BYOD lab: Advanced AI Insights Leveraging Huggingface LLMs as Part of a Multi-Model Approach for IBM Z and LinuxONE

Steve Warren
STSM, Principal Z Solution Architect
IBM Worldwide System Center
swarren@us.ibm.com

Purvi Patel
Senior AI Client Engagement Leader
AI on IBM Z & LinuxONE
purvi@us.ibm.com

Table of Contents

Notices and disclaimers

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information.

This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity. IBM products and services are warranted per the terms and conditions of the agreements under which they are provided. The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

IBM products are manufactured from new parts or new and used parts.
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled,
isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to ensure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements

that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Notices and disclaimers (Continued)

Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

1. Overview and Lab objectives

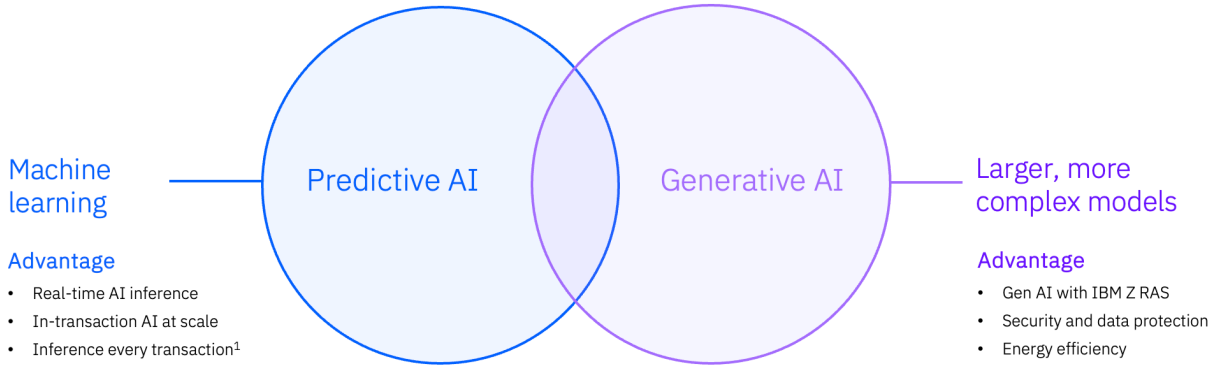
1.1 Introduction: IBM z17 & multiple model AI at speed and scale

The IBM z17 platform ushers in a new era of enterprise AI, redefining what's possible for mission-critical systems. While it continues supporting predictive AI at scale and speed to provide in-transaction insights, it delivers hardware capabilities that are designed to support complex large language models (LLMs) which allow to deploy Generative AI use cases securely and in energy efficient way. (see figure below)
Predictive AI that encompasses machine learning and deep learning models remains essential. Financial institutions, for example, continue to rely on predictive AI for use cases such as fraud detection and anti-money laundering. These models are not being replaced but are increasingly being combined with encoder-based large language models (LLMs) that excel at analysing and extracting insights from structured and unstructured content, rather than generating it. When combined with predictive AI, they **increase AI robustness for a better business outcome** and enhance accuracy in scenarios where confidence levels are uncertain—such as in fraud detection.

Both worlds of predictive AI and Large Language Model deployments can be combined in a hybrid approach called multiple model approach to deliver faster, more accurate results and more insightful predictions than any single model can accomplish alone

By "tag-teaming" models based on the confidence and complexity required for each individual case, IBM z17 helps strike the optimal balance among latency, efficiency, and accuracy. The result is an AI strategy that maximizes insights and performance, using the best-fit model for each workload while maintaining enterprise-grade speed and reliability.

This lab will guide you through the principles and practical implementation of multiple model AI on the IBM z17 platform, demonstrating how these innovations can dramatically improve prediction accuracy and business value.



Multiple-model AI at speed and scale

Combine machine learning and LLMs to achieve higher prediction accuracy and expanded insights

1.2 Intelligent and Automated Underwriting - Medical Insurance

The lab showcases a use case example related to intelligent and automated underwriting for a medical insuracnce.

Business Scenario:

Insurance underwriting is a critical process and fundamental mechanism that insurance companies use to assess and assess risk of insuring an application, thereby enabling the determination of appropriate premiums and appropriate policy terms..

- Underwriters need to analyse factors included in structured health data like labs, vitals, age, BMI, financial history and unstructured clinical notes.

Manual review is slow and inconsistent, which can cause variable risk assessment,sub-optimal pricing, and long processing times

Multi-model Solution on z17:

- To avoid the manual processing of unstructured data, an LLM model i.e. ClinicalBERT reads the applicant's clinical note to extract features such as smoking status, alcohol consumption , and medication/disease categories.
- A Random Forest model combines these extracted features with structured labs/ vitals to produce calibrated approval probabilities and a final decision.

The AI driven insights are provided along with the original inputs, so that underwriters can understand what drove the result.

