AI/ML Coding Round - Data Preparation

Problem:

Train a LLM that can answer queries about JFrog Pipelines' native steps. When posed with a question like "How do I upload an artifact?" or "What step should I use for an Xray scan?", the model should list the appropriate native step(s) and provide an associated YAML for that step.

Requirements

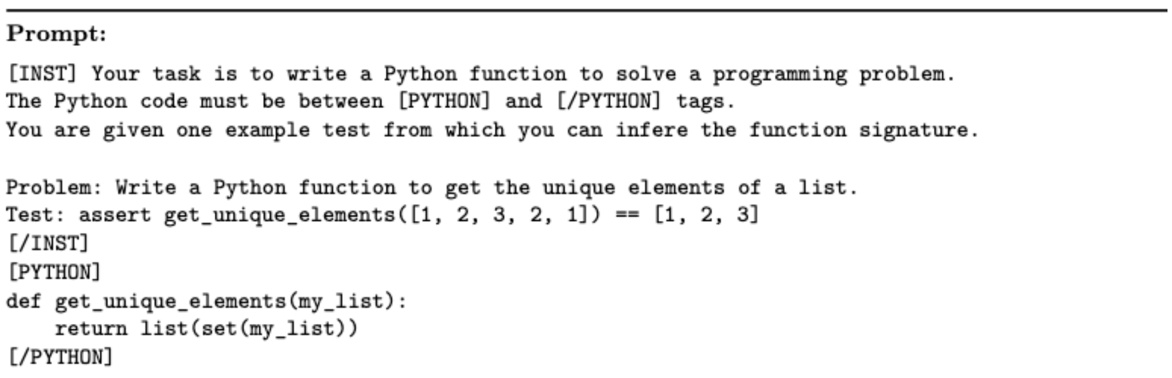
1. Data Collection: Acquire publicly available information on Native Steps from JFrog's website that contain information on native steps for building pipelines. Data that is not publicly accessible falls outside the scope of this coding challenge. (https://jfrog.com/help/r/jfrog-pipelines-documentation/pipelines-steps)
2. Data Preprocessing: Process the text to make it suitable for training. This might involve tokenization, stemming, and other NLP techniques.
3. Model Training: Train a LLM on the (preprocessed) dataset. You can choose one of the freely available open source model like BERT or any other model available
4. Query Handling: Implement a function that takes a user query as input and returns the appropriate native step(s) and a sample YAML configuration.
5. YAML Generation: Implement a function that can generate a sample YAML configuration based on the identified native step(s).

```
@author - Midhun Kumar
@email  - midhunkumar04@agmail.com
```

---

1. Data preparation for training

- we have got the raw data from the site for variopus piplines
- we have data that can talk about the pipoline procedure and sample yaml files
- I have choosen the meta's opensourse `codellama-instruct-7B` model for finetuning since its current best model for code completion and instruction
- Since we are using the above model we need to preprocess for below format


Prompt:

```
[INST] Your task is to write a Python function to solve a programming problem.
The Python code must be between [PYTHON] and [/PYTHON] tags.
You are given one example test from which you can infere the function signature.

Problem: Write a Python function to get the unique elements of a list.
Test: assert get_unique_elements([1, 2, 3, 2, 1]) == [1, 2, 3]
[/INST]
[PYTHON]
def get_unique_elements(my_list):
    return list(set(my_list))
[/PYTHON]
```

- As you can see we have know ewhat part is YAML which we taged inside this blocks `[code][/code]` we need to converts them into `[YAML]` tag in order extract from trained model

Step1: Load the data

In [ ]:
```python
# Importing Libraries
import re
import pandas as pd
pd.set_option('display.max_colwidth',None)
```

In [ ]:
```python
raw_data = pd.read_csv('./jFrog_pipline.csv', index_col=False, encoding='
raw_data.head(1)
```

Out[ ]:

Unnamed: 0	Title	PiplineProcess
		JFrog Pipelines offers JFrog Platform customers three vital capabilities: end-to-end automation (CI/CD), workflow and tool orchestration, and the optimization of the JFrog toolset functionality in use. Consistent with JFrog's customer-centric product philosophy, Pipelines is enterprise-ready and universal.\n### Workflow Automation\nA pipeline is an event-driven automated workflow for executing a set of DevOps activities (CI, deployments, infrastructure provisioning, etc). It is composed of a sequence of interdependent **steps** which execute discrete functions. Steps act on **resources** , which hold the information needed to execute (files, key-value pairs, etc).\nDevelopers can create pipelines easily with a simple declarative YAML-based language. While each step in a pipeline executes in a stateless runtime environment,

