20b-multilingual</b> <small>The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks.</small> Provider: IBM	Type: InstructLab
<b>llama-2-70b-chat</b> <small>Llama-2-70b-chat is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta	Type: Provided model	<b>llama-3-70b-instruct</b> <small>Llama-3-70b-instruct is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta	Type: Provided model
<b>codellama-34b-instruct-hf</b> <small>Code Llama is an AI model built on top of Llama 2, fine-tuned for generating and discussing code.</small> Provider: Code Llama	Type: Provided model	<b>flan-t5-xl-3b</b> <small>A pretrained T5 - an encoder-decoder model pre-trained on a mixture of supervised / unsupervised tasks converted into a text-to-text format.</small> Provider: Google	Type: Provided model
<b>merlinite-7b</b> <small>Merlinite-7b is a Mistral-7b-derivative model trained with the LAB methodology, using Mistral-8x7b-Instruct as a teacher model.</small> Provider: Mistral AI, tuned by IBM	Type: InstructLab	<b>mixtral-8x7b-instruct-v01-q</b> <small>Mixtral-8-7b-instruct-v01-qpt model is made with AutoGPTQ, which mainly leverages the quantization technique to 'compress' the model weights from FP16 to 4... pT and performs 'decompression' on-the-fly before inference.</small> Provider: Mistral AI, tuned by IBM	Type: Provided model
<b>llama-2-13b-chat</b> <small>Llama-2-13b-chat is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta	Type: Provided model	<b>llama-2-13b-chat</b> <small>Llama-2-13b-chat is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta	Type: Provided model
<b>llama-3-8b-instruct</b> <small>Llama-3-8b-instruct is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta	Type: Provided model	<b>mixtral-8x7b-instruct-v01</b> <small>The Mixtral-8x7B Large Language Model (LLM) is a pretrained generative Sparse Mixture of Experts.</small> Provider: Mistral AI	Type: Provided model

## Notes:

- IBM will update this set of models frequently. You may not see exactly this list.
- The provider for each model is listed. Note the 4 models currently provided by IBM (as of May 2024 – this collection will also change over time):
  - **granite-13b-instruct-v2**
  - **granite-13b chant-v2**
  - **granite-20b-multilingual**
  - **granite-7b-lab**
- Model(s) marked with a yellow sign are being deprecated. If you (or a client you work with) are using these, you should look into moving to a different mode

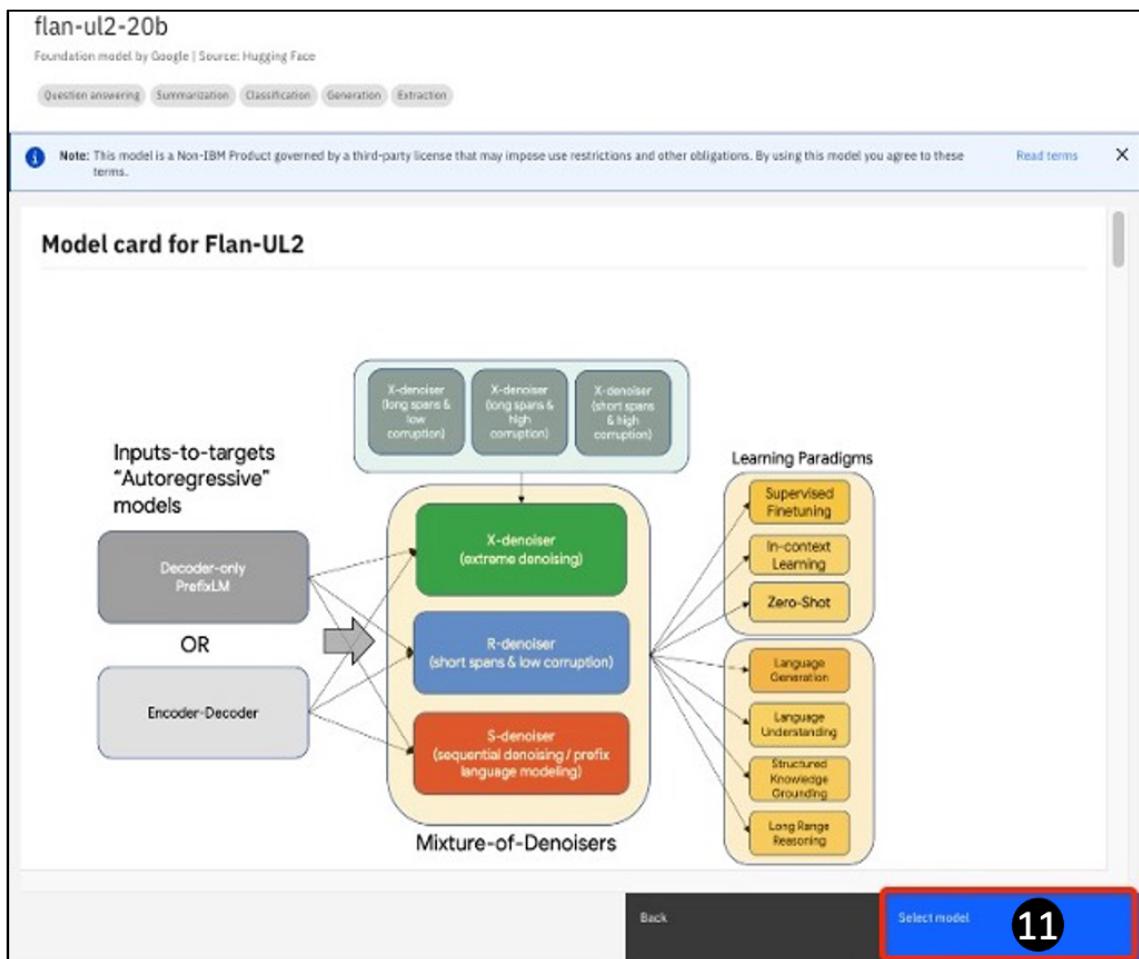
10. In the previous example, you were using the flan-t5-xxl-11b model. This time, select the flan-ul2-20b model. Note that this is a bigger model with 20 billion parameters versus the flan-t5-xxl-11b model which has 11 billion parameters.

Select a foundation model							
Select a model that best fits your needs. All models support English text. Check the model information for other supported languages.							
granite-13b-chat-v2 <small>The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks.</small> Provider: IBM Type: InstructLab	mt0-xxl-13b <small>An instruction-tuned iteration on mT5.</small> Provider: BigScience Type: Provided model	codellama-34b-instruct-hf <small>Code Llama is an AI model built on top of Llama 2, fine-tuned for generating and discussing code.</small> Provider: Code Llama Type: Provided model	flan-t5-xxl-3b <small>A pretrained T5 - an encoder-decoder model pre-trained on a mixture of Supervised / unsupervised tasks converted into a text-to-text format.</small> Provider: Google Type: Provided model				
flan-t5-xxl-11b <small>flan-t5-xxl is an 11 billion parameter model based on the Flan-T5 family.</small> Provider: Google Type: Provided model	flan-ul2-20b <small>flan-ul2 is an encoder-decoder model based on the T5 architecture and instruction-tuned using the Fine-tuned Language Net.</small> Provider: Google Type: Provided model	merlinite-7b <small>Merlinite-7b is a Mistral-7b-derivative model trained with the LAB methodology, using Mistral-8x7b-Instruct as a teacher model.</small> Provider: Mistral AI, tuned by IBM Type: InstructLab	mixtral-8x7b-instruct-v01-q <small>Mixtral-8-7b-instruct-v01-qptq model is made with AutoGPTQ, which mainly rearranges the quantization technique to ‘compress’ the model weights from FP16 to 4-bit and performs ‘decompression’ on-the-fly before inference.</small> Provider: Mistral AI, tuned by IBM Type: Provided model				
granite-13b-instruct-v2 <small>The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks.</small> Provider: IBM Type: Provided model	granite-20b-multilingual <small>The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks.</small> Provider: IBM Type: InstructLab	granite-7b-lab <small>The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative tasks.</small> Provider: IBM Type: InstructLab	llama-2-13b-chat <small>Llama-2-13b-chat is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta Type: Provided model				
llama-2-70b-chat <small>Llama-2-70b-chat is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta Type: Provided model	llama-3-70b-instruct <small>Llama-3-70b-instruct is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta Type: Provided model	llama-3-8b-instruct <small>Llama-3-8b-instruct is an auto-regressive language model that uses an optimized transformer architecture.</small> Provider: Meta Type: Provided model	mixtral-8x7b-instruct-v01 <small>The Mixtral-8x7B Large Language Model (LLM) is a pretrained generative Sparse Mixture of Experts.</small> Provider: Mistral AI Type: Provided model				

When you select a model from this screen, watsonx.ai provides additional information about the model through a model card.

Browse the page to learn more about this model. Notice that this is a 3<sup>rd</sup> party model. You can scroll down to find out how it was trained, and how it might have already been tuned. This information may be of more use and interest to experienced data scientists.

11. Click **Select model** to use this model.



12. Verify you are now using the flan-ul2-20b model, then click **Generate**. You will see the following:

The screenshot shows the AI interface setup. The "Model" dropdown menu at the top is set to "flan-ul2-20b" (highlighted with a red box and labeled 12). A tooltip above the dropdown reads "Ensure that you are using The flan-ul2-20b model". The "Chat" tab is selected. The "Instruction (optional)" field contains the prompt: "Generate a 5 sentence marketing message for a company with the given characteristics." Below it, the "Examples (optional)" section has a "Details" table with a single row: "Characteristics: Company - Golden Bank...". The "Email" field below the table contains the generated marketing message: "We'd like to tell you more about Golden Bank and our great new checking account. Its free, requires no minimum balance and has no monthly maintenance fees. Currently we're offering a 2% interest rate on all balances. To take advantage of this offer, click the link below or call us at 713-852-1411."

The completion is as follows:

We'd like to tell you more about Golden Bank and our great new checking account. Its free, requires no minimum balance and has no monthly maintenance fees. Currently we're offering a 2% interest rate on all balances. To take advantage of this offer, click the link below or call us at 713-852-1411.

This is an interesting completion. While part of the email is similar to the output of flan-t5-xxl-11b, this model has shown a common flaw of generative AI: hallucination. Note that the input does not have the phone number 713-852-1411 – the model has completely made up that part of the email! The LLM also fails to mention the important July 15 deadline.

Here is a simple example where a larger foundation model performs worse than a smaller one.

13. Now repeat Steps 5-7, this time selecting the **mt0-xxl-13b** model. Then repeat Step 9 to ask the Prompt Lab to generate a new response using this new model. You will see the following completion:

**Email**

Pleased to inform you that Golden Bank Ltd. has been preparing special offer for the New Clients. It promises no fees, 2% interest rate and no minimum balance on any account open during your current account until July 15, 2016.

This model also hallucinated in its completion – coming up with the year 2016 (perhaps the model was trained with data up to 2026). There is also a statement for “New Client” which is not explicit in the input. However, compared to the flan-ul2-20b model this is a much better output. If you take our the 2016 year this is a very good email.

14. Try the prompt with one more model – **llama-2-70b-chat** (the largest model in watsonx.ai as of 1Q24). Follow Steps 7-10 to select the model, and then click **Generate**. You will see the following output:

## Email

Subject Line- Open Your Dream Account Today! No Fees & High Interest Rates  
Guaranteed at GolenBank ([link](#)) => Link will be included in actual email campaigns  
but removed here to avoid phishing attempts



This is another example where a larger model is not providing better completion. It can be creative with wordings such as “Dream Account” but it does not provide some key information such as the 2% interest. It also gets the name of the bank wrong (GolenBank versus Golden Bank).

## Section summary

- In this part of the lab, you became familiar with picking different models.
- You learned how to find out more detailed information about various models.
- Depending on the task, different models provide different results. It is important to experiment with different models.
- Larger models do not necessarily perform better than smaller models. In fact, larger models may be more inclined to be “creative” (if the underlying conditions and parameter settings like temperature and greedy/sampling mode remain the same).
- This is a zero-shot exercise (you provide no input/output example) with default settings.
- One should always test before using the foundation model too quickly.

## 6.3 Prompt Engineering – updating a zero-shot prompt

The Prompt Lab is the place for you to try out prompt engineering – the “art” of modifying your input to produce the best result.

The last section demonstrated that given a list of characteristics, different models will generate different marketing emails with various degrees of hallucination. In this part of the lab, you will modify the prompt and look at how different models perform the completion.

Note: you will be looking into model operations which is at a token level. While everyone sees LLMs generating the next word, they are really generating the next token. For high-level discussions, you can assume a token equals a word, but that is not the case (long words can have multiple tokens, punctuation marks are tokens, and so on). In this lab, we will use these words interchangeably and as synonyms unless otherwise noted and keep references to words and tokens at a high level

1. Open a new Prompt Lab session and select the  icon from the left to bring up the list of sample prompts. Click the **Marketing email generation** sample and switch to using the **flan-ul2-20b** model.

The prompt should read as follows:

- For Instruction

Generate a 5 sentence marketing message for a company with the given characteristics.

- For Details under the Try section

Characteristics:  
Company - Golden Bank  
Offer includes - no fees, 2% interest rate, no minimum balance  
Tone - informative  
Response requested - click the link  
End date - July 15

If you cannot find this prompt example or if the text content has changed, please use the text provided above to ensure you will see the same completion as shown in this part.

Click **Generate** to get the following output email:

## Email

We'd like to tell you more about Golden Bank and our great new checking account. It's free, requires no minimum balance and has no monthly maintenance fees. Currently we're offering a 2% interest rate on all balances. To take advantage of this offer, click the link below or call us at 713-852-1411.

2. Add the following text to the **Details** text box **Do not add any additional information.**

Details
Characteristics: Company - Golden Bank Offer includes - no fees, 2% interest rate, no minimum balance Tone - informative Response requested - click the link End date - July 15 <b>Do not add any additional information.</b> <span style="border: 1px solid red; padding: 2px;">2</span>

3. Click **Generate** and you get the following output:

Email
We've got great news for you! Golden Bank is now offering accounts with no fees, a 2% interest rate and no minimum balance. Don't miss out on this deal; it won't last long. Click here to find out more: <a href="http://www.goldenbank.com/feefreepromotion.htm">http://www.goldenbank.com/feefreepromotion.htm</a>

This is again not a good output – the LLM did not provide a phone number but instead hallucinated a website URL.

4. Go back to the Details **input** field. Instead of **Do not add any additional information**, be more explicit and use: **Do not make up any phone number or website.**

Try ^

Test your prompt [\(i\)](#)

**Details**

Characteristics:

Company - Golden Bank

Offer includes - no fees, 2% interest rate, no minimum balance

Tone - informative

Response requested - click the link

End date - July 15

Do not make up any phone number or website.

4

- Now when you click **Generate**, you will see the following output:

Email

We've got great news for you! Golden Bank is now offering accounts with no fees, a 2% interest rate and the ability to open an account with just \$500. To sign up, click the link below. Offers are available until July 15th.

The made-up website is gone. However, the generated output still includes a hallucination. Remember, generative AI models are not answering questions but are generating the best probable text based on input. The larger the model, the more creative the completion tends to be. In this case, the model calculates from the prompt that it is undesirable to generate a phone number or a website, but it probably was trained with finance data that made it determine that an account with a minimum balance of \$500 is of high probability and added it to its output.

- Because flan-ul2-20b is creative, you should try something very explicit. This time, update the input text with: "Do not include any other information not provided above".

Try ^

Test your prompt [\(i\)](#)

**Details**

Characteristics:

Company - Golden Bank

Offer includes - no fees, 2% interest rate, no minimum balance

Tone - informative

Response requested - click the link

End date - July 15

6

Do not include any other information not provided above.

Now click **Generate**, and you will see the following output:

Email

We've got great news for you! Golden Bank is now offering accounts with no fees, a 2% interest rate and no minimum balance. Don't miss out-- sign up today by clicking the link below. Offer ends July 15.

This now is the proper email with no hallucination.

7. Now let's try this with the llama-2-70b-chat model. We saw what it would do with the base prompt (it hallucinated and got the name of the bank wrong). Now run this updated prompt and you will see the following output:

### Email

is from an employee of this bank branch and being sent to customers who have used their services in at least one transaction within past two months but are currently using another savings account provider (competitor). 40-63 years old on average medium income earners living paycheck by check or just getting into retirement planning stages that haven't chosen where yet so still open minded about different products available right now without too much brand loyalty involved making decisions easier when presented rational arguments back up its claims/promises through factual data points such as those highlights mentioned earlier would help make decision process smoother overall! We want these particular demographics because they tend lean towards stability over flashy advertising gimmicks; our product offer speaks itself based off benefits alone rather than relying heavily on promotions which often seen untrustworthiness coming from big named institutions lately anyway according recent news articles circulating online communities social medias outlets

You might not have the exact output, but it should still be very verbose and similar to the above completion.

This is a great surprise. Instead of reigning in the model, it has become loquacious (talkative) and at times incomprehensible. This model's generated email is not close to what you want to send. It talks about **40-63 years old on average medium income, rational arguments to back up its claims/promises**, and many other points not provided in the input, and it is missing key information such as the July 15 date. In short, the instruction that worked with flan-ul2-20b simply does not work here. The LLM started its completion in the middle of a sentence and it also did not finish – probably due to the limitation of output tokens (you will work with this in the next section of the lab).

Note: The llama-2-70b-chat model is huge and can be more unpredictable. You might not see exactly this output. You should still get a much smaller output than the previous one.

## 8. You can experiment further.

- Try changing the prompt to get a feel for how the flan-ul2-20b model may respond to different prompt inputs. If you change the wording above, how does the model respond?
- Try other models besides flan-ul2-20b. How do they respond to prompts that worked well with the flan-ul2-20b model?
- Which model seems to have the most difficult time responding to your prompt with the proper response?

## Section summary

- In this section, you learned that even with zero-shot prompting, it is possible to modify the prompt to get a better response from foundation models.
- Keep in mind that the model is not explicitly “answering” a question or a query, but is looking to extend/generate a response based on the input. This means that you often need to help the model to understand how your input is structured.
- Different models may require different prompts. For larger and more “creative” models, the instruction may need to be very explicit.
- It is useful to look into the specifics of a model. As the name implies, llama-2-70b-chat can be very “talkative” and may be better suited to chat-type use cases versus something like email generation.
- As an aside, models such as ChatGPT have been heavily optimized. It has learned from a lot more examples than most general Large Language Models (a type of foundation model) and has also been tuned with methods such as Reinforcement Learning from Human Feedback (RLHF) to better “understand” human instructions. They are more expensive to run and better able to interpret general questions, but they still can hallucinate.
- Models such as flan-ul2-20b and flan-t5-xxl-11b have not been optimized that way and will need good prompts and few-shot prompting (you will do this later in this lab).
- It is tempting to think that larger models are “smarter”. There is certainly some truth in that, but these models can also be too creative. As you have seen you may have to be very explicit to get the output you are looking for. Note that creativity is not necessarily an issue. This is one of the attractions of generative AI and foundation models. But if you are not trying to generate creative content (as is the case in this example – you want your email to mention the specific points and not add fictitious content), you might want to consider a smaller (and less creative) model or provide stricter prompts to rein in the models.

## 6.4 Prompt Engineering - modifying parameters

In this part of the lab, you will look at how to modify some of the model's inference (runtime) parameters and see how they affect the output of different models.

1. Click on the **sample** icon (DOC) from the left to bring up the list of sample prompts. Select the **Marketing email generation** sample from the **Generate** section. Use the **flan-ul2-20b** model if not already specified.

The prompt should read as follows:

- For Instruction

Generate a 5 sentence marketing message for a company with the given characteristics.

- For Details under the Try section

Characteristics:

Company - Golden Bank

Offer includes - no fees, 2% interest rate, no minimum balance

Tone - informative

Response requested - click the link

End date - July 15

If you cannot find this prompt example or if the text content has changed, please use the text provided above to ensure you will see the same completion as shown in this part.

You have seen that the flan-ul2-20b model will hallucinate and generate a phantom phone number: 713-852-1411 (see Step 11 of Section 6.2). Click Generate to see that same output again.