Products leveraged:

- AI Toolkit for IBM Z and LinuxONE

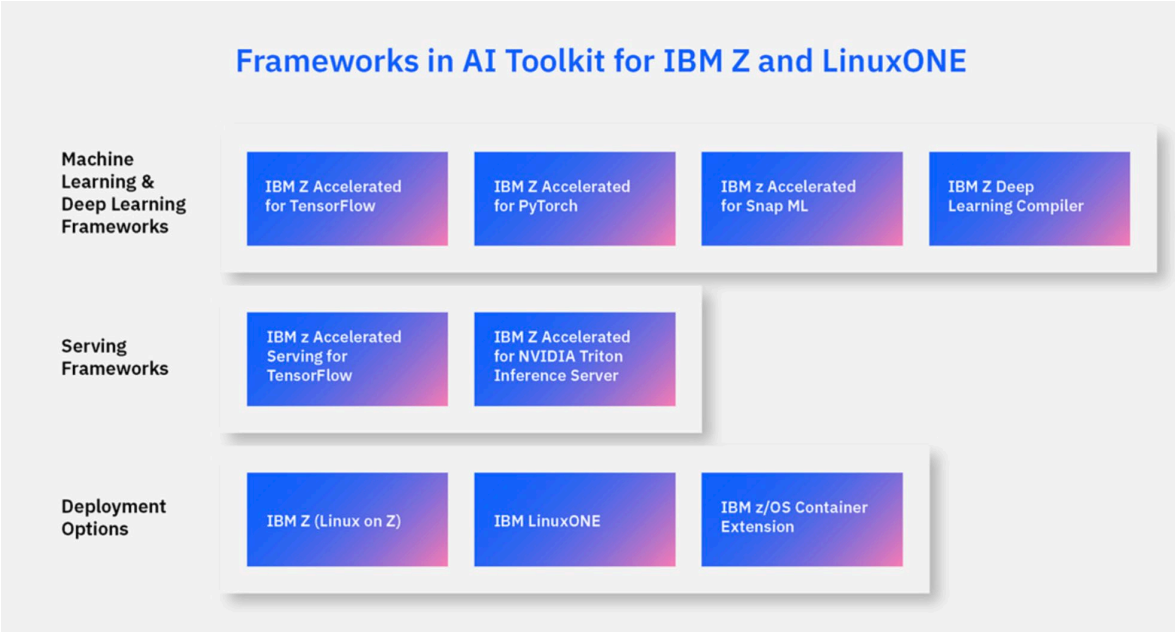
AI Models:

- Hugging face framework – Clinical_BERT LLM (trained off-platform and imported to IBM Z)
- Random Forest

1.3 AI Toolkit for IBM Z and LinuxONE

AI Toolkit for IBM Z® and LinuxONE is a bundle of popular open-source AI frameworks and libraries that are optimized to leverage Telum on-chip AI accelerator and highly efficient for faster inference and scoring of AI models. Accelerate AI adoption using certified containers, integrated accelerators, and dedicated expert support. These frameworks use on-chip AI acceleration in z16®, LinuxONE 4, z17® and LinuxONE 5.

The AI Toolkit provides IBM Elite Support (within IBM Selected Support). In addition, IBM Secure Engineering process is leveraged to scan and vet the open-source AI serving frameworks and IBM-certified containers for security vulnerabilities and validate compliance with industry regulations.



As part of this lab, we will perform the following steps:

1. We will run a pre-built hugging face container.
2. We will explore the content of a notebook that implements and deploys an encoder LLM BERT based model that is designed to extract key insights from the unstructured data.
3. We will explore the content of a notebook that builds and trains a Random Forest based model, which is intended to provide the final predictions using the combined data.
4. We will run the step-by-step guided inferencing notebook using PyTorch from AI toolkit for IBM Z.

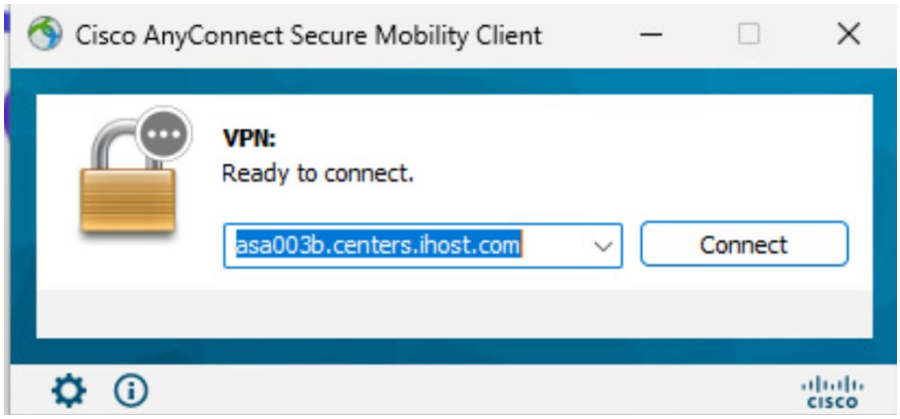
2. Let's connect to the system and IBM Z lab environment