0	0	jfrog-pipelines	Pipelines provides facilities to manage state and step outputs across the workflow so that all dependent steps can access the information they need from upstream steps in order to execute. This helps coordinate activities centrally across diverse DevOps tools and teams without custom DIY scripts.\nWorkflows can be configured for a variety of scenarios, including:\n * Continuous Integration for your applications\n * Continuous Delivery workflows that connect all your CI/CD and DevOps activities across tools and functional silos\n * Automate IT Ops workflows like infrastructure provisioning, security patching, and image building\n\n\n### Get up and running with JFrog Pipelines\nIn this section, you will find information to get you started whether you are a new user or an existing user.\n * If you do not yet have a subscription, get started with trial subscription of the JFrog Platform on the Cloud.\n * If you are a new user, get started with the onboarding videos for JFrog Pipelines.Onboarding Best Practices: JFrog Pipelines\n\n\n### Features\n#### Pipelines as Code\nDefine your automated workflow through code, using a domain specific language in a YAML file of key-value pairs that you can create and maintain with your favorite text editor.\n#### Real Time Visibility\nJFrog Pipelines renders your pipeline definition as an interactive diagram, helping you to see the flow of tasks and their inter-dependencies, as well as view the success record of any runs that were performed.\n#### Universal\nConnect your pipeline automation to your source code repositories in a version control system (such as GitHub or BitBucket) to automatically trigger execution on any new submission (commit) of a code change. Connect to other popular tools through your credentials for storage, issue-tracking, notification, orchestration and more through a library of integrations.\n#### Native Integration with Artifactory\nJFrog Pipelines is designed to be used with Artifactory, with built-in directives for pushing artifacts, performing builds, pushing build information, image scanning, and build promotion.\n#### Integration with JFrog Platform\nJFrog Pipelines is designed as an integral part of the JFrog platform, including scanning artifacts/builds through Xray, the creation and delivery of release bundles through JFrog Distribution, for a complete end-to-end SDLC pipeline from commit to production runtime.\n#### Security First\nFine-grained permissions and access control limit who can access workflows. Centralized, encrypted storage of credentials and keys help ensure secrets stay safe.\n#### Enterprise-Ready\nManage multiple execution nodes using a single installation of Pipelines and automatically distribute Pipeline execution across them for scale and speed.\n### Watch the Screencast\n\n Document url for reference - https://jfrog.com/help/r/jfrog-pipelines-documentation/pipelines-steps

Step3: Save the data as csv

Points to remember:

1. we need to add `[INST]` tag for instruction in each data
2. `<<SYS>>` for system message and `[YAML]` for yaml block - here system message is somthing like asking llm to do specific task
3. we can consider till First `### - 3 Hashs` as Instruction block
4. And will replce `[code]` to `[YAML]`

example:

`[INST] <<SYS>> You are a helpful, respectful and honest assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature. If a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share false information. <</SYS>> {prompt}[/INST]`

Step2: Preparing Data

In [ ]:
```
# data preparation
system_message = f'[INST]<<SYS>> You are a helpful, respectful and honest
```

In [ ]:
```
# Creating function for tag addtion
def tagAddtion(string:str) -> str:
    '''
    This function takes string as input and returns tag added as output
    '''
    string = re.sub('#','[/INST]#',string,1)
    string = re.sub(f'\[code\]','[YAML]', string)
    string = re.sub(f'\[/code\]','[/YAML]', string)
    string = system_message + string
    return string
```

In [ ]:
```
final_data = pd.DataFrame()
final_data['PiplineProcess'] = raw_data.PiplineProcess.apply(tagAddtion)
```

