1.2 AIB2.0 User Journey Workflows

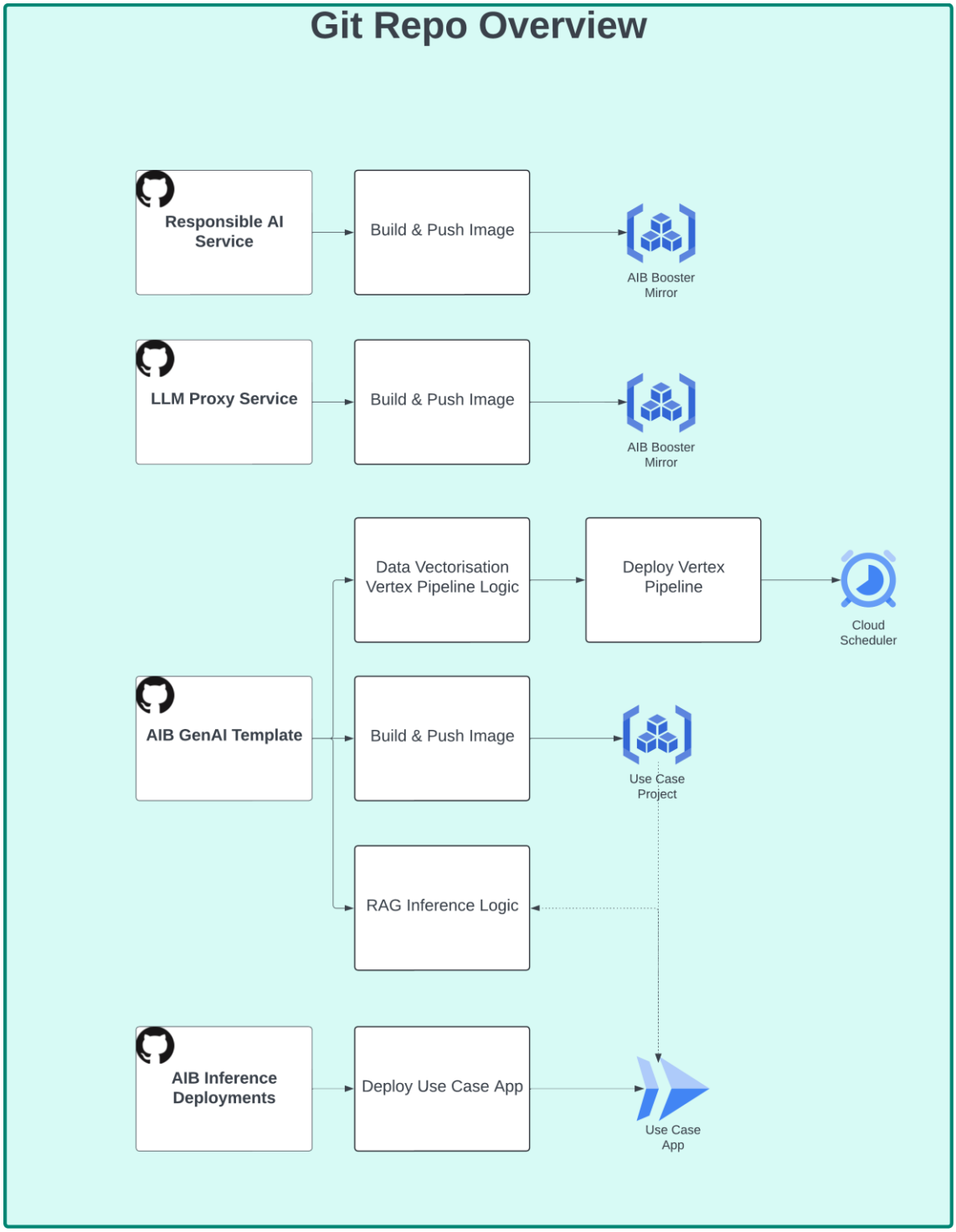
1. Background & Context

AIBooster 2.0 aims to build a generic GenOps Platform on top of the existing AIB1.0 MLOps Platform while providing functional systems & templates to rapidly productionize a Generative AI business use case while maintaining best practices and encapsulating all possible aspects that an enterprise-grade platform should capture (scalability, robustness, logging, monitoring etc)

A developer leveraging this platform can adhere to multiple personas like an AIB ML Engineer, Use Case ML Engineer, AIB Platform Engineer etc.

This playbook aims to guide developers new to the platform through key user journeys to get started with the same in a simplified & concise manner.

2. Key Git Repositories



AIB2.0 as a Platform can be utilised primarily through the use of the following Git repositories:

#	Git Repo	Description	Expected end Outcome
1	Responsible AI Service	This repository covers the logic running for the Responsible AI Service. The purpose of this service is to scan LLM inputs & outputs to meet Security & Privacy	This repository outputs an image stored in a centrally located Google Cloud Artifact Registry via CI/CD &