2.1 Open Windows Powershell and access the IBM Z environment

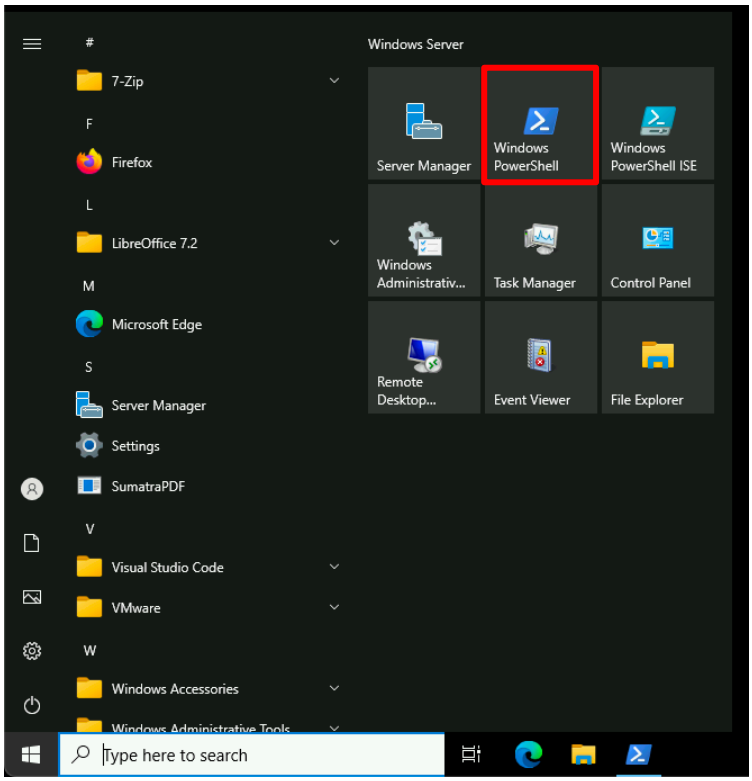
NOTE: To Copy and Paste text from Lab desktop into Bastion Server desktop, use the 'send text' feature listed on the top of screen.

1. Click on Any connect VPN on desktop, connect to below address and enter your provided VPN credentials.

```
asa003b.centers.ihost.com
```



2. Click Windows start button in lower lefthand corner of your virtual desktop and select Windows Powershell from the quick menu. You will be typing all subsequent commands from the Windows Powershell command line.



3. Use your IBM Z Container Extensions (zCX) login credentials that have been assigned to you. Enter the zCX UserID given to you as part of the following ssh command. Replace <zcxwsnn> with the UserID provided to you (e.g. zcxws01)

```
ssh <zcxwsnn>@129.40.117.162 -p 8022
```

When prompted for the password, enter the zCX password provided to you. You are now logged in.

```
EX235N01:MLZUSR:/u/mlzusr: >ssh zcxws15@129.40.117.162 -p 8022
zcxws15@129.40.117.162's password:

Welcome to the IBM z/OS Container Extensions (IBM zCX) shell that provides access to Docker commands.
For more information on how to use this shell to execute Docker commands refer to IBM
Last login: Sun Aug 24 09:59:38 2025 from 10.1.1.1
Sudo only permits specific User Management functions.  See additional documentation for details.


zcxws15@zlp1zcx1:~$
```

2.2 Setup Hugging Face container in zCX

A Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It supports multiple programming languages, like Python, making it a versatile tool for data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more. Jupyter Notebooks are widely used in data science, machine learning, and scientific computing.
Docker is a runtime environment that uses OS-level virtualization to deliver and host software in packages calls containers. zCX provides a Docker Engine running on top of Linux running in a z/OS address space (a zCX server).
We have already pre-built a Docker Container Image (an immutable copy of software that will be able to run as a live container) called huggingface_z17 for Hugging Face. This will allow you to quickly load the pre-trained model and test it.
Hugging Face is an open-source AI platform and community that provides easy access to powerful pre-trained models, datasets, and tools for natural language processing and more. It's especially known for its library of large language models (LLMs) and transformer architectures like BERT and GPT, which enable state-of-the-art AI capabilities in text, vision, and audio tasks.

Issue <`docker images`> command to view all images on this system and make sure that huggingface_z17 is part of the list.

```
zcxws15@zlp1zcx1:~$ docker images
REPOSITORY                                                    TAG       IMAGE ID       CREATED         SIZE
huggingface_z17                                               latest    e3a4c52f221b   22 hours ago    5.41GB
9.114.195.227:5001/huggingface/ubuntu-2204_huggingface_nnpa   latest    11b939c7fabd   6 weeks ago     3.87GB
ibm_zcx_zos_ssh_cli_image                                     latest    56f2d04f527d   5 months ago    428MB
ibm_zcx_zos_cli_base                                          latest    0c3ca6ee3d5b   5 months ago    428MB
icr.io/ibmz/ibmz-accelerated-for-nvidia-triton-inference-server   1.2.0     8c0261c66746   14 months ago   3GB
icr.io/ibmz_hf/hugging-face                                   1.0.0     346d8224fb06   15 months ago   2.43GB
huggingface                                                   latest    6ccb118cc21f   17 months ago   2.6GB
hf-stable-2                                                   latest    5fdff7a02241   18 months ago   2.4GB
icr.io/ibmz/openjdk                                           latest    2c6d55ec8d5b   20 months ago   454MB
jupyter_note                                                  latest    00eb106ca9df   24 months ago   3.13GB
tf2onnxdemo                                                   latest    6b3bdae382d8   2 years ago     6.02GB
tf2onnxdemo                                                   v2        6b3bdae382d8   2 years ago     6.02GB
ubuntu                                                        latest    5f3ff41546d1   2 years ago     73.4MB
icr.io/ibmz/tritonserver-s390x                                <none>    bb2e841bac4f   2 years ago     5.79GB
icr.io/ibmz/tensorflow                                        <none>    6c8984af913a   2 years ago     2.27GB
notebook-distribute                                           latest    be9efdec520c   3 years ago     125MB
9.114.195.227:5002/tensorflow-serving                         2.7       28c3b1172bd6   3 years ago     376MB
tensorflow-serving                                            2.7       28c3b1172bd6   3 years ago     376MB
icr.io/ibmz/uwsgi-nginx-flask                                 1.4.0     9287148d42b4   3 years ago     125MB
icr.io/ibmz/ubuntu                                            20.04     62186f1392d6   3 years ago     69.4MB
icr.io/ibmz/jupyter-notebook                                  6.4.5     e7b441088e73   3 years ago     1GB
icr.io/ibmz/gcc                                               11.2.0    e2601439a983   3 years ago     1.12GB
icr.io/ibmz/portainer                                         2.0       d9921b4c6706   4 years ago     2.42GB
```