2. Recall from Section 6.3 that you can modify the prompt (to a degree) so that the model will not generate any false phone numbers or URLs. In this part of the lab, you will change the prompt a bit to see the effect of changing the parameters on a completion. Go to the Details text field and add the following to the original prompt: **Do not add any additional information** and click **Generate**.

## Details

Characteristics:

Company - Golden Bank

Offer includes - no fees, 2% interest rate, no minimum balance

Tone - informative

Response requested - click the link

End date - July 15

2

Do not add any additional information.

You will see the following output:

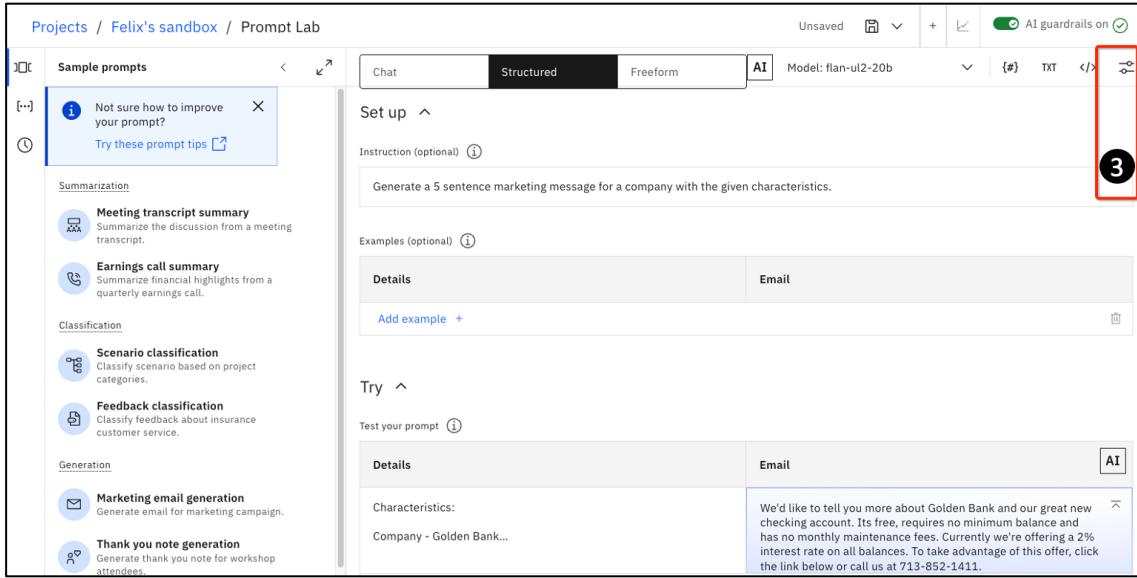
## Email

We've got great news for you! Golden Bank is now offering accounts with no fees, a 2% interest rate and no minimum balance. Don't miss out on this deal; it won't last long. Click here to find out more:

<http://www.goldenbank.com/feefreepromotion.htm>

As can be seen, the model has made up a website (as it did before).

3. Now, we will modify some inference parameters to see the impact on the completion instead of further changing the prompt. On the console's upper right-hand corner, click the  icon.



4. This opens a window that contains a list of parameters that you can modify. While you can change the model (using the model list drop-down), all models have the same inference parameters. Let's look at the different runtime parameters that you can work with.

The inference parameters you can adjust include:

5. **Greedy** vs. **Sampling** Decoding – This is how the model chooses the tokens for output. In Greedy mode, the model selects the highest probability tokens at every step of decoding, and a model is less creative in that mode. In the Sampling mode, the model tends to be more creative and random, but there is a larger risk that the output may be nonsensical.

The first 4 parameters below (b through e) are only visible when in Sampling mode.

6. **Temperature** – a floating point number ranging from 0.0 (which causes the model to operate in the same manner as if Greedy decoding was selected) and 2.00 (which is maximum creativity).

The higher the value of Temperature, the more “creative” the model would be.

The default value is typically 0.7 but is set to 0.8 in this case. This means the model is allowed certain creativity.

7. **Top P** (nucleus sampling) – a floating point number ranging from 0.0 to 1.0. At a high level, when generating a completion, a model calculates the probability of the next word based on all the previous words. Top P decides whether the

model will always pick the most likely outcome, or allow more randomness for the next words.

Top P sampling chooses from the smallest possible set of “next” words whose cumulative probability exceeds the probability p. This way, the number of words in the set can dynamically increase and decrease according to the next word probability distribution. Take, for example, a Top P value of 0.8.

This means that:

- If the probability of the most likely next word is  $> 0.8$ , then the model will return that word as the completion.
- If the probability is  $\leq 0.8$ , then the model will look at the next most likely word and sum up the probabilities. If the sum is now  $> 0.8$ , the model will choose randomly from between the top 2 words as the completion.
- If the sum is still  $\leq 0.8$ . The model will look at the third most likely word, sum the probabilities and so on. It will continue until it finds a candidate list of words whose probabilities sum up to be  $> 0.8$ . The model will select its next output word randomly from this list.

This means the higher the value of Top P, the larger the candidate list of words and so the more random the outcome would be. At high values of P, the model is potentially including a lot of candidate words of low probability, making the outcome more “creative”.

8. **Top K** – an integer from a range from 1 to 100. Unlike Top P, Top K does not look at the probability of the words. Instead, a model simply chooses from the Top K most likely words to be the output. For example:

- **Top K = 3**, the model will choose randomly among the top 3 most likely next words.
- **Top K = 5**, the model will choose randomly among the top 5 most likely next words.

The higher the value of Top K, the more random the outcome would be.

9. **Random seed** – is an integer in the range 1 to 4,294,967,295

- In the Sampling mode, you can generate different outputs from the model by using a different Random seed value each time, or by not specifying any value.
- If you do not change it, you should get back the same results.
- For the best result of the labs, do not change the Random seed value as that will create different results. Quite simply, a Random seed is helpful for the replicability of experiments.

**10. Repetition penalty** – a value between 1 and 2 (a setting of 1 allows repetition, and 2 prohibits it). This setting is used to counteract a model’s tendency to repeat the prompt text verbatim or be stuck in a loop in generating the same output.

**11. Stop sequences** – sequences of characters (can be text, carriage return, etc.). When the model encounters one of the sequences, it will stop generating output (the sequence itself will be included in the output). For example, suppose a model gave you the following output:

We are thrilled that you could make it to our workshop and hackathon today. It was great to meet everyone and we hope you enjoyed yourself. Thanks again for your participation and have a great weekend! Thank you for your participation. Thank you for being on time.

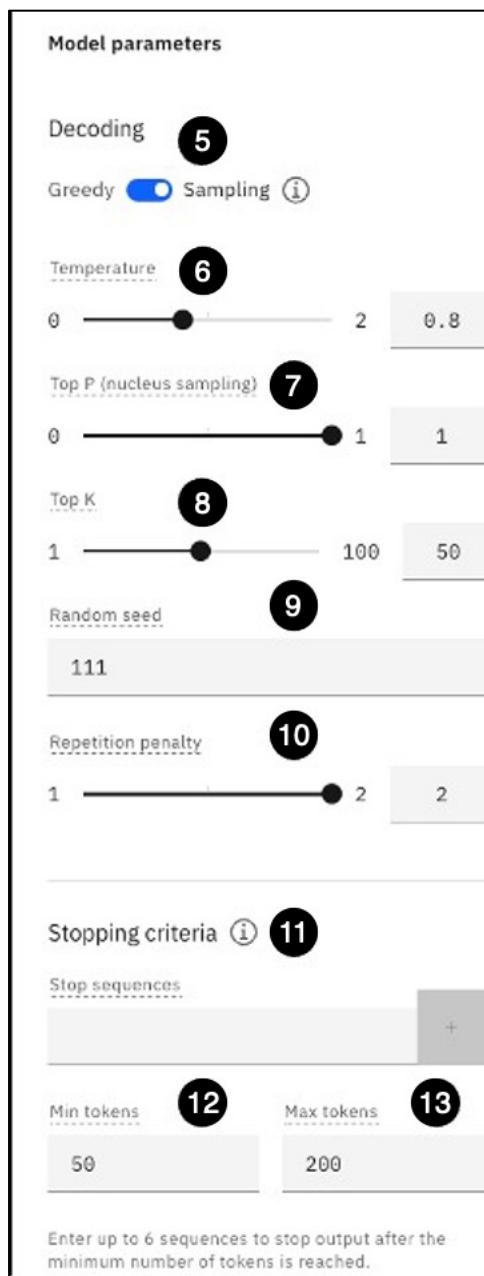
The output is too verbose. If you specify **have a great weekend!** as a stop sequence and you click **Generate**, you will get this output:

We are thrilled that you could make it to our workshop and hackathon today. It was great to meet everyone and we hope you enjoyed yourself. Thanks again for your participation and have a great weekend!

The model encountered the phrase “have a great weekend!” and it stopped generating any further output.

**12. Min tokens** – an integer that specifies the minimum number of tokens in the model’s output. Remember that tokens do not map 1-1 to words.

**13. Max tokens** – an integer that specifies the maximum number of tokens in the model’s output. Sometimes when the generated output looks incomplete, your Max tokens value may be too low.



Another thing to keep in mind is this: typically, you are charged by the number of tokens used, so setting a super high Max tokens value can be costly.

14. The default values used here are **Temperature** = 0.8, **Top P** = 1, and **Top K** = 50. This has resulted in some “creativity” with the flan-ul2-20b output on Marketing email generation:

**Email**

We've got great news for you! Golden Bank is now offering accounts with no fees, a 2% interest rate and no minimum balance. Don't miss out on this deal; it won't last long. Click here to find out more: <http://www.goldenbank.com/feefreepromotion.htm>

You will see how changing parameter values can change the output. Leaving the “Do not add any additional information” in the input Details section, change the value of **Temperature** from 0.8 and 0.4.

Now click **Generate**. You will see the following output:

**Email**

Golden Bank is offering a free checking account with no fees, a 2% interest rate, and no minimum balance. To apply, click on the link below. Offer ends July 15. Thank you for your time. Best Regards, [name]

Note the change in the **Temperature** value does have a positive effect. The generated email no longer contains a made-up website. Lowering the temperature has the effect of making the model less creative, and in this case, stopping a hallucination.

15. Next, you will look at the effect of changing the **Top P** parameter. Go back to the list of parameters and reset the Temperature to 0.8 and change the **Top**

P value from 1.0 to 0.5.

The screenshot shows the Watson Assistant interface. In the top navigation bar, 'Chat' is selected. The main area is titled 'Set up ^'. Under 'Instruction (optional)', there is a text input field containing: 'Generate a 5 sentence marketing message for a company with the given characteristics.' Below this is an 'Examples (optional)' section with two tabs: 'Details' and 'Email'. The 'Email' tab is selected, showing an 'Add example \*' button. On the right side, under 'Model parameters', there is a box containing: 'Reset Temperature from 0.4 to 0.8' and 'Change Top P from 1.0 to 0.5'. A red arrow points down to the 'Top P' slider, which is highlighted with a red box and labeled '15'. Other sliders shown are 'Temperature' (set to 0.8) and 'Top K' (set to 50). At the bottom left, there is a 'Try ^' button.

This setting means that the model will only consider the top set of words whose sum of probabilities adds up to  $> 0.5$ . It is removing a lot of the low-probability words. When you click **Generate**, you will see the following output:

The screenshot shows the generated email content. The subject line is 'Email'. The body of the email reads: 'Golden Bank is offering a free checking account with no fees, a 2% interest rate, and no minimum balance. To apply, click the link below. Offer ends July 15. Thank you for your time. Sincerely, [name removed]'.

This is again the kind of email you can send.

16. Now you will test another parameter. Reset the value of **Top P** to 1.0, and set the value of **Top K** (in half) to 25 as follows.

The screenshot shows the Qwen AI interface with the following details:

- Model:** flan-ul2-20b
- Set up:** Instruction (optional): "Generate a 5 sentence marketing message for a company with the given characteristics." Examples (optional): Details and Email.
- Model parameters:**
  - Decoding:** Temperature: 0.5 (set to 2.5), Top P (nucleus sampling): 1.0 (highlighted with a red box and labeled 16), Top K: 50 (set to 25).

Now click **Generate** to get the following output:

**Email**

We've got great news for you! Golden Bank is now offering accounts with no fees, a 2% interest rate and no minimum balance. Don't miss out on this deal; it won't last long. Click here to find out more: <http://www.goldenbank.com/feefree>

This shows that halving the value of **Top K** to 25 does not have the same kind of effect as halving **Top P**. The output is still hallucinating with the web link.

17. Continue to lower the value of **Top K**. You will find that you must lower it by quite a bit (to around 4) before you get a fairly good email as follows.

**Email**

We've got a special offer to help you save money! Open an account with us and enjoy no fees, a 2% interest rate, and no minimum balance. Click below for details. Offer ends July 15. Thank you. Golden Bank

Why the difference? It is useful to look at the difference between **Top P** and **Top K** which is best illustrated with an example.

Example: Difference between Top P and Top K

Consider this following example – a model has determined a set of potential next tokens for a particular input as follows:

Token	Probability
T1	0.16
T2	0.15
T3	0.14
T4	0.13
T5	0.12
T6	0.11
T7	0.10
T8	0.09

In this case when you set:

- **Top P = 0.5**, the model chooses randomly among T1, T2, T3, and T4 (because the probabilities sum up to  $> 0.5$ ).
- **Top K = 50**, the model chooses randomly among all 8 of the tokens (as  $8 < 50$ )

You need to lower **Top K** to 4 to have a similar effect as **Top P = 0.5** in this case. In that case, both settings ask the model to choose randomly from the top 4 tokens.

On the other hand, if the model sees the following:

Token	Probability
T1	0.92
T2	0.02
T3	0.01
T4	0.01
T5	0.01
T6	0.01
T7	0.01
T8	0.01

Then when you set:

- **Top P** to be 0.1, 0.2, 0.3, ... down to 0.9 – it makes no difference. In each case, the model would only consider token T1 (as T1 has a probability larger than the value of Top P).

- **Top K** to be 10, 20, 30, ... up to 90 – again it makes no difference. In each case (since there are only 8 possible tokens, and 8 is < 10, 20, .... 90), the model will randomly select from all 8 tokens.

It might look as if varying the values of Top P or Top K seem to have little to no effect. But remember a model determines how many possible next tokens and what the probabilities will vary, and generally, there are a lot of possible tokens. However, depending on how the model is designed and trained, varying **Top K** or **Top P** will have different effects.

### Section summary

- In this section, you have examined the effect of changing some configuration parameters.
  - **Temperature** – the higher the value the more creative the model is
  - **Top P** – lower value means less variability
  - **Top K** – lower value means less variability
- **Top P** and **Top K** are similar in limiting the tokens that the model will choose from to create an output but can have very different effects depending on the model (which has different ways of evaluating possible tokens) and calculating the probabilities.
- This exercise only examines varying the parameters one at a time. Combined with additional input to the prompt and combinations of parameters, you can already do a fair bit in making a foundation model respond as you want.
- Note that not all foundation models will handle the same kind of tasks well. You can (and probably should) find out more about a model before using it in production: perhaps experiment on your own with this section using different models.

## 6.5 Working with lists

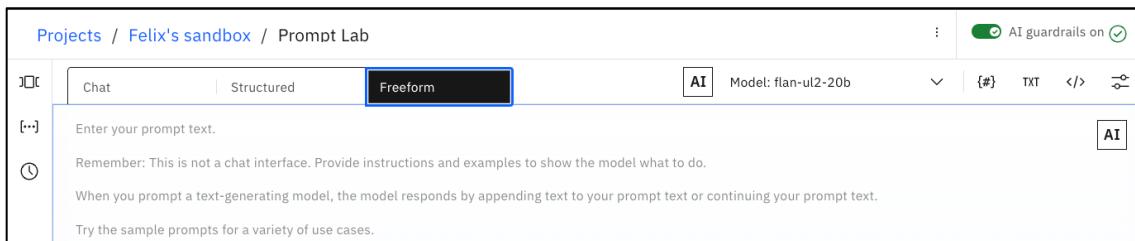
So far in this lab, you have looked at working with prompts and parameters to generate desirable email outcomes. In this part of the lab, you will generate a list output using the Prompt Lab in the Freeform mode.

It is important to remember that a foundation model does not read a prompt and interpret the intent like a human. It simply consumes a series of tokenized words and calculates the best next token. The concept of a list for output is simple for a human reader, here you will look at how LLMs might respond.

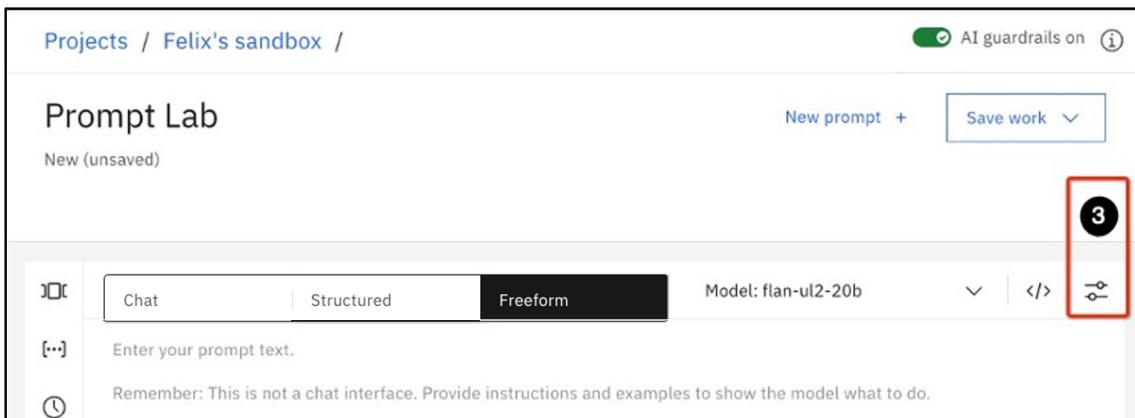
1. Open a new Prompt Lab session and select the **Freeform** tab instead of the default Structured tab.



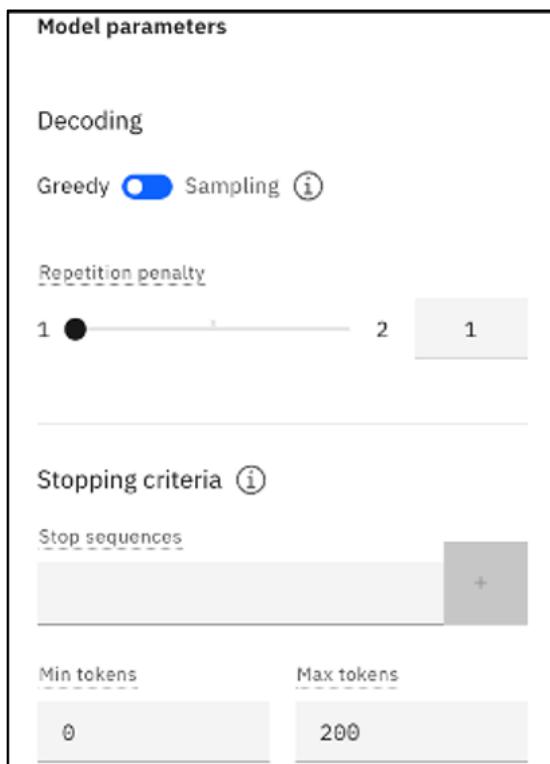
2. You will now be in the Freeform layout of the Prompt Lab. In the Freeform mode, you still have access to all the examples and all the LLMs but there is no structural guide.