In [ ]:
```
final_data.loc[76]
```

Out[ ]:
```
PiplineProcess    [INST]<<SYS>> You are a helpful, respectful and honest
assistant. Always answers only users jFrog pipline related questiions.Yo
u will say i am not sure for other general questions <</SYS>>The **Distr
ibuteReleaseBundle** native step triggers the distribution of Release Bu
ndles to an Artifactory Edge Node. This step requires a signed release b
```

```
undle and one or more distribution rules to successfully execute.Distrib
uting Release BundlesJFrog Artifactory Edge\n[/INST]##### YAML Schema\nT
he YAML schema for DistributeReleaseBundle native step is as follows:\n 
**DistributeReleaseBundle**\n[YAML]\n    pipelines:\n      - name:    <s
tring>\n        steps:\n          - name: my_distribute\n            typ
e: DistributeReleaseBundle\n            configuration:\n              
#inherits all the tags from bash; \n              dryRun: <boolean>     
# optional\n              inputResources:\n                - name: my_re
leaseBundle      # one ReleaseBundle is required\n                  trig
ger: false   \n                - name: my_distributionRule   # one Distr
ibutionRule is required\n                  trigger: false    # default t
rue\n              outputResources:\n                - name: my_releaseB
undleOutput # one ReleaseBundle is optional\n     \n            executio
n:\n              onStart:\n                - echo "Preparing for work..
."\n              onSuccess:\n                - echo "Job well done!"\n 
onFailure:\n                - echo "uh oh, something went wrong"\n      
onComplete: #always\n                - echo "Cleaning up some stuff"\n  
\n[/YAML]\n##### Tags\n###### name\nAn alphanumeric string (underscores 
are permitted) that identifies the step.\n###### type\nMust be `Distribu
teReleaseBundle `for this step type.\n###### configuration\nSpecifies al
l configuration selections for the step's execution environment. This st
ep inherits the Bash/ PowerShell step configuration tags, including thes
e pertinent tags:\nTag\n **Description of usage**\nRequired/Optional  \n
`inputResources`\nMust specify a ReleaseBundle resource and one Distribu
tionRule resource.\nRequired  \n`outputResources`\nMay specify a Release
Bundle resource to be updated with the `name` and `version` of the input
ReleaseBundle.\nOptional  \nIn addition, these tags can be defined to su
pport the step's native operation:\n### Tags derived from Bash\nAll nati
ve steps derive from the Bash step. This means that all steps share the 
same base set of tags from Bash, while native steps have their own addit
ional tags as well that support the step's particular function. So it's 
important to be familiar with the Bash step definition, since it's the c
ore of the definition of all other steps.\nTag\n **Description of usage*
*\nRequired/Optional  \n`dryRun`\nControls whether this should be a dry 
run to test if the release bundle can distribute to the Edge nodes match
ing the distribution rule.\nThe default is true.\nOptional  \n###### exe
cution\nDeclares collections of shell command sequences to perform for p
re- and post-execution phases:\nTag\n **Description of usage**\nRequired
/Optional  \n`onStart`\nCommands to execute in advance of the native ope
ration\nOptional  \n`onSuccess`\nCommands to execute on successful compl
etion\nOptional  \n`onFailure`\nCommands to execute on failed completion
\nOptional  \n`onComplete`\nCommands to execute on any completion\nOptio
nal  \nThe actions performed for the `onExecute` phase are inherent to t
his step type and may not be overridden.\n##### Examples\nThe following 
examples show how to configure a DistributeReleaseBundle step to distrib
ute or for a dry run.\n###### Distribute Input Release Bundle Edge Node\
nDistributes the input release bundle to the edge nodes defined in the d
istribution rule.\n  * This example requires an Artifactory Integration 
and a Distribution Integration.\n  * The Pipelines DSL for this example 
is available in this repository in the JFrog GitHub account.\n\n\n **Dis
tributeReleaseBundle**\n[YAML]\n    template: true   # required for loca
l templates\n    valuesFilePath: ./values.yml\n    \n    resources:\n   
# Build info of first build to bundle\n      - name: gosvc_promoted_buil
```

```
d_info\n        type: BuildInfo\n        configuration:\n          sourc
eArtifactory: {{ .Values.myArtifactoryIntegration }}\n          buildNam
e: svc_build\n          buildNumber: 1\n    \n      # Build info of seco
nd build to bundle\n      - name: appl_promoted_build_info\n        type
: BuildInfo\n        configuration:\n          sourceArtifactory: {{ .Va
lues.demoArtifactoryIntegration }}\n          buildName: backend_build\n
buildNumber: 1\n    \n      # Release bundle\n      - name: release_bund
le\n        type: ReleaseBundle\n        configuration:\n          sourc
eDistribution: {{ .Values.distributionIntegration }}\n          name: de