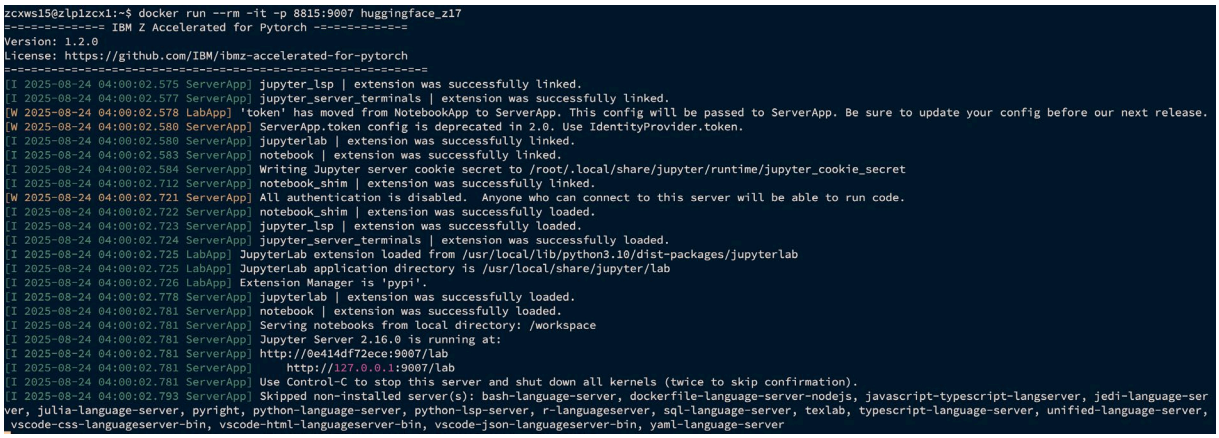
2.3 Hugging Face Docker image with Jupyter Notebook

In this step. you are going to create the running container based on the container image that you have just viewed. You will create your own container using that container image. To start up the container, we will use the docker run command that will initiate the step of container creation as well as container start up.

Issue the following command to start a new Jupyter Notebook container, from image, huggingface_z17 and attach your terminal to it (-it). You will need to substitute <nn> with the last two digits of your zcxwsnn userid that you used to ssh into zCX.

< `docker run --rm -it -p 88<nn>:9007 huggingface_z17` >

For example, if your user ID was zcxws15, you would specify 8815 as seen here. This will give you a unique port number that is mapped internally to Jupyter's port 9007.



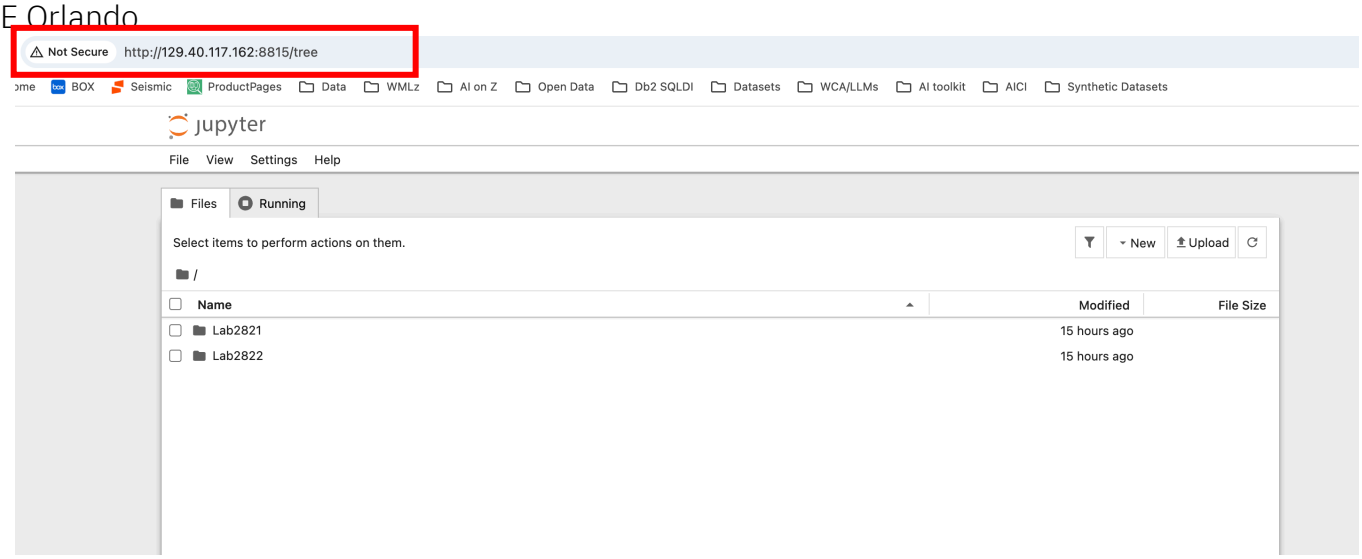
Your personal copy of the Jupyter Notebook container is running at the port you designated. Copy the line highlighted in the red box above. You will use this later to launch the Jupyter Notebook.