3. Click the configuration parameter icon ( $\text{--} \text{--}$ ) on the upper right.

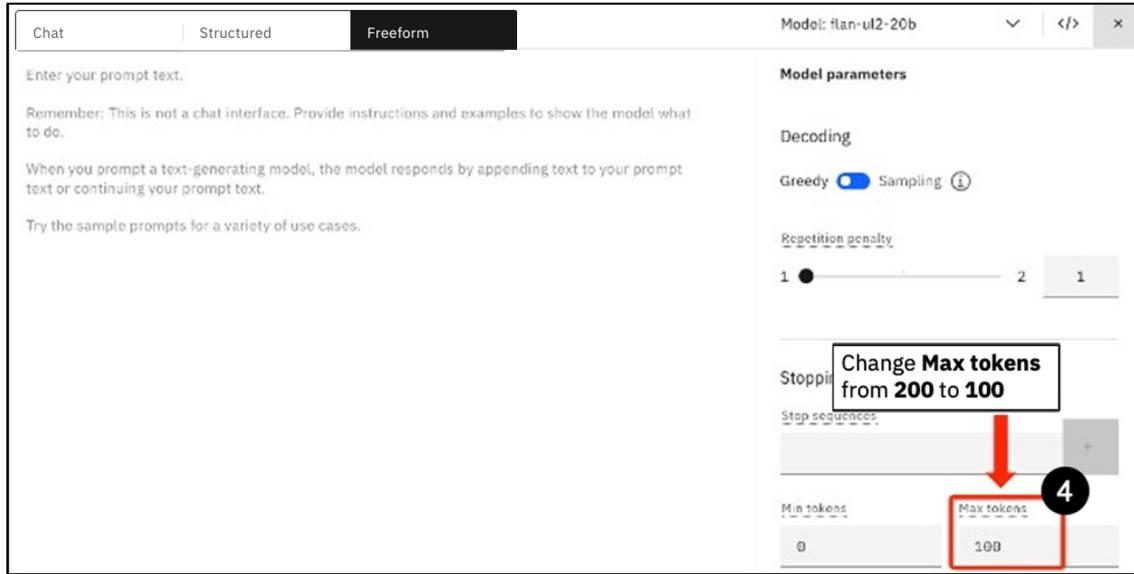


In **Freeform** mode, the parameters (and their defaults) look like:



Notes:

- These are the default if you have not selected a prompt sample. If you select a prompt sample, some of these values will change.
  - The Min tokens value is 0. This means that the output can potentially be empty.
  - Notice how the value of Max tokens is set to 200. In **Freeform** prompting you might be doing classification (which might require a small completion), or generating text (which may generate a lot of text depending on the model). The value of 200 is a good default but you might want to modify this according to your task at hand.
2. Verify that you are using **Greedy** decoding. Change the value of Max tokens to **100**.



3. Ensure that you are using the **flan-ul2-20b** model. Make a change to flan-ul2-20b if necessary. Copy and paste the following text in the input field:

The following paragraph is a consumer complaint.

Read the following paragraph and list all the issues. I called your helpdesk multiple times and every time I waited 10-15 minutes before I gave up. The first time I got through, the line got cut suddenly and I had to call back. This is just not helpful. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and I know what I am doing. It was a complete waste of time. After like an eternity of this pointless conversation, I was told that an SME will contact me. That – was 2 days ago. What is the problem with your support system?

The list of issues is as follows:

Click **Generate**. You will see the following output (in red):

The screenshot shows a user interface for generating text. At the top, there are three tabs: Chat, Structured, and Freeform. The Freeform tab is selected. To the right of the tabs, it says "Model: flan-ul2-20b". Below the tabs, there is a text input area with the following content:

The following paragraph is a consumer complaint.

Read the following paragraph and list all the issues. I called your helpdesk multiple times and every time I waited 10-15 minutes before I gave up. The first time I got through, the line got cut suddenly and I had to call back. This is just not helpful. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and I know what I am doing. It was a complete waste of time. After like an eternity of this pointless conversation, I was told that an SME will contact me. That – was 2 days ago. What is the problem with your support system?

Summarize the list of issues as follows:

I called your helpdesk multiple times and every time I waited 10-15 minutes before I gave up. The first time I got through, the line got cut suddenly and I had to call back. This is just not helpful. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and know what I am doing. It

This is the output text:

Summarize the list of issues as follows:

I called your helpdesk multiple times and every time I waited 10-15 minutes before I gave up. The first time I got through, the line got cut suddenly and I had to call back. This is just not helpful. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and know what I am doing. It

Note the following:

- The Generate button is not active until some text is entered into the field.
  - The model did not output a list.
  - The model just repeated some of the input text word by word.
  - This is not a desirable result – the model probably does not quite understand the idea of a list.
  - It terminates with “It” because it was at the limit of 100 tokens (keep in mind that 100 tokens do not mean 100 words).
4. Switch to the **flan-t5-xxl-11b** model. Remove the generated completion and click **Generate**. Note how the model outputs something similar to that of flan-ul2-20b. Both of the flan models (without adding more to the prompt or changing parameters) seem to have understood that to “list” the issues is simply to recount what it was told.
  5. Tip – you can guide the flan models to provide a better output by adding a “1.” to your input prompt (this is telling the model you want to have a structured list output). Go back to the flan-ul2-20b model and update the prompt as follows ((highlighted in red below):

The following paragraph is a consumer complaint.

Read the following paragraph and list all the issues. I called your helpdesk multiple times and every time I waited 10-15 minutes before I gave up. The first time I got through, the line got cut suddenly and I had to call back. This is just not helpful. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and I know what I am doing. It was a complete waste of time. After like an eternity of this pointless conversation, I was told that an SME will contact me. That – was 2 days ago. What is the problem with your support system?

The list of issues is as follows:

1.

Click on Generate and the output looks like this:

Summarize the list of issues as follows:

1. I called your helpdesk multiple times and every time I waited 10-15 minutes before I gave up. 2. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and I know what I am doing. 3. After like an eternity of this pointless conversation, I was told that an SME will contact me. That

Now the model outputs a list, even though it still is simply repeating the input.

Change to the model **granite-13b-instruct-v2** model. Remove the generated completion and the 1 from the prompt (return to the original prompt) and click **Generate**. You will see the following output:

The list of issues is as follows:

- waited 10-15 minutes every time I called
- the first time I got through, the line got cut suddenly and I had to call back
- after 3 days I finally got through, the agent kept going over a long checklist of trivial things and asking me to verify
- after an eternity of this pointless conversation, I was told that an SME will contact me

This is actually a fairly good output summary and in list format. For this kind of request, the **granite-13b-instruct-v2** model is much better than the flan models. Moreover, it correctly interpreted the sentiment of the client – one of frustration.

6. Let's try one more model – the gpt-neox-20b model ((if necessary, reference Section 6.2, Steps 7-10 on changing model). Remove the generated output. Click **Generate**, you will get the following output:

Summarize the list of issues as follows:

- \* The agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and I know what I am doing.
- \* It was a complete waste of time.
- \* After like an eternity of this pointless conversation, I was told that an SME will contact me.

The following paragraph is a consumer complaint.

Read the following paragraph and list all the issues. I called your helpdes

The output is in a list form but the information is not as good – and at the end of the completion the model is filling in and using up tokens simply because they are available.

#### Section summary:

- In this section, you looked at the Prompt Lab using the Freeform mode. Once you have become more experienced with prompting, you may want to use this mode by default.
- Models handle output requirements differently. Some models (like the flan models) are better with text output (at least with zero-shot prompting). You can guide the output if you update the prompt with the hint of a “list structure”. Other models, such as the **granite-13b-instruct-v2**, and to a lesser degree the **gpt-neox-20b model**, are better able to understand the concept of list output without additional instructions.

In general, the **instruct** models (with the word “instruct” in the name) are better at handling requests for structured output and following instructions.

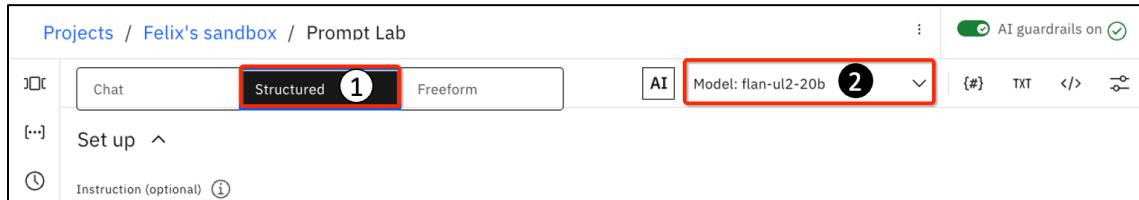
- Once again, the size of the model does not matter – here the model with the smaller number of parameters (**granite-13b-instruct-v2**) provided a better completion than the larger **flan-ul2-20b** model. Users should NOT be sidetracked by the size of the model. It is more important to understand the strengths of the model, and what it might have been trained and specifically tuned for.

In particular, while the **granite-13b-instruct-v2** model is smaller, it was trained by IBM to follow a user's instructions. This is why it is better able to understand prompts where the user is instructing the model to output in a particular (list) format.

## 6.6 Working with JSON

In this section, you will use the Prompt Lab to generate a simple JSON file.

1. Open a new Prompt Lab session in the Structured mode.
2. Ensure that you are using the **flan-ul2-20b** model.



3. Ensure that you are using the **Greedy** mode.
4. Set **Max tokens** to 100.

A screenshot of the Prompt Lab interface. On the left, there is a 'Set up' section with an 'Instruction (optional)' input field containing placeholder text: 'Tell the model what to do. For example: Summarize the transcript.' Below it is an 'Examples (optional)' section with an 'Input' field containing 'Enter your example input here.' and an 'Output' field. There is also a 'Generate' button. On the right, there is a 'Model parameters' section. Under 'Decoding', there is a 'Greedy' toggle switch (which is highlighted with a red box and has a circled '3' over it) and a 'Sampling' link. Under 'Stopping criteria', there is a 'Max tokens' input field containing '100' (which is highlighted with a red box and has a circled '4' over it). A callout box says 'Ensure it is set to Greedy' with an arrow pointing to the Greedy switch. Another callout box says 'Change Max tokens from 20 to 100' with an arrow pointing to the Max tokens input field.

5. Copy and paste the following to the Input field under the Try section

Create a JSON file output with the following information

name: Joe

age: 25

Phone: 416-1234-567

Phone: 547-4034-240

Address: City: Markham, Street: Warden Avenue, Postal Code: L6G 1C7

Your input field should now look like this:

Try ^

Test your prompt ⓘ

**Input:**

Create a JSON file output with the following information

name: Joe  
age: 25  
Phone: 416-1234-567  
Phone: 547-4034-240  
Address: City: Markham, Street: Warden Avenue, Postal Code: L6G 1C7

Click **Generate**. The flan-ul2-20b model returns the following output:

**Output:**

Joe, age 25, has two phone numbers, 416-1234-567 and 547-4034-240. His address is in Markham, Warden Avenue, L6G 1C7.

This shows the strengths of the flan-ul2-20b model in that it responded with a natural language output. However, it clearly is not the JSON output you are looking for.

6. Switch to the **flan-t5-xxl-11b** model and again click **Generate**. Similar to the flan-ul2- 20b model, you will see this output:

**Output:**

```
Joe is 25 years old and lives at Warden Avenue, Markham, Ontario, Canada.  
His phone number is 416-1234-567 and his fax number is 547-4034-240.
```

Again the completion provides a good natural language output, but it is not in JSON. Moreover, this LLM made the 547 number into an incorrect fax number. At this juncture, you can try using one-shot or few-shot prompting to help the model understand how you want to structure the output. However, this would not be an easy task. Instead, we will move on and try other models.

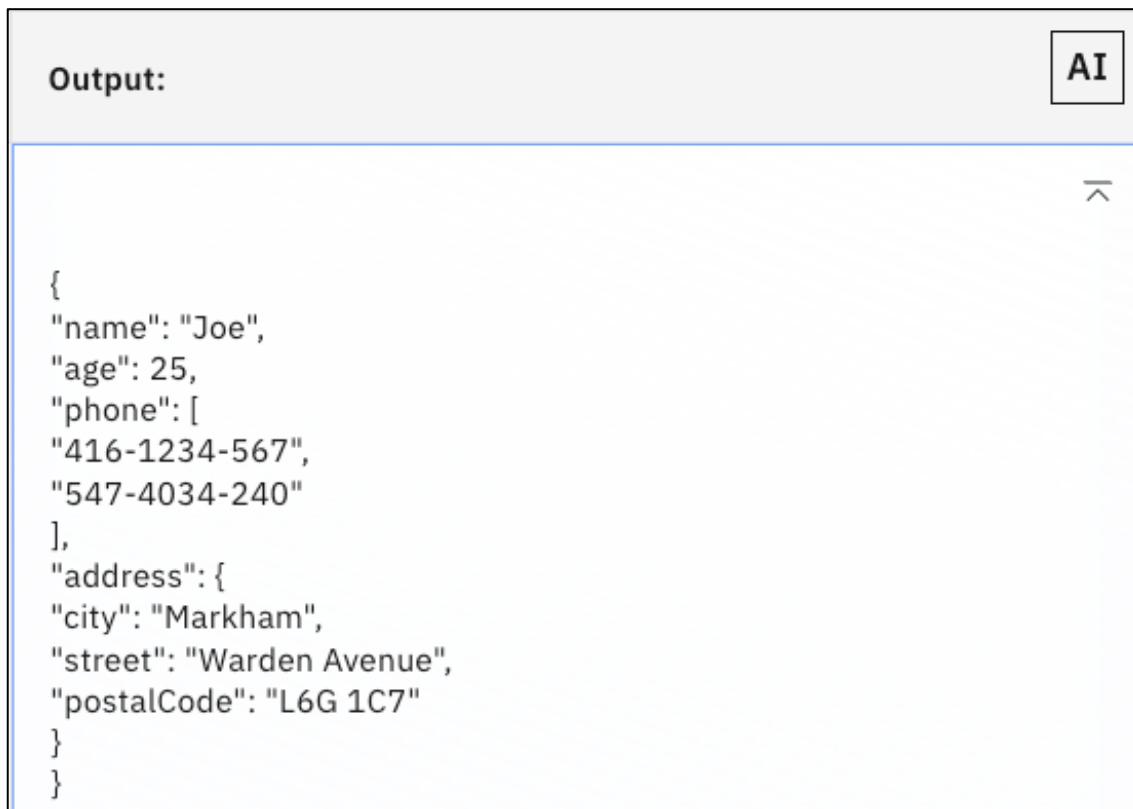
7. Now try the **granite-13b-instruct-v2** model and click **Generate**. You will see this output:

**Output:**

```
{  
  "name": "Joe",  
  "age": 25,  
  "Phone": [  
    "416-1234-567",  
    "547-4034-240"  
  ],  
  "Address": {  
    "City": "Markham",  
    "Street": "Warden Avenue",  
    "Postal Code": "L6G 1C7"  
  }  
}
```

This is exactly the output you want. The **granite-13b-instruct-v2** does have a better understanding of what a JSON output should look like,

- Now try another model that we have not yet experimented with. Select the **codellama-34b-instruct-hf** model and then click on **Generate**. You get the following output.



The screenshot shows a user interface for generating AI output. At the top left is the word "Output:" and at the top right is a small "AI" icon. Below this is a large text area containing the following JSON code:

```
{  
  "name": "Joe",  
  "age": 25,  
  "phone": [  
    "416-1234-567",  
    "547-4034-240"  
  ],  
  "address": {  
    "city": "Markham",  
    "street": "Warden Avenue",  
    "postalCode": "L6G 1C7"  
  }  
}
```

The **codellama-34b-instruct-hf** model generates the valid JSON output, just like **granite-13b-instruct-v2**.

- The last model to try is the **llama-2-70b-chat** model. Switch to this model and click **Generate**. You will see this output.

### Output:

```
{  
  "name": "Joe",  
  "age": 25,  
  "phone": [  
    "416-1234-567",  
    "547-4034-240"  
  ],  
  "address": {  
    "city": "Markham",  
    "street": "Warden Avenue",  
    "postalCode": "L6G 1C7"  
  }  
}  
  
I
```

On one hand, the **llama-2-70b-chat** does generate the proper JSON code. However, as you have seen large models tend to be “talkative” and the LLM certainly continues to generate – as can be seen by the capital **I** at the bottom. The LLM stopped because it ran out of tokens or more text would be generated.

While this is not a big issue, we will now use the **Stop sequences** configuration setting to eliminate the **I**.

In this particular case, you want to stop the generation once the model encounters the following text sequence (2 curly brackets that conclude the JSON section).

```
}
```

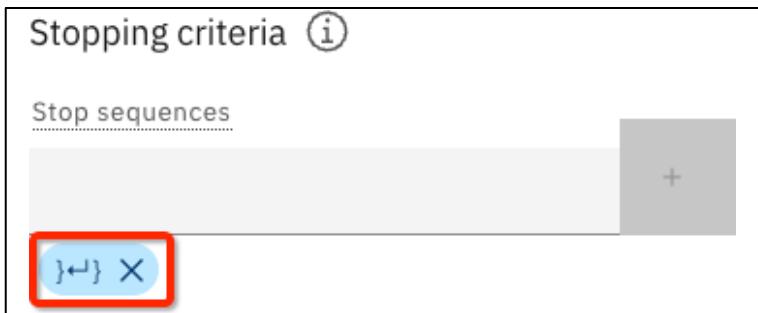
10. In the upper right corner of the console, click the **Model parameters** slide-out icon (☞) to show the list of model parameters. Enter the following keystrokes in the **Stop sequences** field:

```
} <carriage return> }
```

You should see the following, then click on the blue + button to add the sequence.



The stop sequence is now added. You will see this:



Now click Generate. The output (see below) no longer includes the triple double quotes as the model recognizes the stop sequence and ceases to generate anything more.

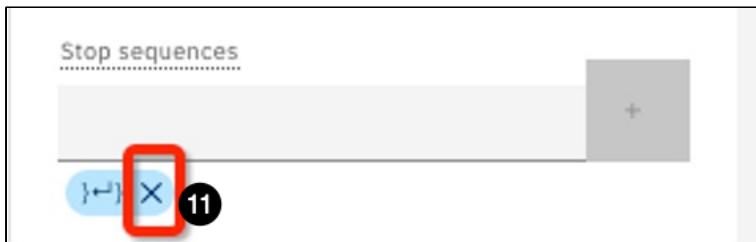
**Output:**

```
{  
  "name": "Joe",  
  "age": 25,  
  "phone": [  
    "416-1234-567",  
    "547-4034-240"  
  ],  
  "address": {  
    "city": "Markham",  
    "street": "Warden Avenue",  
    "postal_code": "L6G 1C7"  
  }  
}
```

**Note:** You *might* need to turn the **AI guardrails off** for this to work. This is currently under investigation and may already be fixed when you run Step 10



11. You can remove the Stop sequence by clicking X.



Bonus tasks/questions:

- Remove any Stop sequences. What if you specify just one curly right bracket instead? Will it work?  
Try it out, does it do what you expected?
- Again remove the previous Stop sequence. What if you specify 2 curly right brackets (with no carriage return in between) like this }}}, will it work?  
Try it out, does it do what you expected?

## Section summary

- In this section, you tried generating a JSON output using different models.
- While the **flan** models have shown good capability in generating text in natural language, they do not seem to understand how to output in JSON format, at least not with zero-shot prompting.
- The **granite-13b-instruct-v2** model and the **codellama-34b-instruct-hf** model both can generate good JSON output.
- You also experienced using the **Stop sequences** to stop a model from outputting unwanted text. There might be cases when a model generates unwanted text and consumes valuable tokens. You can use this technique to

specify when the model should stop (multiple **Stop sequences** can be specified and the model will stop whenever it encounters any one of them).