mo_rb\n          version: v1.0.0\n    \n      # Signed version of the sa
me release bundle\n      - name: signed_bundle\n        type: ReleaseBun
dle\n        configuration:\n          sourceDistribution: {{ .Values.di
stributionIntegration }}\n          name: demo_rb\n          version: v1
.0.0\n    \n      # Distribution rules\n      - name: distribution_rules
\n        type: DistributionRule\n        configuration:\n          sour
ceDistribution: {{ .Values.distributionIntegration }}\n          service
Name: "*"\n          siteName: "*"\n          cityName: "*"\n          c
ountryCodes:\n            - "CN"\n            - "GB"\n    \n    pipeline
s:\n      - name: demo_release_mgmt\n        steps:\n          - name: b
undle\n            type: CreateReleaseBundle\n            configuration:
\n              releaseBundleName: demo_rb\n              releaseBundleV
ersion: v1.0.${run_number}\n              dryRun: false\n              s
ign: false\n              description: "some random test description"\n
inputResources:\n                - name: gosvc_promoted_build_info\n
trigger: true\n                - name: appl_promoted_build_info\n
trigger: true\n              outputResources:\n                - name: r
elease_bundle\n              releaseNotes:\n                syntax: mark
down\n                content: |\n                  ## Heading\n
* Bullet\n                    * Points\n                \n          - na
me: sign\n            type: SignReleaseBundle\n            configuration
:\n              inputResources:\n                - name: release_bundle
\n              outputResources:\n                - name: signed_bundle\
n    \n          - name: distribute\n            type: DistributeRelease
Bundle\n            configuration:\n              dryRun: false\n
inputResources:\n                - name: signed_bundle\n
- name: distribution_rules\n[/YAML]\n###### Trigger a Dry Run\nTriggers
a dry run of the distribution.\n **DistributeReleaseBundle**\n[YAML]\n
pipelines: \n      - name: distributeReleaseBundlePipeline\n        step
s:\n          - name: distributeReleaseBundleDryRun\n            type: D
istributeReleaseBundle\n            configuration:\n              dryRun
: true\n              inputResources:\n                - name: myRelease
Bundle\n                - name: myInputDistributionRule\n    \n[/YAML]\n
###### Triggers Distribution and Updates Output Resource\nTriggers a dis
tribution and updates the output resource with the name and version of t
he input.\n **DistributeReleaseBundle**\n[YAML]\n    pipelines: \n
- name: distributeReleaseBundlePipeline\n        steps:\n          - nam
e: distributeReleaseBundleDryRun\n            type: DistributeReleaseBun
dle\n            configuration:\n              dryRun: false\n
inputResources:\n                - name: myReleaseBundle\n
- name: myInputDistributionRule\n              outputResources:\n
- name: myOutputReleaseBundle\n    \n[/YAML]\n###### Same Step for Dry R
uns and to Distribute\nIn this example, the same step is used for both d
ry runs and to distribute the release bundle to Edge Nodes. The dry_run
```

```
variable may be set in the  pipeline configuration section or  step conf
iguration or added as a run variable by an earlier step in the pipeline
using add_run_variable.\n **DistributeReleaseBundle**\n[YAML]\n    pipel
ines: \n      - name: distributeReleaseBundlePipeline\n        steps:\n
- name: distributeReleaseBundleStep\n            type: DistributeRelease
Bundle\n            configuration:\n              dryRun: ${dry_run}\n
inputResources:\n                - name: myReleaseBundle\n
- name: myInputDistributionRule\n    \n[/YAML]\n##### How it Works\nWhen
you use the **DistributeReleaseBundle** native step in a pipeline, it pe
rforms the following functions in the background:\n  * Create the distri
bution payload (the JSON object that will be in the request to Distribut
ion)\n  * curl $distributionUrl/api/v1/distribution/$releaseBundleName/$
releaseBundleVersion (send the distribution or dry run payload to Distri
bution)\n  * curl $distributionUrl/api/v1/release_bundle/$releaseBundleN
ame/$releaseBundleVersion/distribution/$trackerId (if not a dry run, usi
ng the tracker ID returned by Distribution, check if the distribution is
complete)\n  * write_output (update the output ReleaseBundle resource)\n
\n\n\n Document url for reference - https://jfrog.com/help/r/jfrog-pipel
ines-documentation/pipelines-steps
Name: 76, dtype: object
```

In [ ]:
```
# Verify if all 244 docuemnts has the proper tags
len(final_data.loc[final_data.PiplineProcess.str.contains('[/INST]')]), l
```

Out[ ]:
```
(244, 244)
```

- We can see we have tags for all the 244 documents
- We can also see the YAML tags also in place

Step3 : We have our tarining data lets save this

In [ ]:
```
final_data.to_csv('./final_data_for_training.csv')
```

In [ ]:
```
```