2.4 Launch your Jupyter Notebook in zCX

1. Launch another browser session or tab from within your virtual desktop.
2. Change the 127.0.0.1:8888 hostname of the URL to be the zCX DVIPA IP address that you used to SSH and the port that you specified on the docker run command.

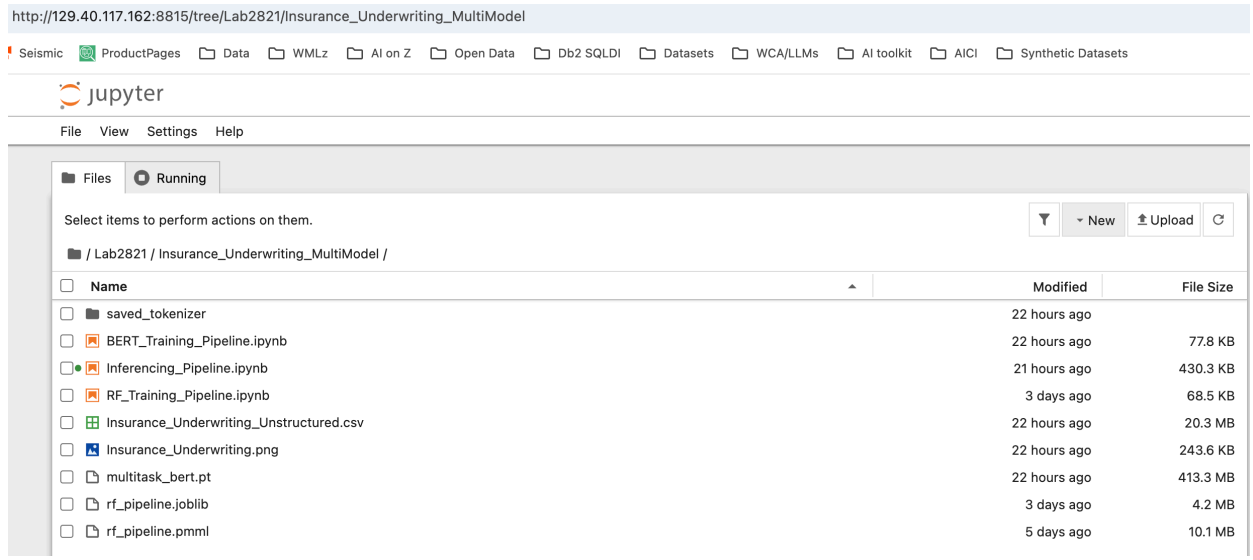
   For example: if you are zcxws15, change 127.0.0.1:8888 to **129.40.117.162:8815**.

   This should open the Jupyter Notebook application that shows two lab folders.