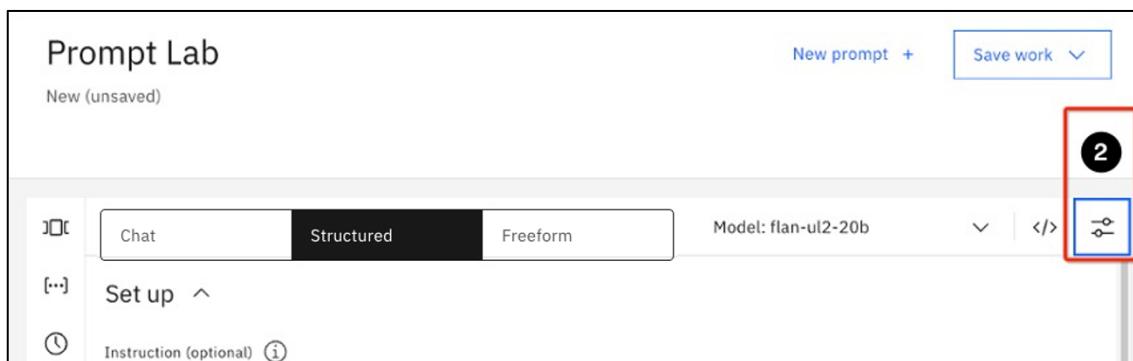
You learn that in **Stop sequences** it is important to include things such as <carriage return> or it will not work properly.

- From the list output and the JSON output exercise, you learned 2 very important facts:
  - No one model will rule them all.
  - Different models are trained/tuned for different tasks and abilities. You should look for models that are trained with domain-specific data with a focus on various data structures and capabilities.

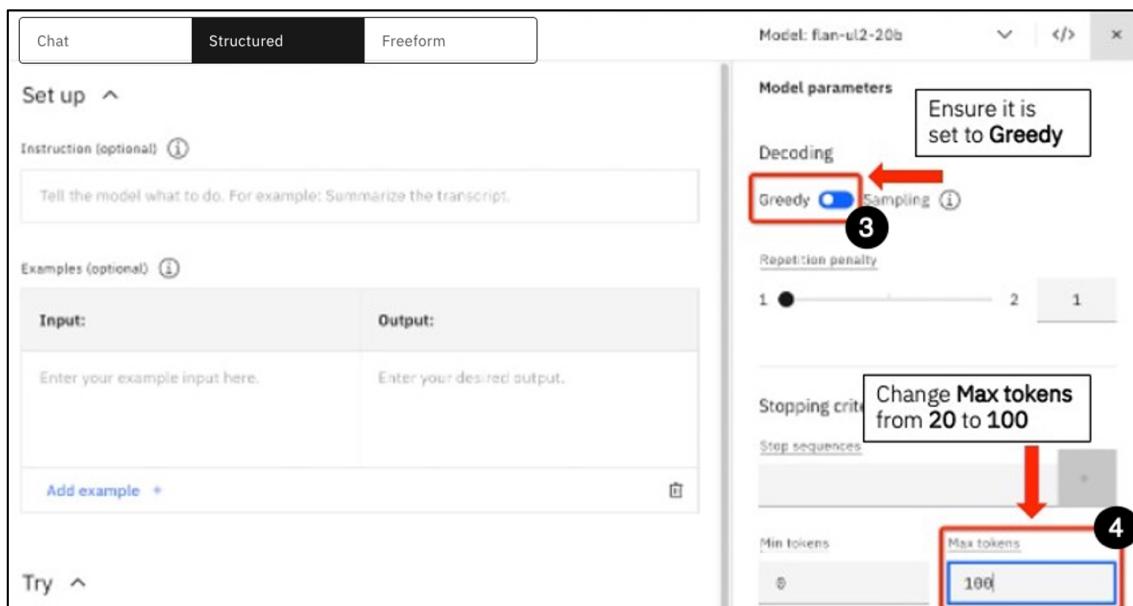
## 6.7 Working with one-shot prompting

So far you have been working with zero-shot prompting. In this section, you will experiment with one-shot prompting. For this section, you will be using the Structured mode.

1. Open a new Prompt Lab session. Ensure that you are using the **flan-ul2-20b** model.
2. Click the **Inference parameter** (  ) icon to slide out the configuration parameters.



3. Ensure that you are using the **Greedy** mode.
4. Change the number of **Max tokens** from **200** to **100**.



5. Cut and paste the following text into the Try section's Input field.

The following paragraph is a consumer complaint. The complaint is about one of these options: credit cards, credit reporting, mortgage and loans, retail banking, or debt collection. Read the following paragraph and list all the issues.

I called your help desk multiple times and every time I waited 10-15 minutes before I gave up. This is just ridiculous. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and I know what I am doing, It was a complete waste of time. After like an eternity of this pointless conversation, I was told that an SME will contact me. That - was 2 days ago. What is the problem with your support system?

Your section should now look like this:

Try ^

Test your prompt (i)

**Input:**

The following paragraph is a consumer complaint. The complaint is about one of these options: credit cards, credit reporting, mortgage and loans, retail banking, or debt collection. Read the following paragraph and list all the issues.

I called your help desk multiple times and every time I waited 10-15 minutes before I gave up. This is just ridiculous. When I finally got through like after 3 days (yes, 3 days) your agent kept going over a long checklist of trivial things and asking me to verify, after I repeatedly told the agent that I am an experienced user and I know what I am doing, It was a complete waste of time. After like an eternity of this pointless conversation, I was told that an SME will contact me. That - was 2 days ago. What is the problem with your support system?

Click **Generate**. Your output will get an **Output** like this:

**Output:**

[The input was rejected as inappropriate] ^

6. The reason is that there is this word **ridiculous** in the input and you have **AI guardrails on**. You can change and remove that word, or for this lab, simply turn off the **AI guardrails**.



7. Click **Generate** and you will get the following:

A screenshot of the AI output section. It has a header "Output:" and a main content area with a light blue background. The text "Credit Cards" is displayed in white. There is a close button (an "X") in the top right corner of the blue area.

This is clearly not the output you want as you want to list the issues. This model took the first part of the prompt and simply read the content and determined that it was about credit cards. In this case, the model has performed a classification instead of a summarization.

You could try to update your prompt; however, this is not the focus of this part of the lab. Instead, you will try one-shot prompting – that means you provide an example to the model to “teach” (or steer) it to respond properly to the prompt.

8. Cut and paste the following text to the **Set up** section’s **Input** field in the **Examples (optional)** section.

The following paragraph is a consumer complaint.  
The complaint is about one of these options: credit cards, credit reporting, mortgages and loans, retail banking, or debt collection. Read the following paragraph and list all the issues.

I bought a GPS from your store and the instructions included are in Spanish, not English. I have to use Google Translate to figure it out. The mounting bracket was broken, and so I need information on how to get a replacement. Moreover, the information seems to be outdated because I cannot see the new roads put in around my house within the last 12 months.

## Set up ^

Instruction (optional) [\(i\)](#)

Tell the model what to do. For example: Summarize the transcript.

Examples (optional) [\(i\)](#)

### **Input:**

The following paragraph is a consumer complaint.

The complaint is about one of these options: credit cards, credit reporting, mortgages and loans, retail banking, or debt collection. Read the following paragraph and list all the issues.

I bought a GPS from your store and the instructions included are in Spanish, not English. I have to use Google Translate to figure it out. The mounting bracket was broken, and so I need information on how to get a replacement. Moreover, the information seems to be outdated because I cannot see the new roads put in around my house within the last 12 months.

This is a sample **input**. You also need to provide a sample **output** to complete the one-shot prompt.

9. Cut and paste the following text into the **Output** field next to the **Set up Input** field:

The list of issues is as follows:

- 1) The instructions are in Spanish, not English.
- 2) The mounting bracket is broken.
- 3) The information is outdated.

Your screen should look like this:

Examples (optional) [\(i\)](#)

Input:	Output:
The following paragraph is a consumer complaint. The complaint is about one of these options: credit cards, credit reporting, mortgages and loans, retail banking, or debt collection. Read the following paragraph and list all the issues	The list of issues is as follows: 1) The instructions are in Spanish, not English. 2) The mounting bracket is broken. 3) The information is outdated.

[Add example +](#) ✖

10. Click **Generate** to get a new output. You will see this in the Try section's **Output** field:

**Output:**

The list of issues is as follows: 1) Waiting time to talk to someone 2) Agent kept going over a long checklist of trivial things 3) Agent kept asking me to verify 4) Agent kept telling me that an SME will contact me 5) SME never contacted me ✖

This output has been vastly improved from the previous output. Note how the completion listed all the valid issues and put them out in a numbered list.

**ATTENTION:** Do NOT close out the session or remove any information from any field. You will need this for the next section.

## Section summary

- In this section, you experimented with one-shot prompting and observed how providing an input example can help a model respond more desirably. An example can:
  - Teach a model to focus on the right information to use from the input; for example, ignore the type of complaint but focus on the substance of the complaint.
  - Teach a model how to generate a desired output; for example the structure – in our example a numbered list.
- In few-shot prompting, you would provide multiple input/output examples. Remember! Foundation models are not answering questions. However, they can learn patterns. A one-shot or few-shot prompt provides patterns to guide a model to generate output that has the best “probability” of matching the provided samples.

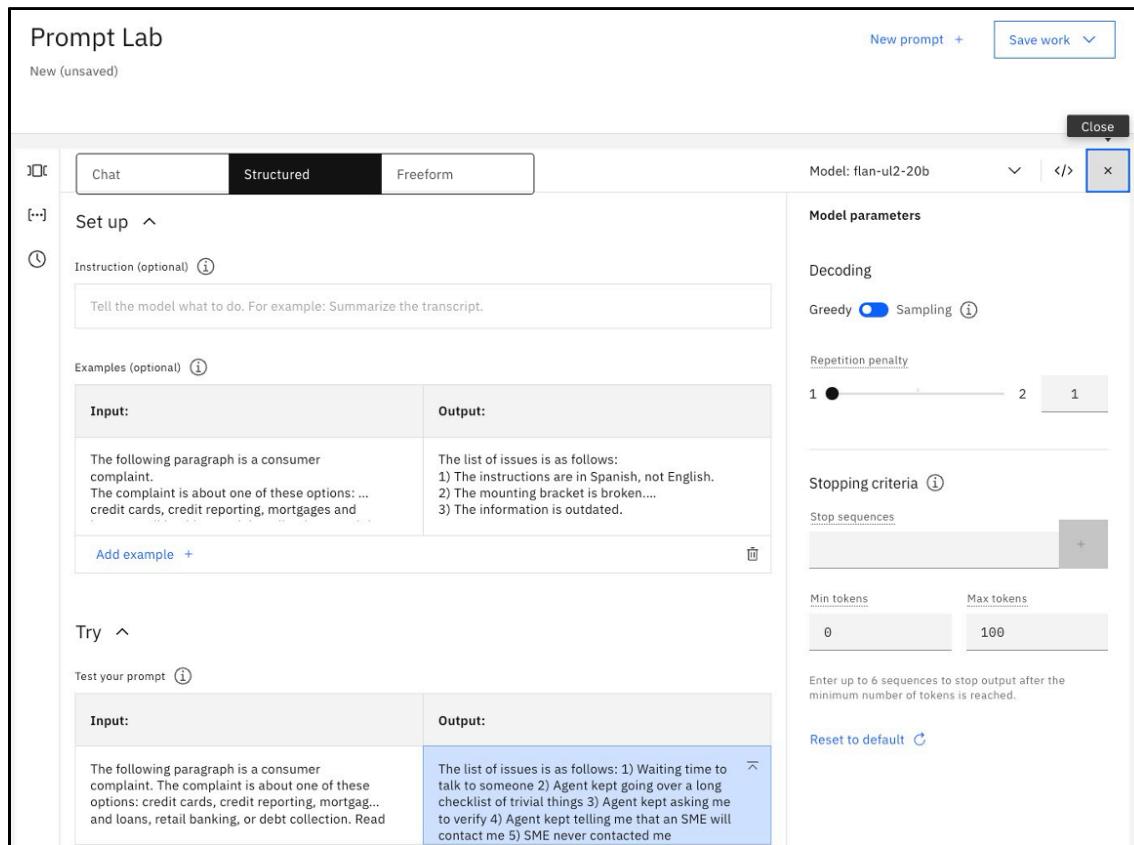
## 6.8 Saving prompts and prompt sessions

You have seen many examples of prompts and have done several prompt engineering examples. The more experienced you are, the better you will be able to modify a prompt and use different parameter settings to optimize the model for your desired output. You have also seen how to add additional examples to “teach” the model with single-shot in-context learning.

Sometimes you must modify a prompt several times in different ways to achieve the desired effect. You definitely want to save prompts that work as they are good examples and reminders.

You can save a “finished” prompt as follows.

1. You should have the Prompt Lab still open with the modified prompt from Section 6.7. If not, you can re-generate it using the steps from 6.7. It should look like this:



2. Go to the top of the console and click on the **floppy disk** (  ) pulldown and click **Save as**.



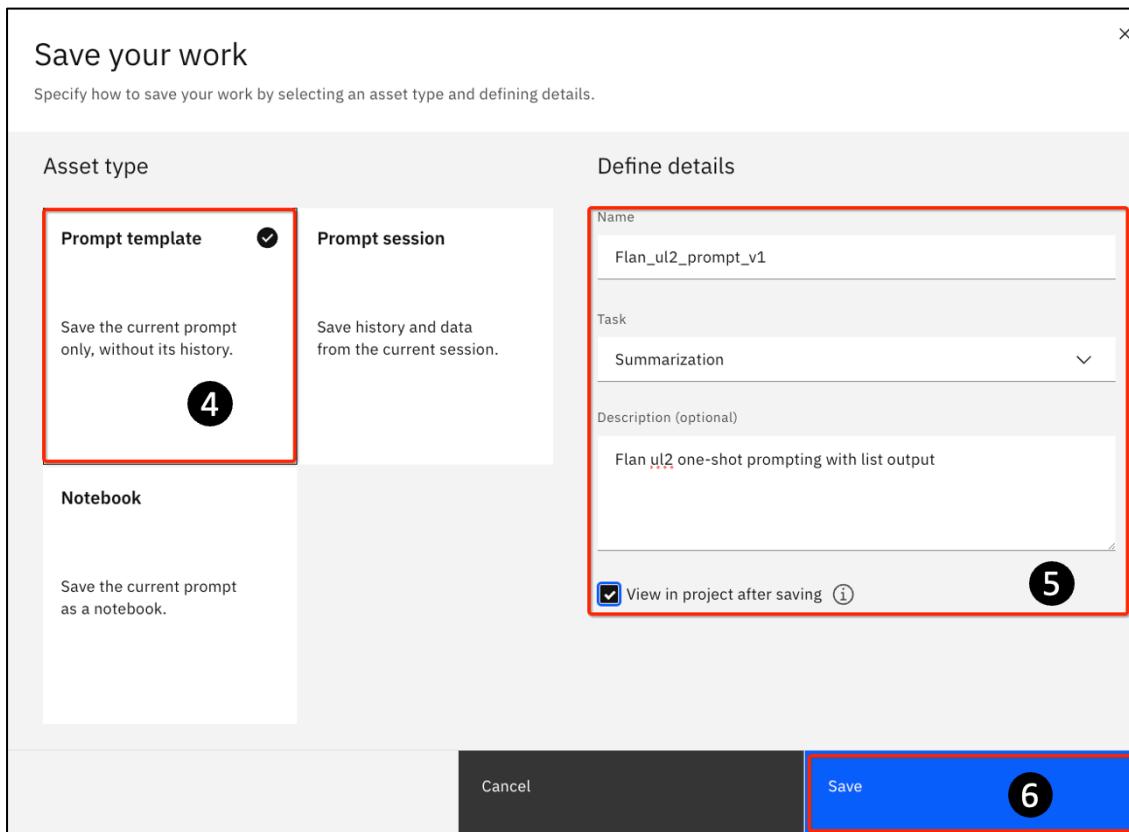
3. You will now see the following panel:

 A screenshot of the 'Save your work' dialog box. It has two main sections: 'Asset type' and 'Define details'. 
 - \*\*Asset type\*\*: A table with two rows.
 - Row 1: 'Prompt' (selected) vs 'Prompt session'. 'Prompt' is described as 'Save the current prompt only, without its history.'; 'Prompt session' is described as 'Save history and data from the current session.'.
 - Row 2: 'Notebook'. It is described as 'Save the current prompt as a notebook.'.
 - \*\*Define details\*\*: This section is titled 'Name' and contains a text input field with placeholder 'Enter a name'. Below it is a 'Description (optional)' field with placeholder 'What's the purpose of this prompt asset?'.
 - At the bottom right of the dialog is a checkbox labeled 'View in project after saving' with a small help icon.

You can save your assets in 3 different ways:

- As a **Prompt template** – This saves the current state of the prompt which includes any input, output, and configuration runtime parameters you might have changed. Think of this as taking a snapshot of the panel. Do this when you find a combination that works well for your purpose.
- As a **Prompt session** – Saves the history of your prompt engineering session, tracking the changes in prompt text and configuration parameters (up to 500 steps). This is useful when you are experimenting and you want to go back to a previous setting that might have worked better.
- As a **Notebook** – Saves the information in a Jupyter notebook that you can edit and modify later. You will do this in a later section of this lab.

4. Select the **Prompt template** tile.
5. Enter the following contents on the right.
  - **Name:** Flan.ul2\_prompt\_v1
  - **Task:** Summarization
  - **Description:** Flan.ul2 one-shot prompting with list output
  - Select **View in project after saving**
6. Click **Save**. Your project will open up to the Assets tab.



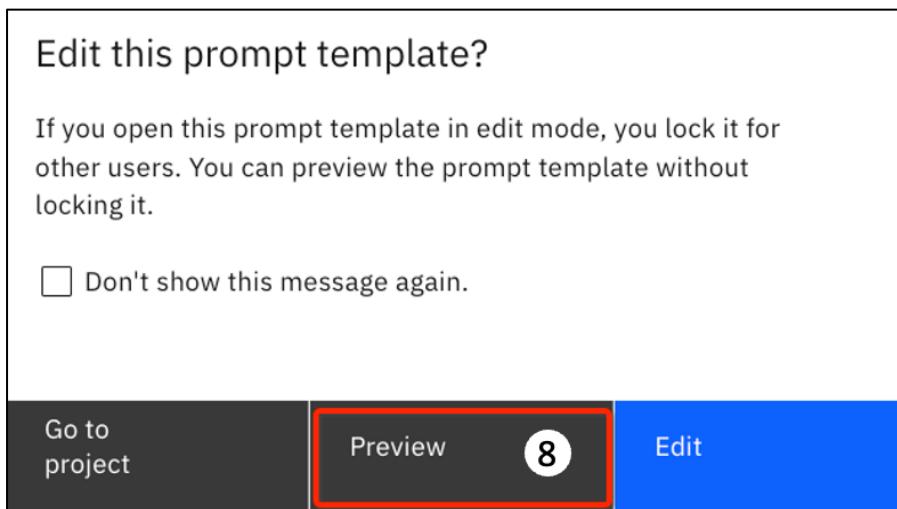
You can see your saved Flan.ul2\_prompt\_v1 prompt (you may or may not have other assets that will be shown here too).

7. Click on the **Flan.ul2\_prompt\_v1** asset to load the saved snapshot of your prompt.

The screenshot shows the Project Overview page with the 'Assets' tab selected. On the left, there's a sidebar with '9 assets' and 'All assets' highlighted. The main area shows a table titled 'All assets' with two rows:

Name	Last modified
[...] Flan.ul2_prompt_v1 Prompt	Now Modified by you
Successful L2 example_1 Notebook	3 days ago Modified by you

8. On the subsequent dialog box, select **Preview**.



This puts you in the view mode. You can also go into the **Edit** mode to change it.

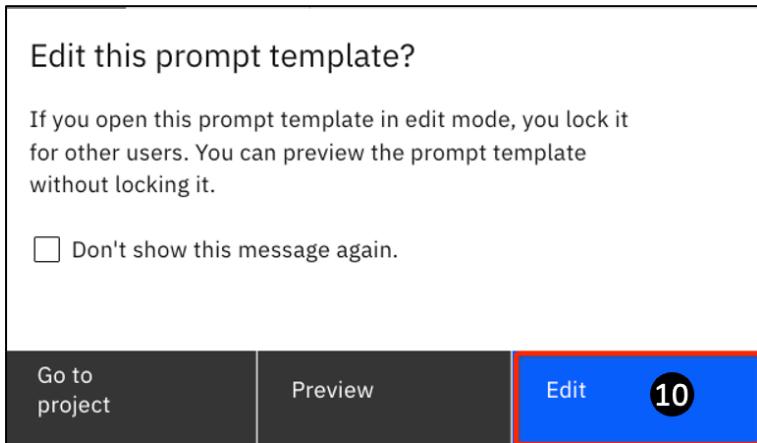
You can save your prompt whenever you find something that is useful and come back to it later. However, there are times when as you change a prompt (adding text to the prompt, changing parameters, etc.) the results deteriorate and do not improve. In these circumstances, you may want to move back to a previous setting and change other parameters, or simply settle on a version that you tried to improve (but were not able to).

You can always save your prompt every time you make a change; however, it would be difficult to keep track and a better approach would be to save a prompt session.

9. Click the pencil icon (  ) to go into **Edit** mode.



10. Click **Edit** this time.



11. Click the **Inference parameter** (  ) icon to slide open the list of configuration parameters. You will make a series of changes. The focus of this exercise is tracking the prompt change. You should also observe the generated output to see how it changes as you modify the parameters. Ensure you click Generate at the end of each step.

- Change the **Repetition penalty** to 1.8, then click **Generate**
- Change the **Repetition penalty** to 1.6, then click **Generate**
- Change the **Repetition penalty** to 1.4, then click **Generate**
- Change the **Repetition penalty** to 1.2, then click **Generate**
- Change the **Decoding mode** from **Greedy** to **Sampling**, then click **Generate**
- Change **Top P** to 0.5, then click **Generate**
- Change **Top K** to 25, then click **Generate**

12. Go to the top of the console and click on the **floppy disk** (  ) pulldown and click **Save as**.



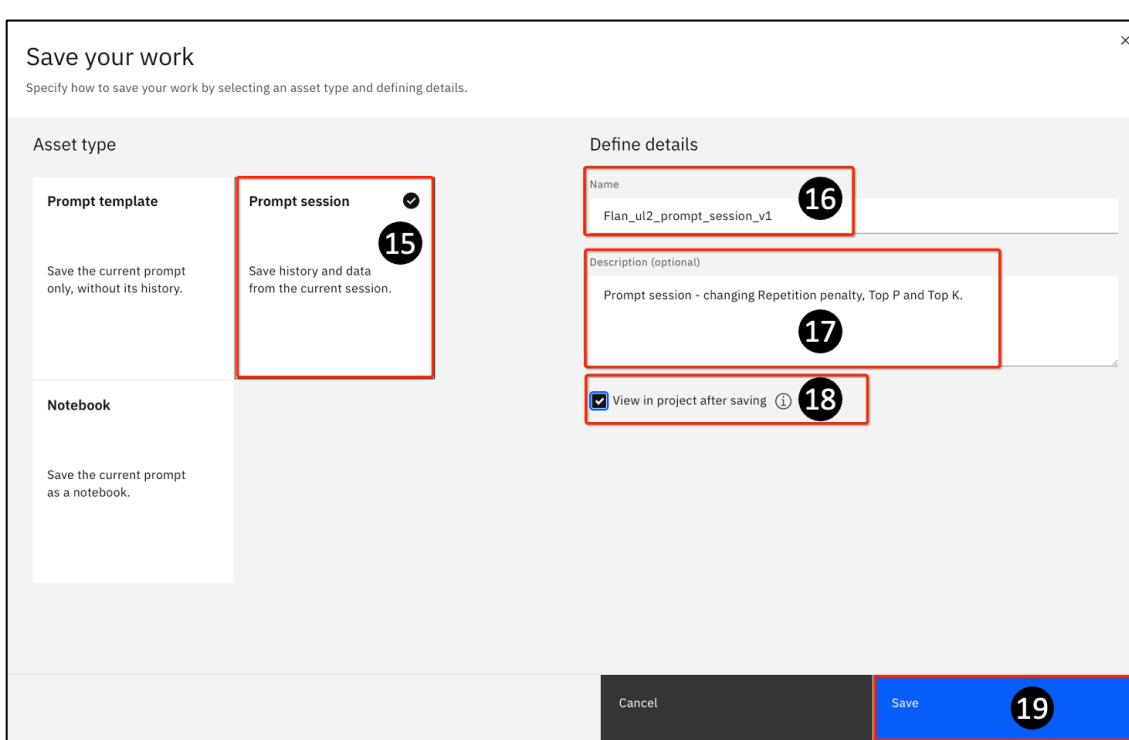
There are Save and Save as options (see 1 above). You started with a saved prompt, so if you select Save you will be updating that saved prompt.

13. Instead, you will select **Save as** to save this prompt session to a different asset – this time as a Prompt session.



14. You will see the **Save your work** panel that you saw in Step 5. Do the following:

15. Select the **Prompt session** tile and enter the following information on the right.
16. **Name:** Flan\_ul2\_prompt\_session\_v1.
17. **Description:** Prompt session - changing Repetition penalty, Top P and Top K.
18. Select **View in project after saving**.

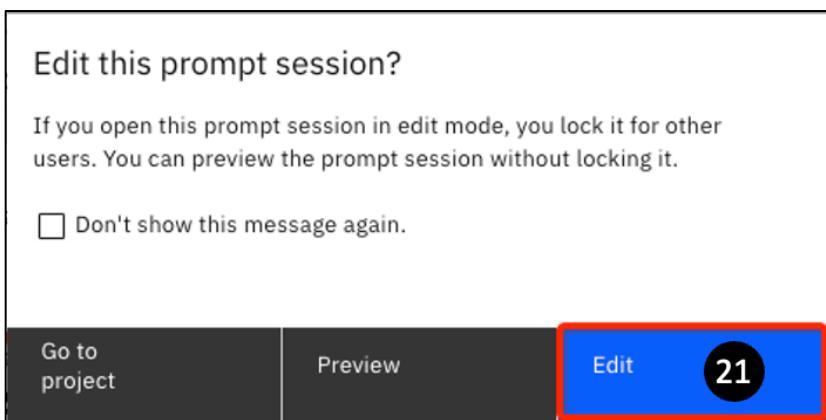


20. On the project panel, you will now see this saved prompt session (as well as the previously saved prompt). Click on the **Flan\_ul2\_prompt\_session v1** prompt session.

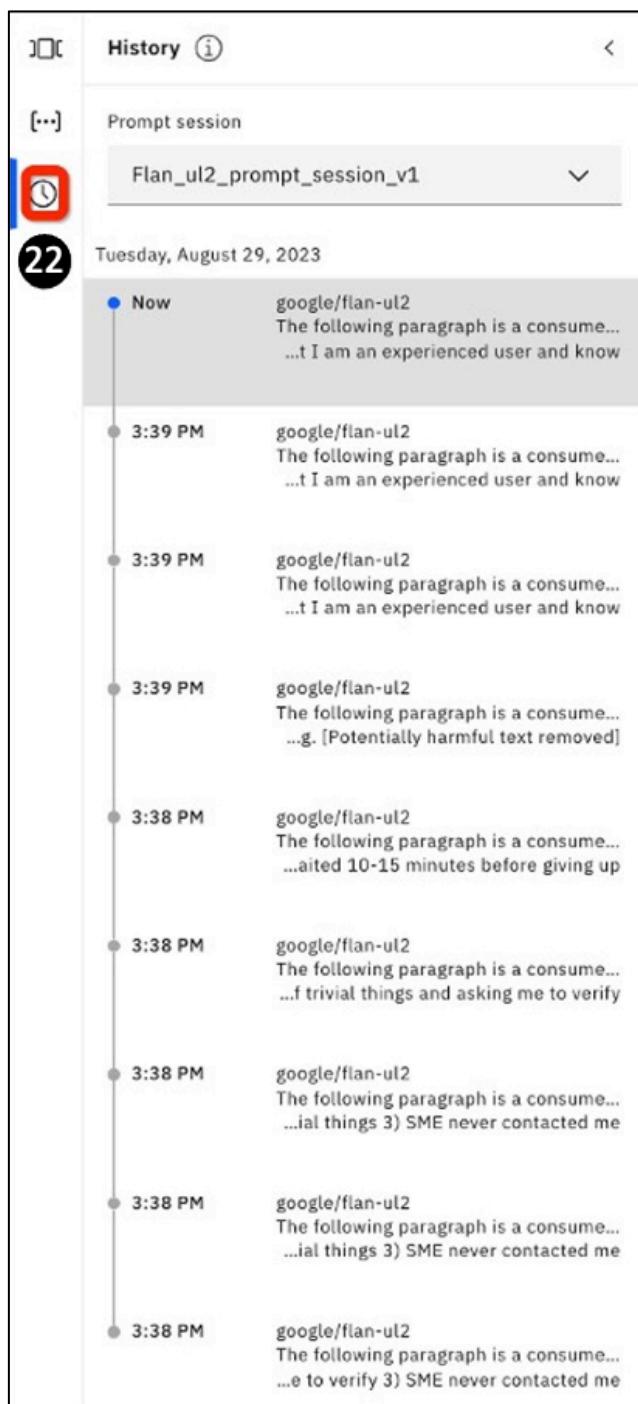
The screenshot shows the 'Assets' tab in the Project interface. On the left, there's a sidebar with '5 assets' and 'Asset types'. The main area shows a table titled 'All assets' with columns for 'Name', 'Last modified', and actions. Two assets are listed: 'Flan\_ui2\_prompt\_session\_v1' (Prompt session) and 'Flan\_ui2\_prompt\_v1' (Prompt template). The first asset is highlighted with a red box and circled with the number 20.

Name	Last modified
[...] Flan_ui2_prompt_session_v1 Prompt session	Now Modified by you
[...] Flan_ui2_prompt_v1 Prompt template	20 minutes ago Modified by Service

21. For this part of the lab, do NOT pick **Preview**, click **Edit**.



22. Click the **clock** (⌚) icon on the left to open the history of the prompt session.



The screenshot shows the 'History' interface for a 'Prompt session' named 'Flan\_ul2\_prompt\_session\_v1'. The session was last updated 'Now' at Tuesday, August 29, 2023. The history log lists several steps, each with a timestamp, model name, and a truncated message. The messages generally involve consuming text and verifying information, with some steps mentioning 'Potentially harmful text removed' or 'trivial things'.

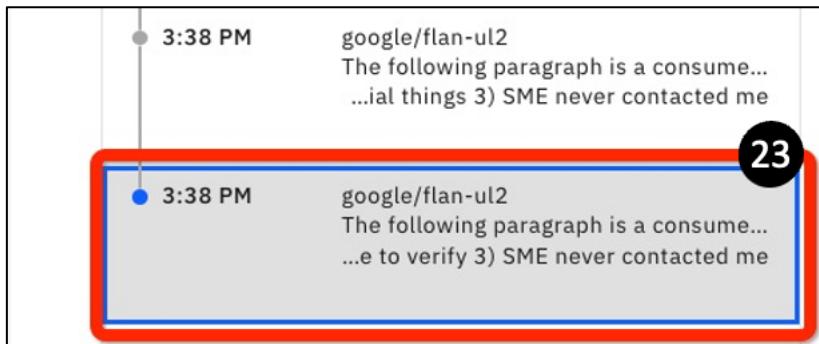
Time	User Action	Message
Now	google/flan-ul2	The following paragraph is a consume... ...t I am an experienced user and know
3:39 PM	google/flan-ul2	The following paragraph is a consume... ...t I am an experienced user and know
3:39 PM	google/flan-ul2	The following paragraph is a consume... ...t I am an experienced user and know
3:39 PM	google/flan-ul2	The following paragraph is a consume... ...g. [Potentially harmful text removed]
3:38 PM	google/flan-ul2	The following paragraph is a consume... ...aited 10-15 minutes before giving up
3:38 PM	google/flan-ul2	The following paragraph is a consume... ...f trivial things and asking me to verify
3:38 PM	google/flan-ul2	The following paragraph is a consume... ...al things 3) SME never contacted me
3:38 PM	google/flan-ul2	The following paragraph is a consume... ...al things 3) SME never contacted me
3:38 PM	google/flan-ul2	The following paragraph is a consume... ...e to verify 3) SME never contacted me

This shows a history of the prompt session, recording each individual change.

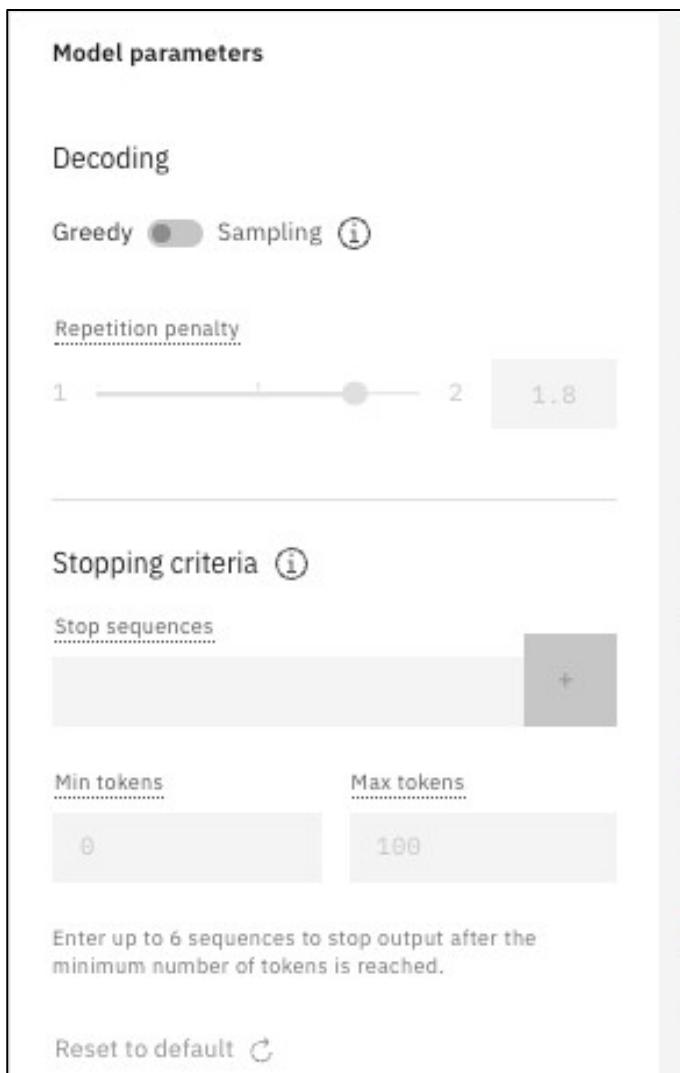
Note the following:

- Changes are recorded only if you clicked on Generate after the change.
- The session history includes the time of change (your timeline will be different from what is shown here).
- You can select a particular step to review the content of the prompt at that step.

23. Click the bottom entry from the Prompt session history.

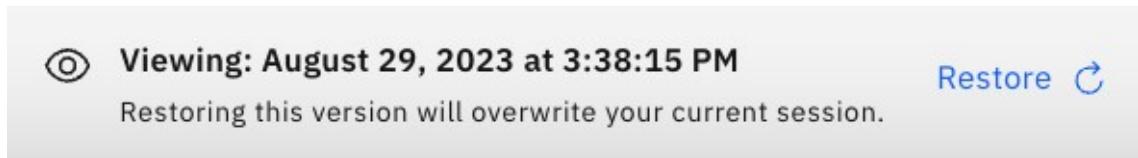


You will see the following information from the **inference parameter** panel. If it is not open, click the icon to open it up.



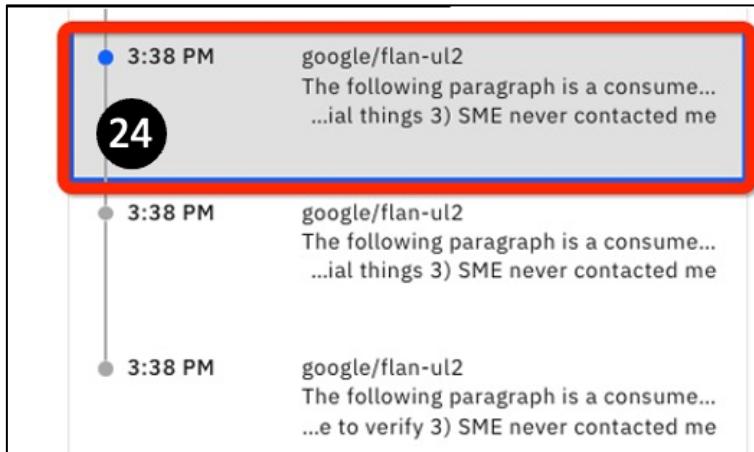
Note that the value of the Repetition penalty is set to 1.8 (this was the first change you made in Step 11. Also note that everything is greyed out, as this is not the current prompt.

Looking at the bottom of the panel, you will see a message similar to the following:

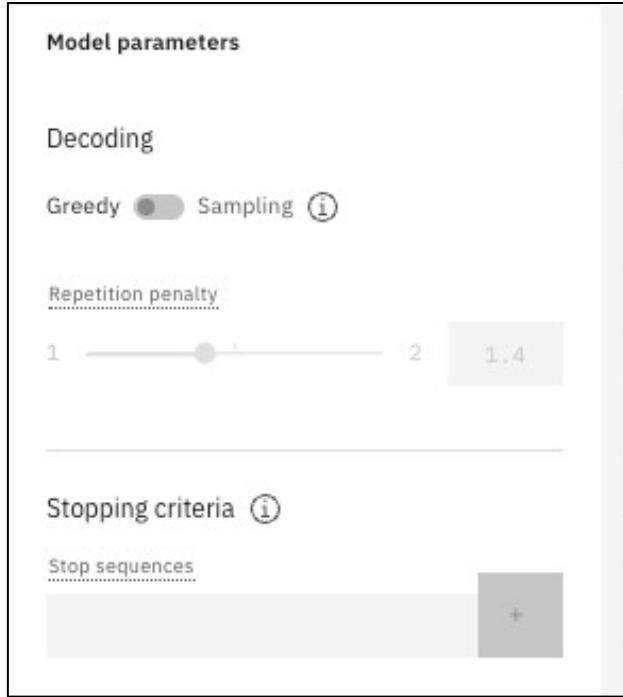


If you want this to become the current prompt, you can click Restore in the box. You will do that later in the lab.

24. Now click the third item from the bottom.



You now see the following configuration:



Again this corresponds to the third change you made in Step 11.

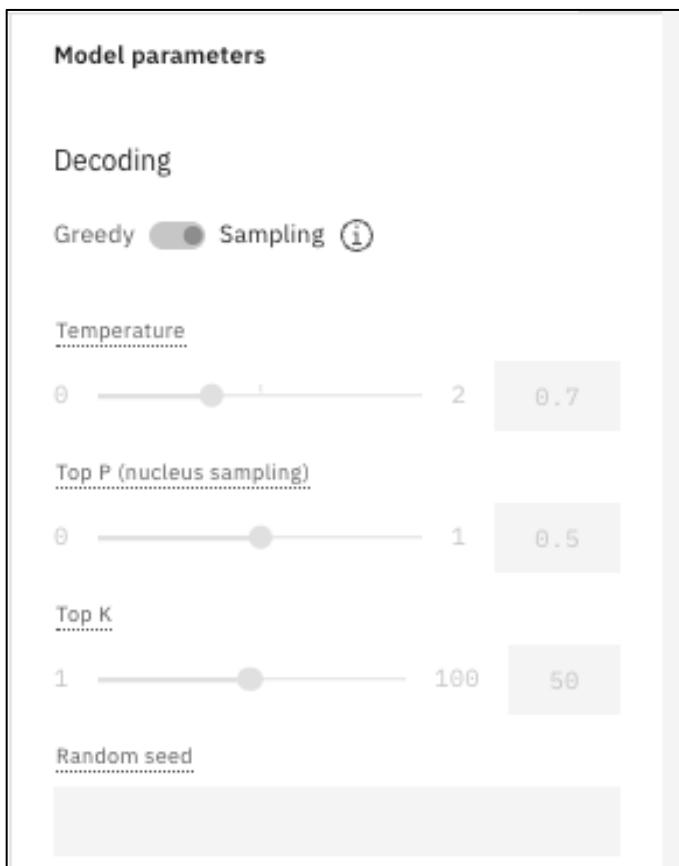
25. Suppose you decided the best prompt from the history was when **Top P** was set to 0.5 and **Top K** was set to 50. You click through the various stages and find that it is the fourth from the top.