3. Open your lab folder, Lab2821 for AI toolkit for IBM Z and LinuxONE lab.

4. Open **Insurance_Underwriting_MultiModel** folder.

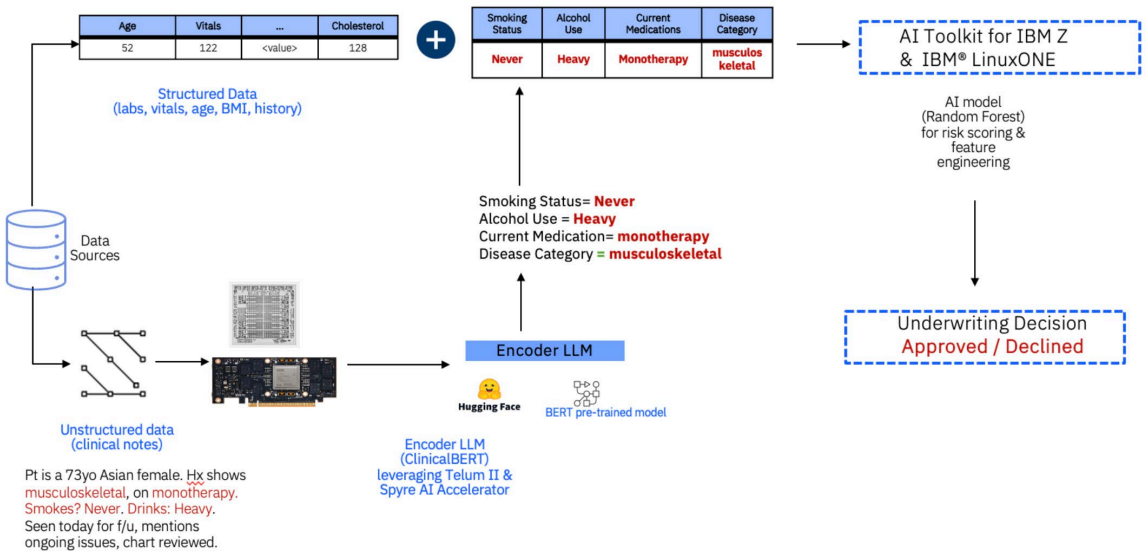


Throughout this lab, you will be using this notebook. Jupyter Notebooks include different parts of the code in a series of so called cells. To execute the code contents of a cell, “Tab” to the cell. Once it is highlighted, you can press “Shift-Enter” on your keyboard to execute the cell. If there is any output related to the cell execution, it will be displayed below the code after execution. The output could be data visualization, a result of calculations, or empty if the execution is not expected to deliver any output. It could be also listing any errors during the code processing. A cell that is being executed shows a [*] close to the top left of the cell. Once executed it shows a number that usually corresponds to the sequence of cells execution. You can then tab to the next cell and trigger its execution.

3. Making the predictions using BERT + Random Forest pipeline

In the scope of this lab, we have trained a Random Forest model and fine-tuned a ClinicalBERT LLM model.

- ClinicalBERT is an encoder LLM model that reads the applicant's clinical note to extract features such as **smoking status, alcohol consumption, and medication/ disease categories.**
- A Random Forest combines these text-derived features with structured labs/ vitals to deliver calibrated approval probabilities and a final decision.
- The AI insights are presented with the combined inputs, so that underwriters can see what drove the result.



3.1 Review the training notebooks

1. Open the **BERT_Training_Pipeline** notebook. Review the training code and understand the training process by reading the comments provided.

   NOTE: *Don't run the BERT training notebook, since it will take some time to train the LLM. We have already trained the model and saved it in the notebook.*

   Understanding BERT and Bio_ClinicalBERT

   BERT (Bidirectional Encoder Representations from Transformers) is a powerful pre-trained language model developed by Google. It revolutionized Natural Language Processing (NLP) by introducing a bidirectional training approach, allowing to understand the context of a word based on all other words in a sentence, rather than just sequential context.

Bio_ClinicalBERT is a specialized version of BERT, further pre-trained on a vast amount of biomedical and clinical text (e.g., PubMed abstracts, clinical notes). This domain-specific pre-training enables Bio_ClinicalBERT to better understand medical terminology, clinical concepts, and the nuances of

healthcare language, making it highly effective for tasks like medical text classification, named entity recognition in clinical notes, and more. It leverages the Transformer architecture, which uses self-attention mechanisms to weigh the importance of different words in a sequence.

We will be using **Bio_ClinicalBERT** for our use case. Extracted features from this step are smoking_status, alcohol_use, current_medications_category and disease_category.
Once the model is trained, it is saved with the name multitask_bert.pt.

2. Now, open the **RF_Training_Pipeline** notebook that is intended to train a Random Forest model leveraging the structured health lab features and the extracted features from unstructured clinical notes. In data science, those columns that impact the predictive result and used for model building are referred to as features.

   Run the notebook step by step, cell by cell. A cell that is being executed shows a [*] close to the top left of the cell. Once executed it shows a number that usually corresponds to the sequence of cells execution. You can then tab to the next cell and trigger its execution.

```
# Load dataset
data = pd.read_csv("Insurance_Underwriting_Unstructured.csv")
data.head()
```

	patient_id	visit_id	visit_date	age	sex	bmi	race	smoking_status	alcohol_use	current_medications_category	...	alt	ast	tsh	free_t4
0	P000000	P000000_V1	01/12/25	80	M	32.4	Black	Never	Occasional	monotherapy	...	32.54	29.18	3.03	1.32
1	P000001	P000001_V1	25/12/24	73	F	20.6	Asian	Never	Heavy	monotherapy	...	28.29	27.02	1.76	1.34
2	P000002	P000002_V1	25/04/25	18	M	25.0	Black	Former	NaN	monotherapy	...	26.12	30.11	2.95	1.37
3	P000003	P000003_V1	10/07/22	64	F	19.4	Hispanic	Former	Heavy	polypharmacy	...	29.58	22.70	2.36	1.24
4	P000004	P000004_V1	12/10/23	66	M	27.8	Asian	Former	Heavy	polypharmacy	...	28.68	31.81	2.32	1.52

5 rows × 38 columns

A Random Forest combines the text-derived features with structured labs/ vitals to produce calibrated approval probabilities and a final decision.

3.2 Load the pre-trained Clinical BERT and Random Forest models and run inference

After having reviewed the pre-trained models, the next step consists of load them for final inferencing/ predictions.
Open the **Inferencing_Pipeline** notebook. The notebook clearly explains the Insurance Underwriting use case leveraging the multiple model AI architecture.