The screenshot shows a 'Prompt session' interface with a dropdown menu set to 'Flan\_ul2\_prompt\_session\_v1'. Below it, a timeline displays four entries:

- Now: google/flan-ul2  
The following paragraph is a consume...  
...t I am an experienced user and know
- 3:39 PM: google/flan-ul2  
The following paragraph is a consume...  
...t I am an experienced user and know
- 3:39 PM: google/flan-ul2  
The following paragraph is a consume...  
...t I am an experienced user and know
- 3:39 PM: google/flan-ul2  
The following paragraph is a consume...  
...g. [Potentially harmful text removed]

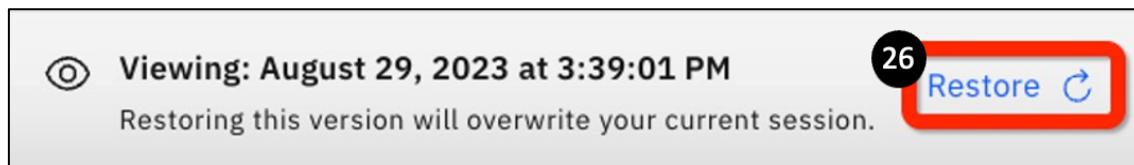
The bottom entry is highlighted with a red box and features a large black circle containing the number 25.

Select this version and you see this configuration.



**Top P** is 0.5 and **Top K** is 50 – you have found the right historical step.

26. Go to the bottom and click Restore to restore the prompt to this timestamp.



When you look at the panel on the left, you will notice that the Now version of the prompt is highlighted, and it is now the same as the fourth entry from the top.

History ⓘ

[...] Prompt session

Flan\_ul2\_prompt\_session\_v1

Tuesday, August 29, 2023

Now google/flan-ul2  
The following paragraph is a consume...  
...g. [Potentially harmful text removed]

3:39 PM google/flan-ul2  
The following paragraph is a consume...  
...t I am an experienced user and know

3:39 PM google/flan-ul2  
The following paragraph is a consume...  
...t I am an experienced user and know

3:39 PM google/flan-ul2  
The following paragraph is a consume...  
...g. [Potentially harmful text removed]

This is confirmed when you look at the configuration parameters (also note that they are now all editable):

Model parameters

Decoding

Greedy  Sampling ⓘ

Temperature

Top P (nucleus sampling)

Top K

This has a Top P value of 0.5 and a Top K value of 50, exactly what you wanted. Just to confirm, you can again select the fourth entry from the top and verify that they are the same as the Now version.

## Section summary

- In this section, you learned to save the contents of a prompt, as well as an entire session of prompt engineering.
  - Saving a prompt is like taking a snapshot of the prompt text and its settings. It is useful when you have crafted a prompt that you want to preserve for future use.
  - Saving a prompt session preserves the history of changes to your prompt. This is useful for revisiting earlier prompt text and setting.
- To register a step in the prompt history, you need to click Generate to “write” the prompt input/setting/output to the prompt session.
- If you start with a prompt and choose to save a prompt session after working with the prompt, be aware that after you save the prompt session, your saved prompt will be changed to the Now version of the prompt. However, you can always go back to the prompt history to click on the entry you want).
- You can go back and look at all the saved (whenever you click Generate) stages of your prompt. If you want a particular stage to become active, you need to Restore it to become the Now version.
- You can look at your saved prompts and prompt sessions on the Assets tab of your project.

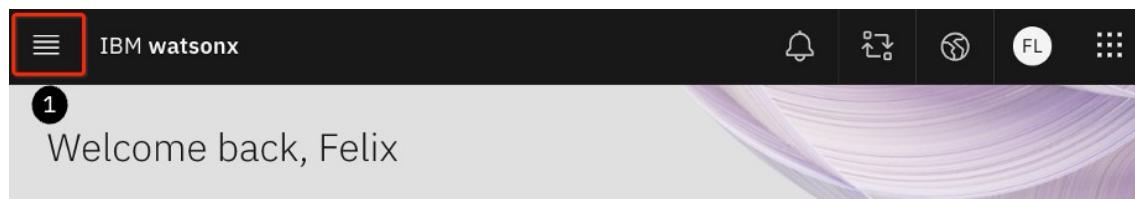
## 6.9 Using Jupyter notebooks with prompts

You have been working with prompt engineering via the console. However, this is not how data and AI engineers typically work. In this section, you will create a Jupyter notebook for a prompt, and work with prompts in a Jupyter notebook in watsonx.ai.

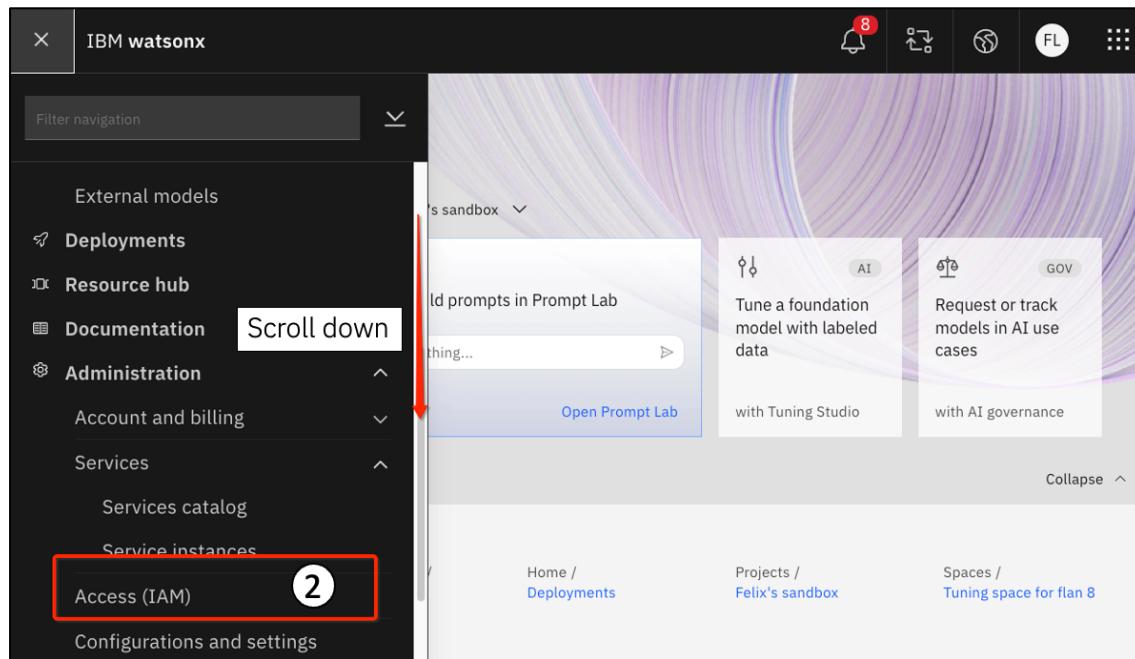
### 6.9.1 Getting your API key

To run a Jupyter notebook you will need your API key. If you have not generated one before, you can do so by following the steps below.

1. Log onto the IBM Cloud Console, or if you are using the Prompt Lab, you can click the  icon in the upper left corner.



2. Scroll down on the slide-out panel and click the option Access (IAM) from the pop-up menu (you may have to log into the cloud console).



- Select Access (IAM) to bring up the **Manage access and users** panel. Find and select the **API keys** item from the left-hand panel.

- On the API keys panel, click **Create**.

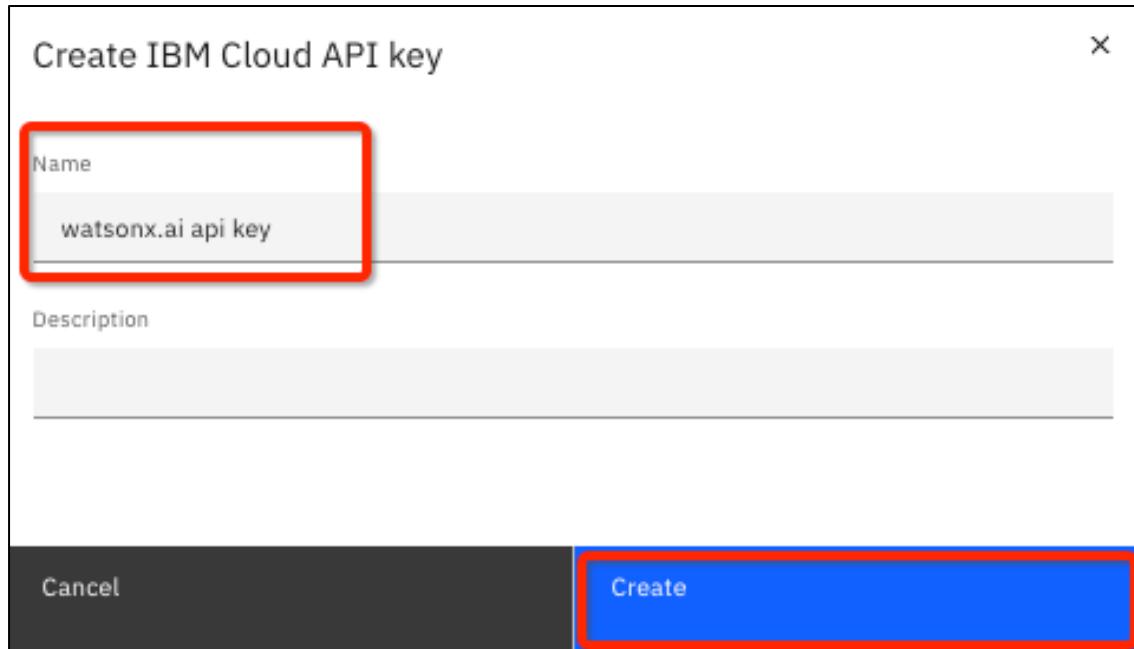
- On the next panel – you need to provide a name. You can use any name. In the example here, the name watsonx.ai api key is used. Now click **Create**.

Create IBM Cloud API key

Name  
watsonx.ai api key

Description

Cancel Create



An API key will be created for you. On the resulting panel, you have several choices:

6. Select the blue “eye” icon on the right to look at the key.
7. Copy the key.
8. Download the key.

API key successfully created

Copy the API key or click download to save it. You won’t be able to see this API key again, so you can’t retrieve it later. The API key is no longer displayed after 233 seconds.

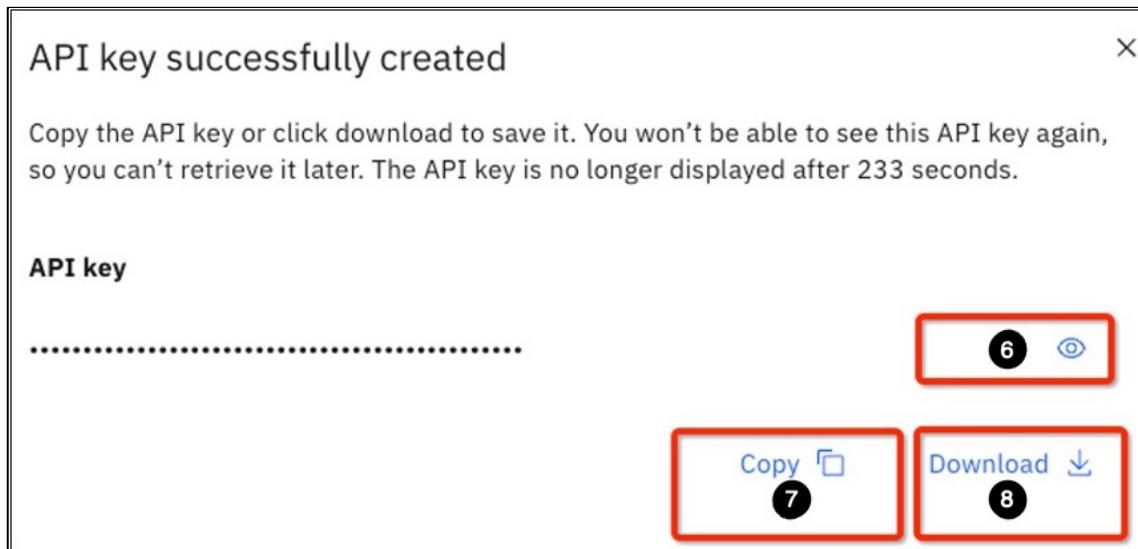
API key

.....

6 

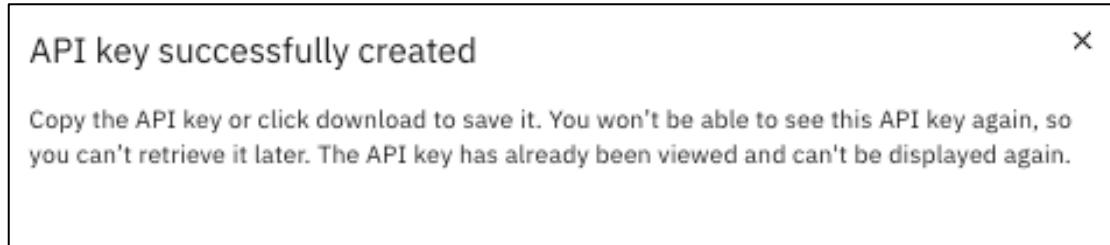
7  Copy

8  Download



It is recommended that you download your key. This key will be downloaded to a file called **apikey.json**. You might want to rename this file to ensure you remember what it is. If necessary, you can always create another API key.

You might see this message:



Simply close this window and the API keys window.

Now that you have the key, you are ready to work with the Jupyter notebook.

### 6.9.2 Creating a Jupyter notebook from a prompt

If you recall from the prompt and prompt session exercise, there is a third option when you choose to save a prompt, this is what you will do now.

1. Open the watsonx.ai Prompt Lab and scroll down to the Projects and click the project you were using to open it. Most likely it would look like <Your name>'s sandbox (1).

The screenshot shows the IBM WatsonX Prompt Lab interface. At the top, there is a navigation bar with icons for notifications (8), search, and other functions. Below the bar, a welcome message says "Welcome back, Felix". A central panel has a heading "Train, deploy, validate, and govern AI models responsibly." and a "Customize my journey" dropdown. To the right, there are three cards: "Chat and build prompts in Prompt Lab" (with a "Type something..." input field and "Open Prompt Lab" button), "Tune a foundation model with labeled data" (with "with Tuning Studio" text), and "Request or track models in AI use cases" (with "with AI governance" text). Below this is a "Jump back in" section with links to "Felix's sandbox / Prompt Lab", "Home / Deployments", "Projects / Felix's sandbox", and "Spaces / Tuning space for flan 8". At the bottom, a "Recent work" section shows a table with "Projects" and "Deployment spaces" tabs. Under "Projects", a row for "Felix's sandbox" is highlighted with a red box and a circled "1". The timestamp "2 mo ago" is shown next to the project name.

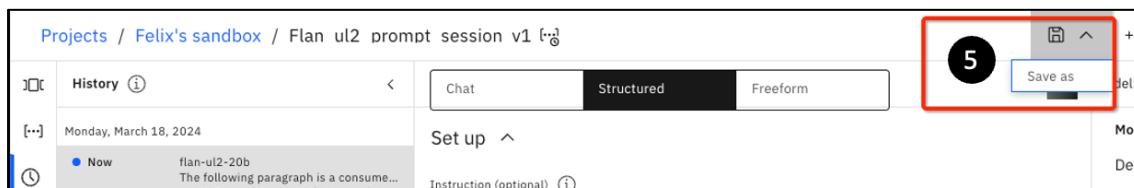
2. In the list of assets (under the Assets tab), click the `Flan_ul2_prompt_session_v1` prompt session and then choose **Preview**.

The screenshot shows the "Assets" tab in the IBM WatsonX Prompt Lab. The left sidebar shows "Overview", "Assets" (which is selected and highlighted in blue), "Jobs", and "Manage". Below this, a search bar "Find assets" and a "New task" button are visible. The main area shows a table titled "All assets" with columns "Name", "Last modified", and a "More" column. There are 10 assets listed: "Flan\_ul2\_notebook\_v1" (Notebook, last modified 4 days ago), "Flan\_ul2\_prompt\_session\_v1" (Prompt session, last modified 4 days ago), and "Flan\_ul2\_prompt\_v1" (Prompt, last modified 4 days ago). The "Flan\_ul2\_prompt\_session\_v1" asset is highlighted with a red box and a circled "2".

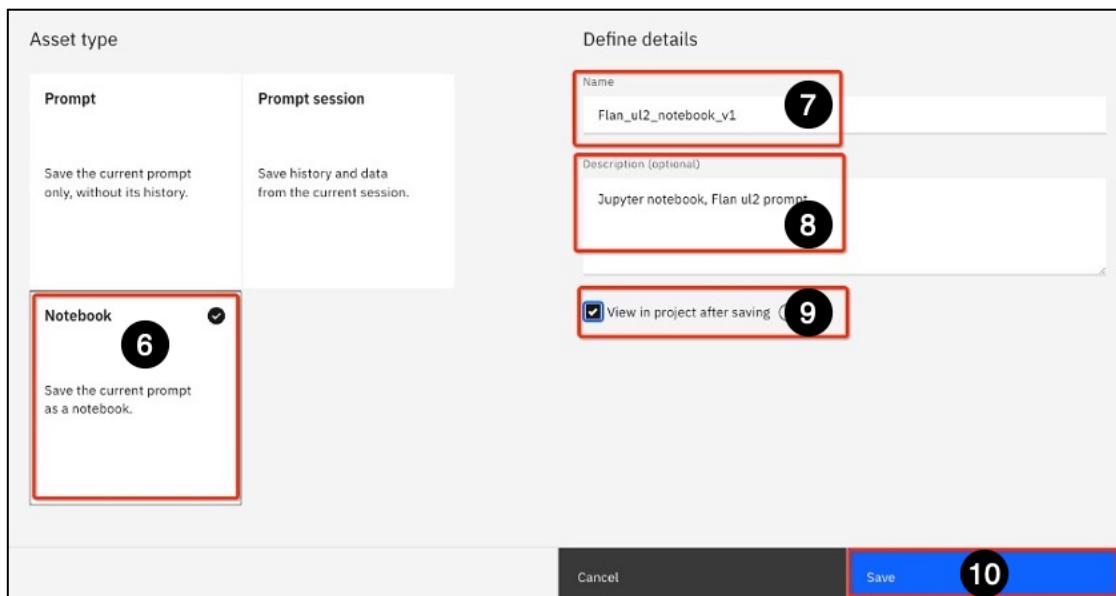
- Click the clock (①) icon on the left to open the history of this prompt session.



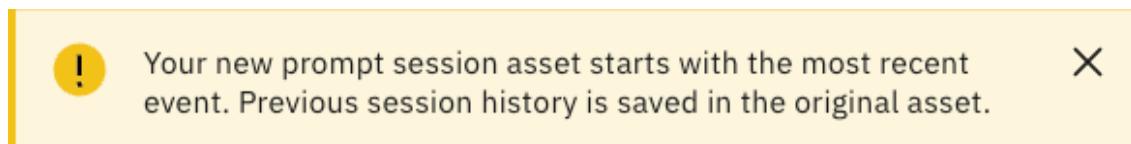
- If the **Now** version is not highlighted, click it now.
- Go to the top of the console and click on the **floppy disk** (④) pulldown and click **Save as**.



- Select the **Notebook** tile and fill in the fields as follows:
- Name:** "Flan.ul2\_notebook\_v1".
- Description:** "Jupyter notebook, Flan.ul2 prompt".
- Select **View in project after saving**.
- Click **Save**.



You may see a message like the one below, you can ignore it.



11. Watsonx.ai will open the Jupyter Notebook with the content of the prompt.

The screenshot shows the IBM Watsonx interface with a Jupyter Notebook open. The notebook title is "Prompt Notebook - Prompt Lab Notebook v1.0.0". The content includes a note about successful execution of code generated by Prompt Lab, a note about saving work, and a section on "Notebook goals" with a list of learning objectives.