You will execute each cell in Jupyter notebook and review the output. [Press SHIFT + Enter for each cell]

1. Import required libraries for inferencing.

```
# 1. Imports
import torch
import torch.nn as nn
import pandas as pd
import json
import numpy as np
from transformers import BertTokenizer, BertModel
from pypmml import Model
from IPython.display import HTML, display
import warnings
warnings.filterwarnings("ignore")
```

2. Load the pre-trained BERT and RF models along with label_mappings and saved_tokenizer.

```
# ------------------------
# Config
# ------------------------
MAX_LEN                 = 128                                # max tokenized note length for BERT
BERT_PRETRAINED_PATH    = "multitask_bert.pt"                # fine-tuned weights (.pt) for multi-task heads
RF_MODEL_PATH           = "rf_pipeline.pmml"                 # Random forest ML model
TOKEN_DIR               = "saved_tokenizer"                  # folder with tokenizer
LABELS_PATH             = "saved_tokenizer/label_mappings.json" # saved label mappings
BASE_MODEL_NAME         = "emilyalsentzer/Bio_ClinicalBERT"    # HuggingFace model used during training
```

saved_tokenizer: A saved_tokenizer is a serialized file or folder containing a Hugging Face tokenizer, allowing you to reload and reuse the same text preprocessing logic for consistent input handling in later tasks or deployments.

label_mappings: label_mappings refer to a dictionary or list that maps model output indices (like class numbers) to their actual semantic labels (such as 'spam', 'not spam'), ensuring easy-to-understand results from predictions.

3. Load Pre-trained BERT and tokenizer.

```
# ========================================
# 5. Load trained BERT & tokenizer
# ========================================
# Initialize the model with correct head sizes and load the fine-tuned weights.
# model.eval() disables dropout etc. for deterministic inference.
# The tokenizer must match BASE_MODEL_NAME used to train the model.

USE_NNPA = True

# Try to use NNPA, fallback to CPU if unavailable
try:
    import torch_nnpa
    device = torch_nnpa.device()  # This gives you the NNPA device object
    print("NNPA device found. Using NNPA for inference.")
except ImportError:
    print("torch_nnpa not installed, using CPU.")
    device = torch.device("cpu")

NUM_LABELS = {task: len(label_map[task]) for task in TASKS}

model = BertMultiTaskSimple(BASE_MODEL_NAME, NUM_LABELS)
model.load_state_dict(torch.load(BERT_PRETRAINED_PATH, map_location="cpu"))
model.eval
# Only after loading, move model to NNPA device
model.to(device)

tokenizer = BertTokenizer.from_pretrained(BASE_MODEL_NAME)
```

As you see in the code, USE_NNPA is set to True. This indicates that we are using **NNPA** to leverage the on-chip AI accelerator on z16 and z17.
NNPA (Neural Network Processing Assist) is a new architected instruction with the IBM z16 and z17 which is used to execute work on the IBM Integrated Accelerator for AI. Framework and compiler developers would typically use IBM zDNN library instead of attempting to directly code the NNPA instruction. Note that this is not an aspect that end users need to worry about as frameworks and compilers like the IBM Deep Learning Compiler (ONNX-MLIR), Snap ML, TensorFlow will handle this for you (at the right software levels!).

4. Predict the final underwriting insights.

For a sample transaction, you will extract few features from unstructured data leveraging the BERT-based model:

Given Input Data


```
{
    "patient_id": "P000001",
    "visit_id": "P000001_V1",
    "visit_date": "25/12/24",
    "age": 73,
    "sex": "F",
    "bmi": 20.6,
    "race": "Asian",
    "detailed_clinical_notes": "Pt is a 73yo Asian female. Hx shows musculoskeletal, on monotherapy. Smokes? Never. Drin
ks: Heavy. Seen today for f/u, mentions ongoing issues, chart reviewed. Noted past RA. meds list updated.",
    "glucose": 76.26,
    "hba1c": 4.53,
    "total_cholesterol": 157.92,
    "ldl_cholesterol": 93.25,
    "hdl_cholesterol": 52.04,
    "triglycerides": 111.75,
    "systolic_bp": 125.97,
```


BERT Extracted Features

- **Smoking Status:** Never
- **Alcohol Use:** Heavy
- **Current Medications Category:** monotherapy
- **Disease Category:** musculoskeletal

5. These features will then be combined with further medical lab values and vitals to give the final underwriting prediction.

Combined Input to RF Model


```
{
    "glucose": 76.26,
    "hba1c": 4.53,
    "total_cholesterol": 157.92,
    "ldl_cholesterol": 93.25,
    "hdl_cholesterol": 52.04,
    "triglycerides": 111.75,
    "systolic_bp": 125.97,
    "diastolic_bp": 92.62,
    "creatinine": 1.05,
    "bun": 14.35,
    "alt": 28.29,
    "ast": 27.02,
    "tsh": 1.76,
    "free_t4": 1.34,
    "wbc": 4.7,
    "rbc": 4.32,
```


Final Underwriting Decision

Probabilities: {'probability(0)': 0.3086120857360034, 'probability(1)': 0.6913879142639965}

Decision: 1

Status: DECLINED

Looking at the probabilities, transaction status is flagged as **DECLINED**, i.e., underwriters will Decline the policy approval for the claimant.

Tip: You can change 'iloc[ROW_NUMBER]' to any number between 0 and 14 to try different rows and see how structured + text inputs affect the decision.

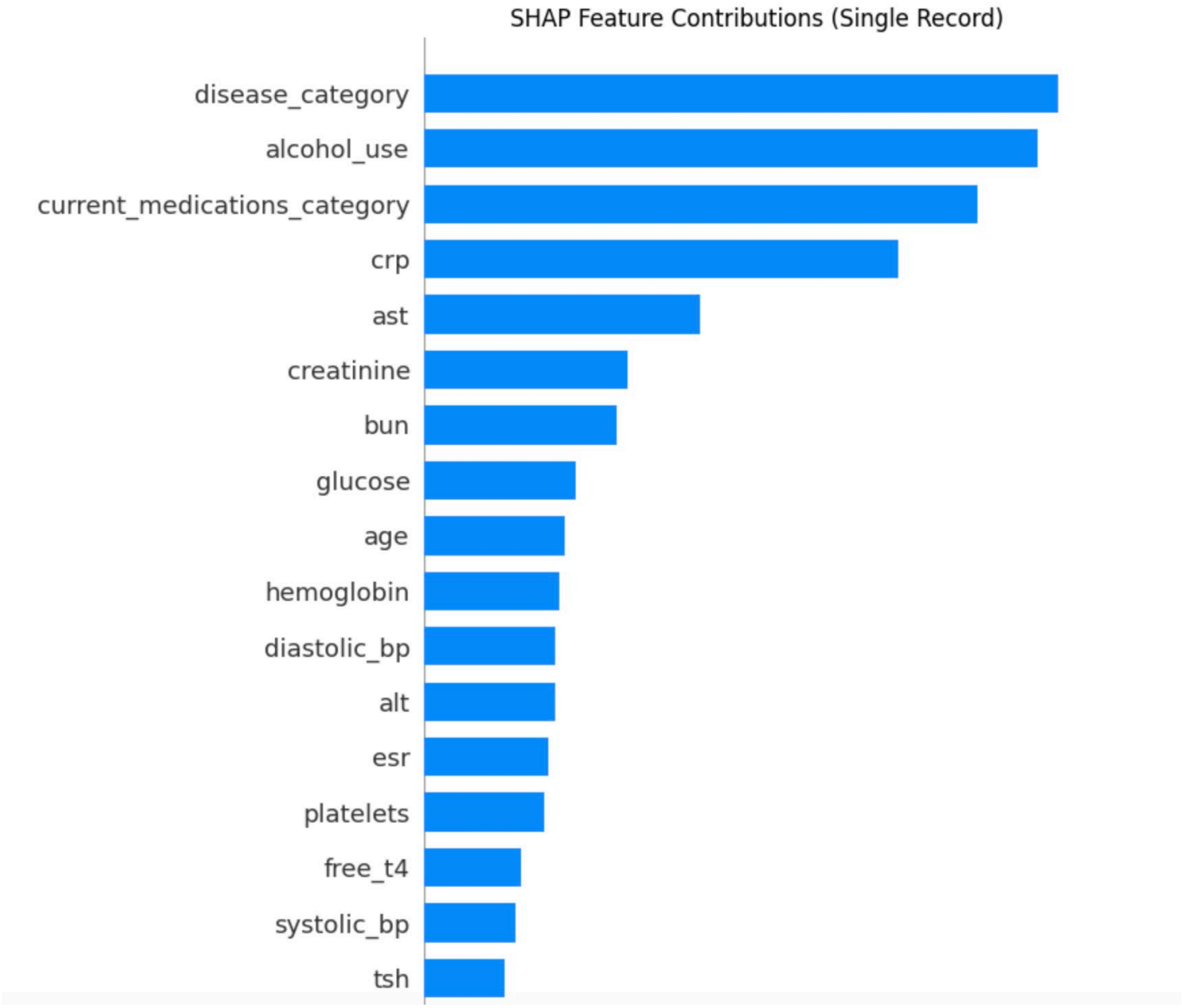
```
# ========================================
# 9. Run Example
# ========================================
# Demo: run the end-to-end pipeline on one row from the dataset.
# Tip for students:
#   You can change 'iloc[ROW_NUMBER]' to any number between 0 and 14
#   to try different rows and see how structured + text inputs affect the decision.

# Load data
data = pd.read_csv("Insurance_Dataset_for_Inferencing.csv")

# Students can change this index to try different rows
row_number = 1  # default example
sample_row = data.iloc[row_number]
```

6. Now, if we need to add explanations to our predictions, why underwriting decision was DECLINED, we can review the **SHAP Explanations**.

SHAP (SHapley Additive exPlanations), is a powerful library for model explainability. SHAP values help to explain the output of any machine learning model by assigning an importance value to each feature for a particular prediction. This helps in understanding which features contribute positively or negatively to the outcome.



As you can see, Key features such as 'disease_category', 'alcohol_use', and 'current_medications_category' identified from unstructured clinical notes using the BERT model—play a significant role in underwriting decisions. Omitting these from AI analysis would undermine the process. This highlights the pivotal value of a multiple model approaches that lead to more advanced, accurate and robust insights thru the processing of both structured and unstructured data in the AI solution.

4. Summary

Let’s recap! In this lab you were able to

- Launch a pre-built Hugging Face container for AI workloads.
- Explore a Jupyter notebook utilizing an encoder LLM BERT-based model.
- Go thru another notebook featuring a Random Forest model that delivers predictive insights by combining all relevant data.
- Execute a guided inference notebook step-by-step, leveraging PyTorch from the AI toolkit for IBM Z.