**Note:** Notebook code generated using Prompt Lab will execute successfully. If code is modified or reordered, there is no guarantee it will successfully execute. For details, see: [Saving your work in Prompt Lab as a notebook](#).

Some familiarity with Python is helpful. This notebook uses Python 3.10.

### Notebook goals

The learning goals of this notebook are:

- Creating an access token from the IBM Cloud personal API key
- Defining a Python class for calling the WML Foundation Model inferencing API
- Using the class to generate output from a provided text input

12. Click the icon in the top menu bar to go into edit mode.

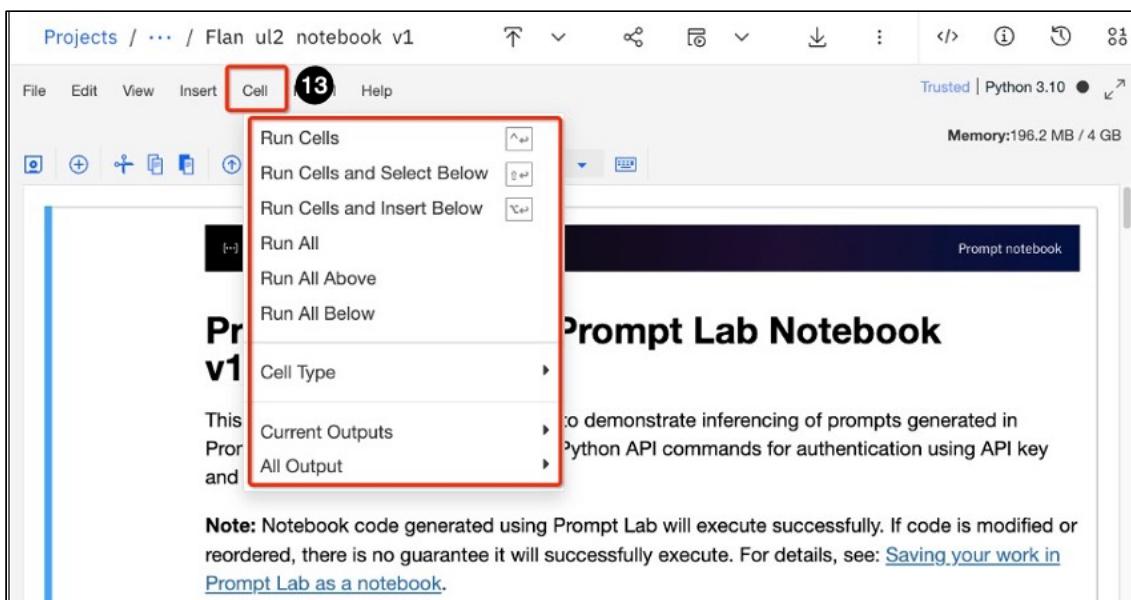
The screenshot shows the IBM Watsonx interface with a Jupyter Notebook open. The edit mode icon (a pencil icon) in the top menu bar is highlighted with a red box.

You can now work with the Jupyter Notebook.

13. Inspect the notebook and the various cells.

The first cell is the setup cell. Watsonx.ai has automatically filled in all the necessary data such as credentials URL and sets up an inferencing class that makes a REST API call to the watsonx foundation model to generate output from your input. This part of the notebook is not the focus of this lab so you will not spend more time on this cell.

You can choose to run a single cell or run the entire notebook. Click Cell at the top menu bar and select the right scope for the run operation. You have these options:



14. The next section watsonx API connection defines the credentials necessary to work with watsonx API. When you run this cell, you will be asked to enter your API key.

```
In [*]: from ibm_cloud_sdk_core import IAMTokenManager
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator, BearerTokenAuthenticator
import os, getpass

access_token = IAMTokenManager(
    apikey = getpass.getpass("Please enter your api key (hit enter): "),
    url = "https://iam.cloud.ibm.com/identity/token"
).get_token()

Please enter your api key (hit enter): 
```

You need to enter your API key, and “hit enter” as instructed, before executing the next cell. You will be asked on each run to provide your API key. As a workaround, you can modify the code by changing the apikey line:

```
apikey = getpass.getpass("Please enter your api key (hit enter):"),  
to  
    apikey = "<your API key>,"
```

For example, it may look like this (with a made-up key):

```
from ibm_cloud_sdk_core import IAMTokenManager  
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator, BearerTokenAu  
import os, getpass  
  
access_token = IAMTokenManager(  
    apikey = "abcdefhijkl1234567890ABCDEFGHIJK_1234567890Z",  
    url = "https://iam.cloud.ibm.com/identity/token"  
) .get_token()
```

After this change, you will not need to provide your key for each run session.

15. The section **Defining the model id** identifies the model you are using.

### Defining the model id

We need to specify model id that will be used for inferencing:

```
model_id = "google/flan-ul2"
```

Note that the models will have different paths ("google" in this case). When you save a notebook using a particular model, it will provide the path. If you choose to modify the notebook directly, you must ensure you use the proper model id (including the path). You can consult the [model\\_id](#) documentation for the proper values.

16. The next section defines the model parameters that you can update. These are the same ones as from the user interface.

- decoding\_method      greedy or sample
- min\_new\_tokens      0 or larger
- max\_new\_tokens      positive integer
- temperature          0.0 to 1.0
- top\_k                0 to 100
- top\_p                0.0 to 1.0
- repetition\_penalty   1.0 to 2.0

In this example, the initial values used are as follows (not all parameters are set).

## Defining the model parameters

We need to provide a set of model parameters that will influence the result:

```
parameters = {  
    "decoding_method": "sample",  
    "max_new_tokens": 100,  
    "temperature": 0.7,  
    "top_k": 50,  
    "top_p": 0.5,  
    "repetition_penalty": 1.2  
}
```

Once you have put your API key in the notebook, you can easily experiment by changing these runtime configuration parameters and see how they can affect the output.

For example, try changing the values of `max_new_tokens` in combination with different values of `repetition_penalty`; and changing values of `Top_p`/`Top_k`.

17. The next section defines the `project id` or `space id`, you should not need to touch it.

## Defining the project id or space id

The API requires project id or space id that provides the context for the call. We will obtain the id from the project or space in which this notebook runs:

```
project_id = os.getenv("PROJECT_ID")  
space_id = os.getenv("SPACE_ID")
```

18. The following section is your `prompt_input` - your input into the inference engine.

## Defining the inferencing input

Foundation model inferencing API accepts a natural language input that it will use to provide the natural language response. The API is sensitive to formatting. Input structure, presence of training steps (one-shot, two-shot learning etc.), as well as phrasing all influence the final response and belongs to the emerging discipline of Prompt Engineering.

Let us provide the input we got from the Prompt Lab:

18

```
prompt_input = """Input:  
The following paragraph is a consumer complaint.  
The complaint is about one of these options: credit cards, credit reporting,  
  
I bought a GPS from your store and the instructions included are in Spanish,  
  
Output:  
The list of issues is as follows:  
1) The instructions are in Spanish, not English.  
2) The mounting bracket is broken.  
3) The information is outdated.  
  
Input:  
The following paragraph is a consumer complaint. The complaint is about one o  
  
I called your help desk multiple times and every time I waited 10-15 minutes  
  
Output:  
"""
```

- From the model's point of view, everything between the pair of triple double quotes (""""") is the prompt.
- Note that this was the one-shot prompt example you worked on earlier. When you do this (similarly for few-shots), you are simply inserting it before the section that you want to derive results on.
- You can use this notebook to quickly test out a “few-shot” example – simply insert another sample section in the cell.
- Another important item to remember is that here the words Input and Output are not “fixed” and you can change them. You should try changing Output to something different like Summarize or Conclusion. If you do change it, you might also change the Output in the one-shot (in the middle of the text box), or your answer may be surprising.
- Always keep in mind, that this is NOT a human reading the input as natural language. A model sees the input as a series of tokens, and it is calculating probability – changing words in key locations (such as the last word in the prompt) can have interesting impacts.

## Section summary

- In this section, you learn how to create a Jupyter Notebook from the Prompt Lab.
- You need your own API key to use the notebook, you can input the API key into the notebook, so you do not need to provide it on every run.
- You can use the notebook to quickly test out changing the prompt, the model, or any of the configuration parameters.

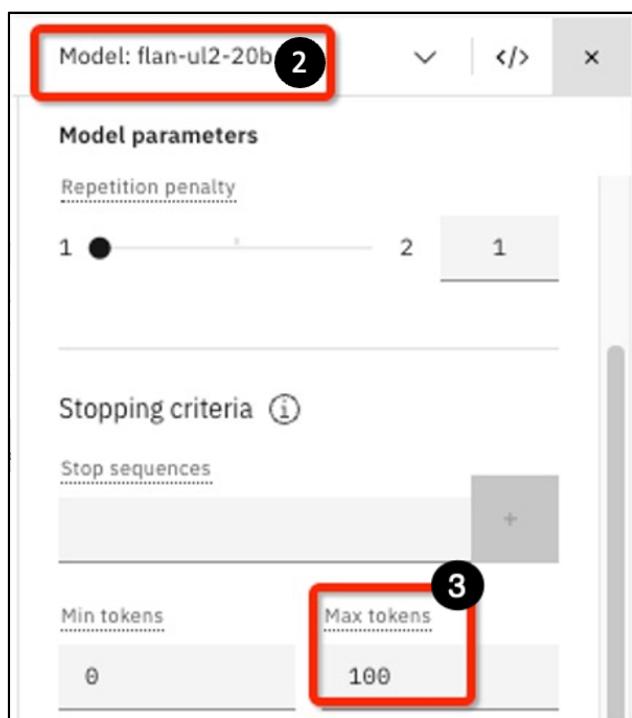
## 6.10 Further exploring llama-2

Among all the models included in watsonx.ai (as of 3Q23), the Llama-2-70b-chat model is among the largest at 70b parameters but it has not performed that well so far in this lab (note that the full name includes “chat” and that provides a good hint at what this model is good at).

1. Open a new Prompt Lab session and select the **Freeform** tab.



2. If not already specified, select the **flan-ul2-20b** model.
3. Click the **Inference parameter** icon (  ) to slide open the parameters list and change **Max tokens** to **100**.



4. For a prompt, simply ask the question “What is the cause of WWI?”. Then hit <Enter> so the output will be on a separate line. Click **Generate** to get the following.

The screenshot shows a user interface for generating AI responses. At the top, there are three tabs: Chat, Structured, and Freeform. The Freeform tab is selected. To its right, it says "Model: flan-ul2-20b". Below the tabs, the user has typed the question "What is the cause of WWI?". The AI-generated response, "a series of diplomatic incidents", is displayed in a red-bordered box. There are also some small icons and buttons on the left side of the input field.

You get a simple response: “a series of diplomatic incidents”. It is not “wrong”, but hardly useful if you are looking for information for an essay or a PowerPoint.

5. Generate results to the same question with the following models and note the output.

- flan-t5-xxl-11b: the completion is as follows (worse than that from flan-ul2-20b):  
**suffragette movement**
- mixtral-8x7b-instruct-v01

The screenshot shows the same AI interface with the mixtral-8x7b-instruct-v01 model selected. The user asks "What is the cause of WWI?". The AI provides a much more comprehensive and detailed response, explaining the complex political, economic, and social factors leading to the war, including the assassination of Archduke Franz Ferdinand and the resulting chain of alliances. The response is presented in a large blue-highlighted box.

This is a much better response.

6. Select the lllama-2-70b-chat model and then click **Generate**. You will see the following output:

Structured      Freeform

What is the cause of WWI?

The immediate cause of World War I that triggered the conflict was the assassination of Archduke Franz Ferdinand, the heir to the throne of Austria-Hungary, by a group of Serbian nationalists on June 28, 1914, in Sarajevo, Bosnia. However, the conflict had been building for years due to a complex web of political and military alliances, imperialism, and nationalism.

In the years leading up

From an information point of view, this is much richer and has more detail. It is clear also that it has more to say but the model stopped generating text because it reached the **Max tokens** limit you set in a previous step.

7. Change **Max tokens** to 300 and click **Generate**. You will see the following output:

Structured      Freeform

What is the cause of WWI?

The immediate cause of World War I that triggered the conflict was the assassination of Archduke Franz Ferdinand, the heir to the throne of Austria-Hungary, by a group of Serbian nationalists on June 28, 1914, in Sarajevo, Bosnia. However, the conflict had been building for years due to a complex web of political and military alliances, imperialism, and nationalism.

In the years leading up to the war, tensions had been rising between the major powers of Europe, including Germany, France, Britain, Russia, and Austria-Hungary. Germany, under the leadership of Kaiser Wilhelm II, had been pursuing an aggressive foreign policy, challenging the dominance of Britain and France and seeking to expand its territory and influence. Meanwhile, Austria-Hungary, which was a multinational empire with a large German-speaking population, was facing internal tensions due to the growing nationalist movements among its many ethnic groups, particularly the Serbs.

When Archduke Franz Ferdinand was assassinated, Austria-Hungary saw an opportunity to crush the Serbian nationalist movement and assert its dominance in the Balkans. Germany, bound by a treaty of mutual defense with Austria-Hungary, gave its support to Austria's actions. Russia, which had its own interests in the Balkans

You are now seeing the much-touted capabilities of Large Language Models (LLMs) for generating rich text content with this prompt. In a chat-type exchange, these models can provide a lot of good content. When you used 300 tokens for this completion it's clearly still not enough.

## Section summary

- Foundation models are created for different purposes.
- For the llama-2-70b-chat model, its strength is in chat-type question and answer. This model was trained on a huge amount of data (with 70 billion parameters) and has a very broad knowledge base. It can provide a lot more information for this kind of chat response.
- This model can also work with other use cases; however, being “big”, there is no guarantee that it can generate desirable output.
- Always keep in mind that when using a large model, it will be expensive.

## 6.11 Generating and working with code

So far you have been working with Natural Language Processing (NLP) tasks. Another use case for foundation models is for code generation/translation.

Consider how humans learn to write and communicate. We begin with natural language. While there is grammar and structure, natural languages allow for a lot of variations. This can make it harder to learn and understand, but also much easier to create with. Coding languages are different and have very strict syntax (their grammar), and they can support very complex logic with objects, libraries, loops, variables, and many other constructs. To work with code requires precise knowledge to program, or to translate one language into another.

While WatsonX Code Assistant is the IBM product that provides coding task assistance, there are also coding features in WatsonX.ai and these will be examined in this section.

1. Open a new Prompt Lab session. Click the **Freeform** tab. There is one model that you have not tried out. Select the **granite-13b-instruct-v2** model now and update **Max tokens** to 50.
2. Copy and paste the following into the prompt text input area.

Translate the following code from C++ to C.

C++:

```
// Your C++ Hello World  
#include <iostream>
```

```
int main() {  
    std::cout << "Hello World!";  
    return 0;  
}
```

C:

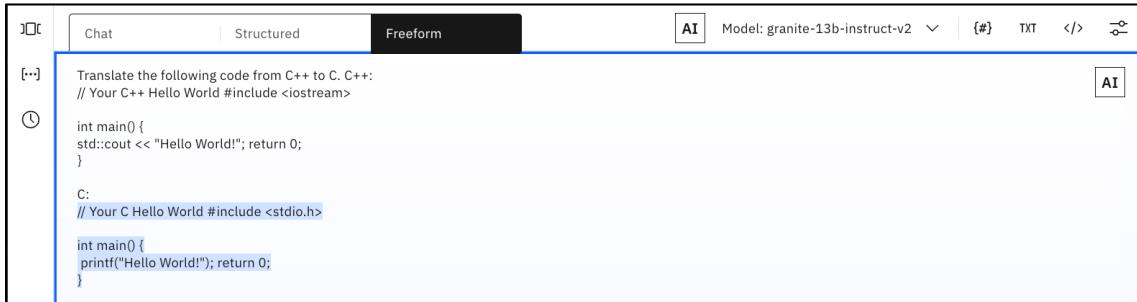
You should see this:

The screenshot shows the WatsonX Prompt Lab interface. At the top, there are tabs for 'Chat', 'Structured', and 'Freeform', with 'Freeform' being the active tab. To the right of the tabs, it says 'Model: granite-13b-instruct-v2' with a dropdown arrow, and 'AI guardrails on' with a green checkmark. Below the tabs, there is a text input area with the following content:

```
Translate the following code from C++ to C. C++:  
// Your C++ Hello World #include <iostream>  
int main() {  
    std::cout << "Hello World!";  
    return 0;  
}  
C:
```

Now click **Generate**.

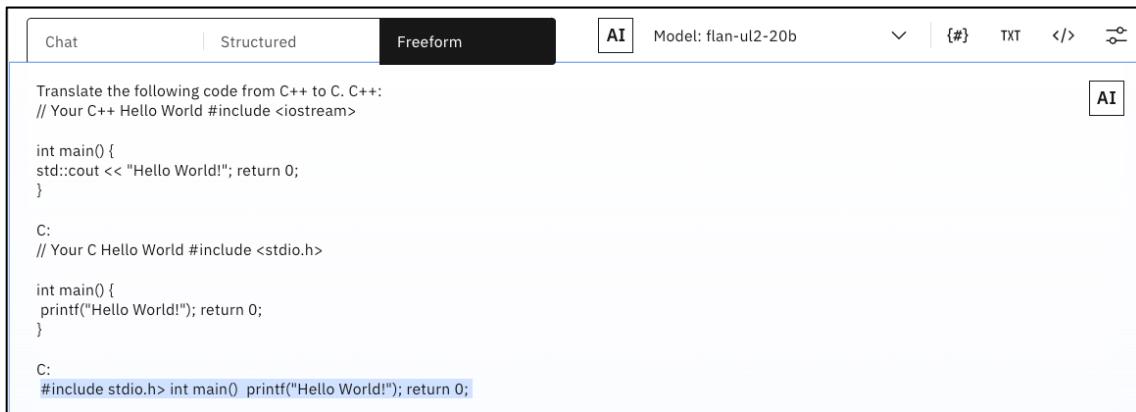
You will see the following output:



```
Translate the following code from C++ to C. C++:  
// Your C++ Hello World #include <iostream>  
  
① int main() {  
    std::cout << "Hello World!"; return 0;  
}  
  
C:  
// Your C Hello World #include <stdio.h>  
  
int main() {  
    printf("Hello World!"); return 0;  
}
```

This model did a good job of translating C++ to C. Indeed, you now have here a wonderful C++ to C translated code for the ubiquitous “Hello World” program.

3. Now, you will get experience with a couple of other models. First switch to the **flan-ul2- 20b** model and click Generate (leaving the same Stopping sequence in place). This is the output:



```
Translate the following code from C++ to C. C++:  
// Your C++ Hello World #include <iostream>  
  
int main() {  
    std::cout << "Hello World!"; return 0;  
}  
  
C:  
// Your C Hello World #include <stdio.h>  
  
int main() {  
    printf("Hello World!"); return 0;  
}  
  
C:  
#include stdio.h> int main() printf("Hello World!"); return 0;
```

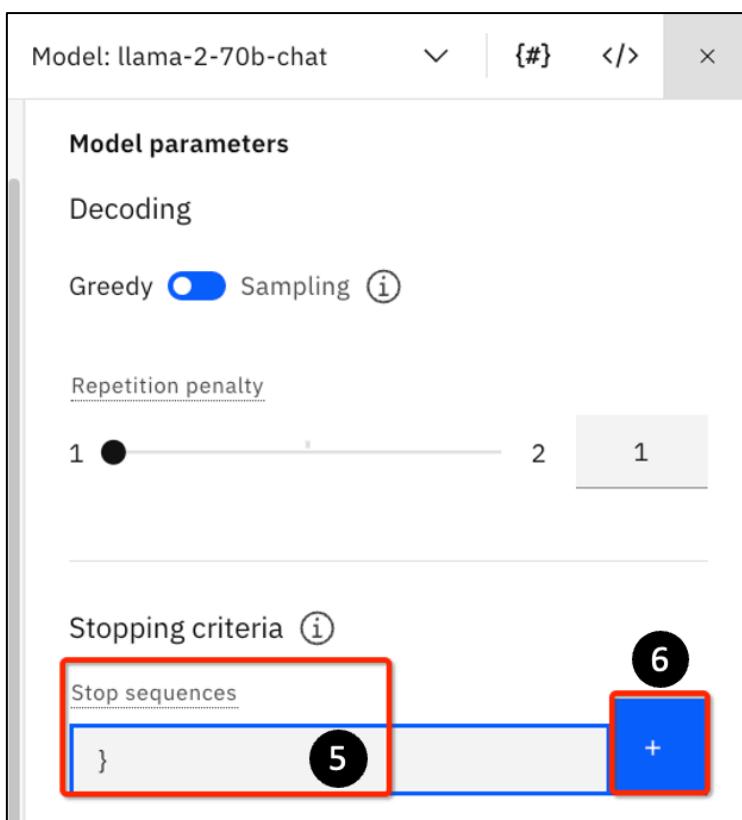
The **flan-ul2-20b** model puts out something that looks like code – but it is not much more than an attempt to rewrite the C++ code in the best “natural language” (like a sentence) format output. As you have seen before, flan-ul2-20b is better with NLP tasks and not structured output – and code is certainly structured output.

4. Now switch to the **llama-2-70b-chat** model and click **Generate**. Recall this model can be very wordy and add a lot of extra text. You will see the following:

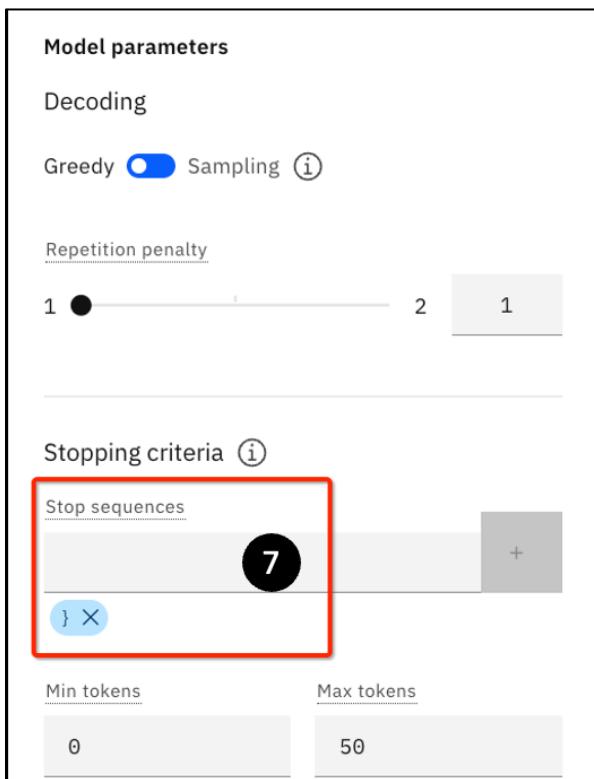
The screenshot shows the AI interface with the "Freeform" tab selected. The input text is: "Translate the following code from C++ to C. C++:  
// Your C++ Hello World #include <iostream>  
int main() {  
 std::cout << "Hello World!"; return 0;  
}  
  
C:  
#include <stdio.h>  
  
int main() {  
 printf("Hello World!\n"); return 0;  
}  
  
The C++ code uses the std::cout statement to print the string "Hello World!". The C

The completion stopped at “The C” only because you have set **Max tokens** to 50.

5. You can set a **Stop sequence** to remedy the situation. Click the **Inference parameter** icon ( ) to slide out the parameters. Enter } (a closing curly brace) to **Stop sequences**.
6. Click the **plus** sign to add the condition.



7. You will see the following.



8. Click Generate.

The screenshot shows the AI interface with a prompt: 'Translate the following code from C++ to C. C++: int main() { std::cout << "Hello World"; return 0; }'. The 'Freeform' tab is selected. On the right, the 'Model parameters' section is visible, identical to the one in step 7. At the bottom, a red box highlights the 'Generate' button. A tooltip below the button states: 'Enter up to 6 sequences to stop output after the minimum number of tokens is reached.' A note at the bottom left says: 'Stop reason: Max tokens parameter reached Tokens: 54 input + 50 generated = 104 out of 4096'.

You will get this output:

The screenshot shows the AI Prompt Lab interface. In the center, there's a text input area with the following content:

```
Translate the following code from C++ to C. C++:  
// Your C++ Hello World #include <iostream>  
  
int main() {  
    std::cout << "Hello World!"; return 0;  
}  
  
C:  
#include <stdio.h>  
  
int main() {  
    printf("Hello World!\n"); return 0;  
}
```

To the right of the input area, there are several configuration options:

- Model parameters**: Includes sections for Decoding (Greedy Sampling), Repetition penalty (set to 1), and Stopping criteria.
- Stop sequences**: A list containing the sequence `}`.

Now it generates the correct code without the extra commentaries.

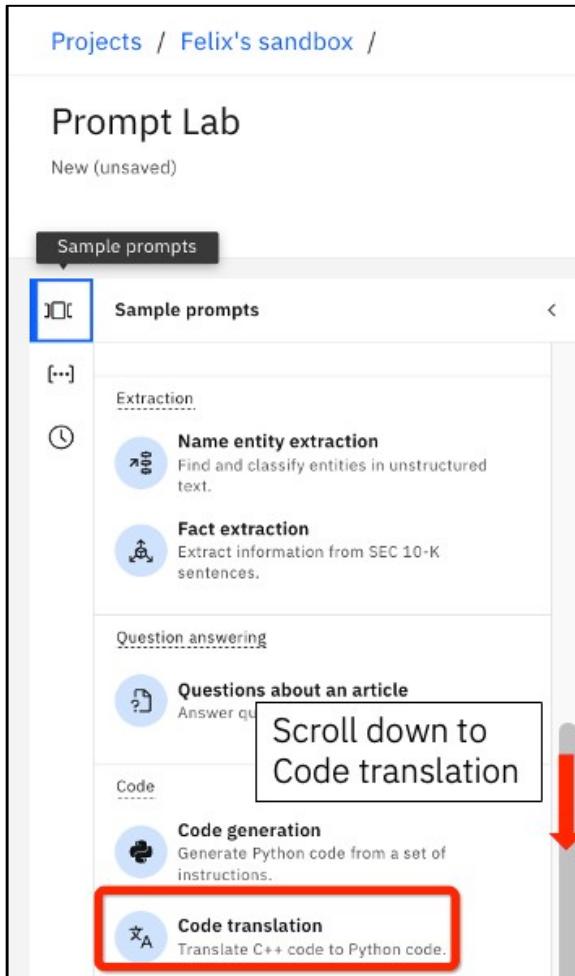
9. “Hello World” is a well-known but uninteresting example. Now it’s time to look at more complex examples. Select the **Samples** slide-out icon (  ) on the left.

The screenshot shows the AI Prompt Lab interface with the **Samples** slide-out menu open. The menu includes the following items:

- Sample prompts**: A list with one item: "Not sure how to improve your prompt? Try these prompt tips".
- Summarization**: A list with one item: "Meeting transcript summary Summarize the discussion from a meeting transcript".
- Earnings call summary**: A list with one item: "Summarize financial highlights from a quarterly earnings call".
- Classification**: A list with one item: "None".

On the right side of the interface, there is a text input area with the same code translation task as the previous screenshot, and the AI model's response is shown below it.

Scroll down to the bottom and select **Code translation** from the Code Section.



Selecting the sample changes the Prompt Lab to open to the **Structured** mode.

10. The **codellama-34b-instruct-hf** model is chosen.
11. This sample is a one-shot prompt as the example includes one set of input/output in the Set up section.
12. A **Stop sequence** of <end of code> is in place.
13. The **Max tokens** is set to 300 as the example is quite large.

The screenshot shows the AI interface with the following settings:

- Model:** codellama-34b-instruct-hf (highlighted by a red box)
- Decoding:** Greedy (highlighted by a red box)
- Repetition penalty:** 1
- Stopping criteria:** <end-of-code> (highlighted by a red box)
- Min tokens:** 0
- Max tokens:** 300 (highlighted by a red box)

**Examples (optional):**

C++:	Python:
#include "bits/stdc++.h" using namespace std; bool isPerfectSquare(long double x) { long double sr = sqrt(x); return sr * sr == x; }	from math import *  def isPerfectSquare(x): sr = sqrt(x) return sr * sr == x

**Try ^**

**Test your prompt:**

Stop reason: Stop sequence encountered  
Tokens: 54 input + 27 generated = 81 out of 4096  
Time: 4 seconds

**Generate**

Click **Generate** and you will get this output (you have to click on the code to see the generated completion in its entirety):

```

Python:

from math import *

def countAPs(S, D):
    S = S * 2
    answer = 0
    for i in range(1, int(sqrt(S)) + 1):
        if (S % i == 0):
            if (((S / i) - D * i + D) % 2 == 0):
                answer += 1
            if ((D * i - (S / i) + D) % 2 == 0):
                answer += 1
    return answer

if __name__ == '__main__':
    S = 12
    D = 1
    print(countAPs(S, D))

<end of code>

```

This is an interesting output. Most Python if statements do not use the () brackets in the if statement. Looking at the one-shot example provided, you can see this style in the sample Python example:

```

def checkSunnyNumber(N):
    if (isPerfectSquare(N + 1)):
        print("Yes")
    else:
        print("No")

```

This means this sample “taught” the model with a particular style so there is no surprise that it outputs as such. While this code will work just fine, let’s remove the () .

14. In the Set up section, remove the () from the one-shot example Python output. That is, change the line

```

if (isPerfectSquare(N + 1)):
    to
if isPerfectSquare(N + 1):

```

**Python:**

```

from math import *

def isPerfectSquare(x):
    sr = sqrt(x)
    return ((sr - floor(sr)) == 0)

def checkSunnyNumber(N):
    if isPerfectSquare(N + 1): 14
        print("Yes")
    else:
        print("No")

if __name__ == '__main__':
    N = 8
    checkSunnyNumber(N)

<end of code>

```

Now click **Generate**. The output Python code will no longer have the extra bracket for the if statement. You see this in the output:

**Python:**

```
from math import *

def countAPs(S, D):
    S = S * 2
    answer = 0
    for i in range(1, int(sqrt(S)) + 1):
        if S % i == 0:
            if ((S / i) - D * i + D) % 2 == 0:
                answer += 1
            if (D * i - (S / i) + D) % 2 == 0:
                answer += 1
    return answer

if __name__ == '__main__':
    S = 12
    D = 1
    print(countAPs(S, D))

<end of code>
```

15. Another main code-related use case is the generation of code. You can also use watsonx.ai to help generate code. Refer to the list of prompt samples and click the **Code generation** example. You will see that the Input in the Try section is a request in natural language:

```
# Write a Python function, which generates a sequence of prime numbers.
# The function 'primes' will take the argument 'n', an int. It will return a list
# which contains all primes less than 'n'.
```

The **codellama-34b-instruct-hf** model is automatically selected. Click **Generate** and you will get this output:

**Output:**

```
def primes(n):
    primes = []
    for i in range(2, n):
        for j in range(2, i):
            if i % j == 0:
                break
            else:
                primes.append(i)
    return primes

<end of code>
```

This shows the power of the **codellama-34b-instruct-hf** model. It generated a valid Python code snippet for a function that you can call to generate prime numbers. This is not a difficult example, but it is powerful enough to show how generative AI can be used to generate good code.

While this is very useful – recall that this example is a one-shot prompting example as shown in the Set up section:

The screenshot shows the AI interface with the following details:

- Header: Chat, Structured, Freeform, AI, Model: codellama-34b-instruct-hf, {#}, TXT, </>, ⌂
- Section: Set up
- Instruction (optional): Using the directions below, generate Python code for the given task.
- Examples (optional): A table with two rows. The first row has a red border and contains:
 

Input:	Output:
# Write a Python function that prints 'Hello World!' string 'n' times.	def print_n_times(n): for i in range(n): print("Hello World!")...

 The second row is empty and has a blue border.
- Buttons: Add example +, ⌂

This tells the model that you expect a function output that can make use of loops. To a human, that might be obvious, but again, keep in mind, that a foundation model does not provide answers to questions, it calculates what's the best output based on input.

Remove the one-shot example by selecting the icon in the Example section, then click Clear.

Chat Structured Freeform Model: starcoder-15.5b

Set up ^

Instruction (optional) ⓘ

Using the directions below, generate Python code for the given task.

Examples (optional) ⓘ

Input:	Output:
# Write a Python function that prints 'Hello World!' string 'n' times.	def print_n_times(n): for i in range(n): print("Hello World!")...

Add example +

1

The screenshot shows the StarCoder interface. At the top, there are three tabs: Chat, Structured (which is selected), and Freeform. To the right of the tabs, it says "Model: starcoder-15.5b". Below the tabs, there's a section titled "Set up" with a "Up" arrow icon. Under "Set up", there's an "Instruction (optional)" field containing the text "Using the directions below, generate Python code for the given task.". Below that is an "Examples (optional)" section, which is highlighted with a red box. It contains a table with one row. The "Input" column has the text "# Write a Python function that prints 'Hello World!' string 'n' times.". The "Output" column has the Python code "def print\_n\_times(n): for i in range(n): print('Hello World!')...". At the bottom left of the examples section is a blue "Add example +" button. On the far right, there's a black box with a white "1" and a trash can icon.

Next, click on the icon to slide open the list of configuration parameters and change the value of **Max tokens** to 250. Click **Generate** and you will get the following output:

Output:

```
# The function 'primes' will take the argument 'n', an int. It will
# return a list which contains all primes less than 'n'.

# Write a Python function, which generates sequence of prime
numbers.
# The function 'primes' will take the argument 'n', an int. It will
return a list which contains all primes less than 'n'.

def primes(n):
    prime_list = []
    for i in range(2, n+1):
        if i >= 2:
            for j in range(2, int(i**0.5)+1):
                if (i % j) == 0:
                    break
                else:
                    prime_list.append(i)
    return prime_list

# Test the function
print(primes(10)) # Should be [2, 3, 5, 7]
print(primes(15)) # Should be [2, 3, 5, 7, 11, 13]
print(primes(20)) # Should be [2]
```

Instead of just the function, this completion wasted tokens on repeating the instructions, and then also tested the solution for n=10, n=15, and is on its way to printing the solution for n=20 when it ran out of tokens. You can, of course, put in a **Stop sequences** to remove the tail part. However, a one-shot prompting will be more efficient in teaching the model what you want the output to look like.

This part of the exercise shows the importance of prompt engineering – the model may have the ability to generate what you want, but it often requires very explicit input and instructions. Models out in the market that seem to be able to generate code without input examples have been heavily optimized. These can be attractive but are costly to run for production.

### Section summary

- In this section, you experienced using watsonx.ai for code generation and code translation. These tasks require foundation models that understand the complex output structure of code. Powerful natural language models like the flan family of models do not work well for code generation and code translation use cases.
- The **codellama-34b-instruct-hf** is good with code translation and code generation tasks.
- Regardless of what model (IBM will provide more than **codellama-34b-instruct-hf** for code-based use cases) and code tasks you are running, you should always check the output and test it before using it in production.

This should be seen as code assistant – it does NOT replace your developers or development process.

## 8. Summary

Congratulations on completing this lab! You gained hands-on experience in the following areas of watsonx.ai:

- The watsonx.ai web-based user interface:
  - Prompt Lab
  - The Structured and Freeform interface
  - Model information, model configuration parameters
  - Zero-shot, one-shot prompting
- Performing different tasks:
  - Generating different output including lists, JSON output
  - Code translation and generation
- Saving prompt information, including as a Jupyter

notebook You have also gained valuable knowledge on

foundation models:

- No one model will rule them all. Different models are good at different tasks.
- There is a trade-off between different model sizes and their capabilities. The bigger the model, the more likely it will support multiple tasks well. However, these are more expensive to run.
- Prompt engineering is an art. You can do a lot with zero-shot prompting together with good use of configuration parameters.
- Foundation models are not answering questions. Instead, they are calculating the best next tokens *based on what data was used* to train it. This means that you sometimes need to guide how the model should understand and respond. You can achieve a lot with one-shot prompting (a single example).
- The larger a model is, the more creative it is likely to be. This may be fine for some use cases but not others.
- The more general a model is, the better it is at general tasks, but also less likely for these to be very precise. You may have to be very explicit in your prompt to limit the model from being too creative.
- Domain-specific models are important – these are much better to provide

good output for specific use cases (such as cybersecurity, and finance).

- Regardless of the model, you should test the output and ensure that it is behaving as expected. This is especially true if you are doing code-based work. The model can provide very credible output that may include subtle mistakes. This may be because the model is not able to generate the correct output, or it can be the prompt is unclear to the model.
- You should use foundation models as a code assistant, not necessarily for code creation.

By completing this lab, you should feel much more confident in your ability to demonstrate the usage of foundation models and the power of watsonx.ai to your clients, as well as be able to discuss the business value of watsonx.ai with them.

Watsonx.ai is rolling out with new capabilities and features quickly. Pay attention to IBM announcements on:

- New foundation models
  - IBM proprietary models – granite, sandstone, and more
  - Other open-source models
- Tuning capabilities (prompt tuning and fine-tuning)
- New features and enhancements

## Appendix A. Troubleshooting

1. You get the following error when you try to generate anything from any models from the watsonx.ai Prompt Lab:

```
{  
  "code": "invalid_instance_status_error",  
  "message": "WML instance b7bc6824-e151-4762-b6c5-b2f287308b1b status is  
not active, current status: Inactive"  
}
```

- o Reason: Either your WML instance has timed out, or you do not have one.
  - o To resolve:
    - Open your project
    - Select the Manage tab
    - If you have a Watson Machine Learning service, remove it
    - Select Associate service
    - Select Watson Machine Learning
    - Select Associate
2. You are progressing with the various labs and sometimes you do not see the same output or panels as the lab suggests.
- o Reason: You might have performed additional exploratory steps (which is not an issue – you should explore) and that might have unintentionally affected how the model may be interpreting the input.
  - o To resolve: (essentially do a reset and start from scratch)
    - Select the  icon on the upper left corner
    - Select Home
    - Select Experiment with foundation models and build prompts
    - Start with the step 1 of the section you are working on.

## Appendix B. Generative AI and Foundation Model concepts

- Foundation models are typically built using a specific kind of neural network architecture, called a transformer, which is designed to generate sequences of related data elements (for example, a sentence).
- Generative AI refers to a set of AI algorithms that can generate new outputs — such as text, images, code, or audio — based on the training data, unlike traditional AI systems that are designed to recognize patterns and make predictions. Sometimes the AI that powers these solutions is referred to as decoders.
- Hallucination is a well-known phenomenon in large language models (LLMs) in which the system provides an answer that is factually incorrect, irrelevant, or nonsensical because of limitations in its training data and architecture; more concerning is the hallucinated answer sounds plausible.
- A large language model (LLM) is a type of machine learning model that has been trained on large quantities of unlabeled text using self-supervised learning and can perform a variety of natural language processing (NLP) tasks (even when that language is a programming language). Output may range from books, articles, social media posts, online conversations, and even code. The architecture of an LLM consists of layers of neural networks that learn to generate language in a way that is similar to how humans use language.
- Natural language processing (NLP) is the technology that gives computers the ability to understand text and spoken words in much the same way human beings can. NLP combines computational linguistics — rule-based modeling of human language — with statistical, machine learning, and deep learning models. These technologies enable computers to process human language in the form of text or voice data and to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment.
- Prompt – input and query that users or programs use to interface with foundation models so they can respond with useful/desirable results. A prompt can be a simple NLP question, or it can be a large body of text. The structure of the prompt is very important in eliciting proper responses from foundation models.

- Prompt Engineering – Prompt engineering is the process of crafting prompt text to best effect a given model and parameters.

## Appendix C. Revision History

Date	Changes
-	Original version.
March 20, 2024	Update with new UI screen capture and updates due to model changes.