#	Git Repo	Description	Expected end Outcome
		requirements while ensuring minimal impact to the process of productionizing a GenAI use case This service is deployed as a Cloud Run instance in an AI Gateway Orchestration layer For more details, please refer to 1.3.2 Responsible AI Service	the same can be used as a base to deploy the service to Cloud Run
2	LLM Proxy Service	This repository covers the logic running for the LLM Proxy Service. Any LLM interactions that a GenAI use case needs to run passes through this service providing a framework to control which LLMs can a use case access & what kind of Responsible AI configuration is required for the same For more details, please refer to 1.4.1 LLM Proxy Service	This repository outputs an image stored in a centrally located Google Cloud Artifact Registry via CI/CD & the same can be used as a base to deploy the service to Cloud Run
3	AIB GenAI Template	This repository provides a reference implementation for productionizing a GenAI use case on Google Cloud, & can be used as a starting point w/ configurable boilerplate code. The implementation includes: • Data ingestion and vectorisation pipeline using Kubeflow Pipelines and Vertex AI Pipelines. • LLM inference using Vertex AI with RAG setup by Vertex AI Vector Search. • Infrastructure-as-Code using Terraform for Cloud Run services, PubSub and Cloud Scheduler.	This repository once cloned for a specific use case, can output: • A Base image to be used by the overall use case logic (RAG Inference + Data Vectorisation + LLM Evaluation) • A Cloud Scheduler to run Data Vectorisation regularly • A Cloud Scheduler to run Evaluation pipelines regularly
4	AIB Inference Deployments	This repository serves as a centrally-managed place to deploy any GenAI Use Case application as a Cloud Run Service. Using YAML-based configurations hooked w/ automated CI/CD under-the-hood, a developer can quickly plug in configuration parameters to deploy an application across environments (lab/non-live/live)	This repository runs terraform-based deployment of all GenAI Use Case applications to Cloud Run via automated CI/CD under the hood

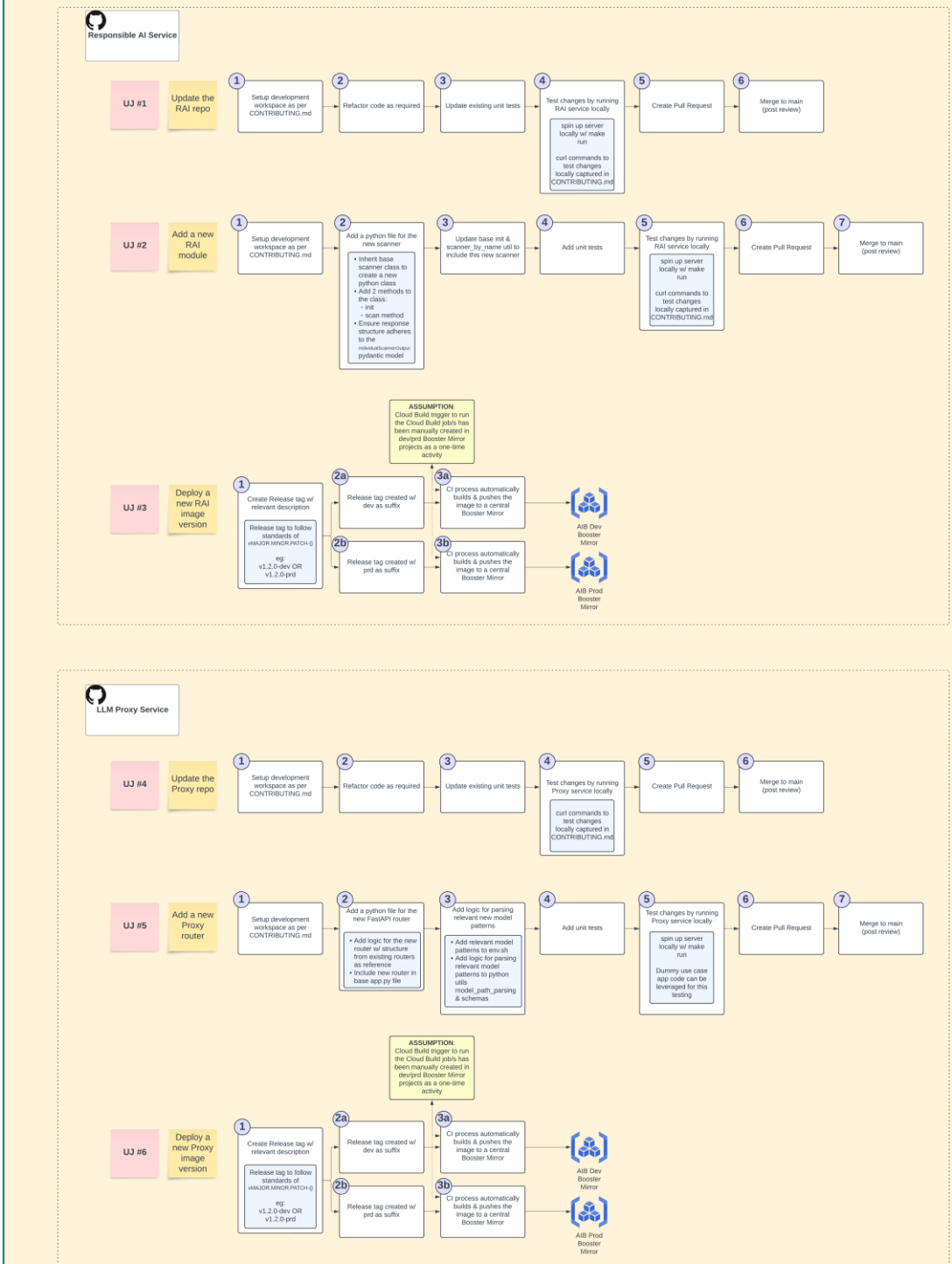
3. Key User Journeys

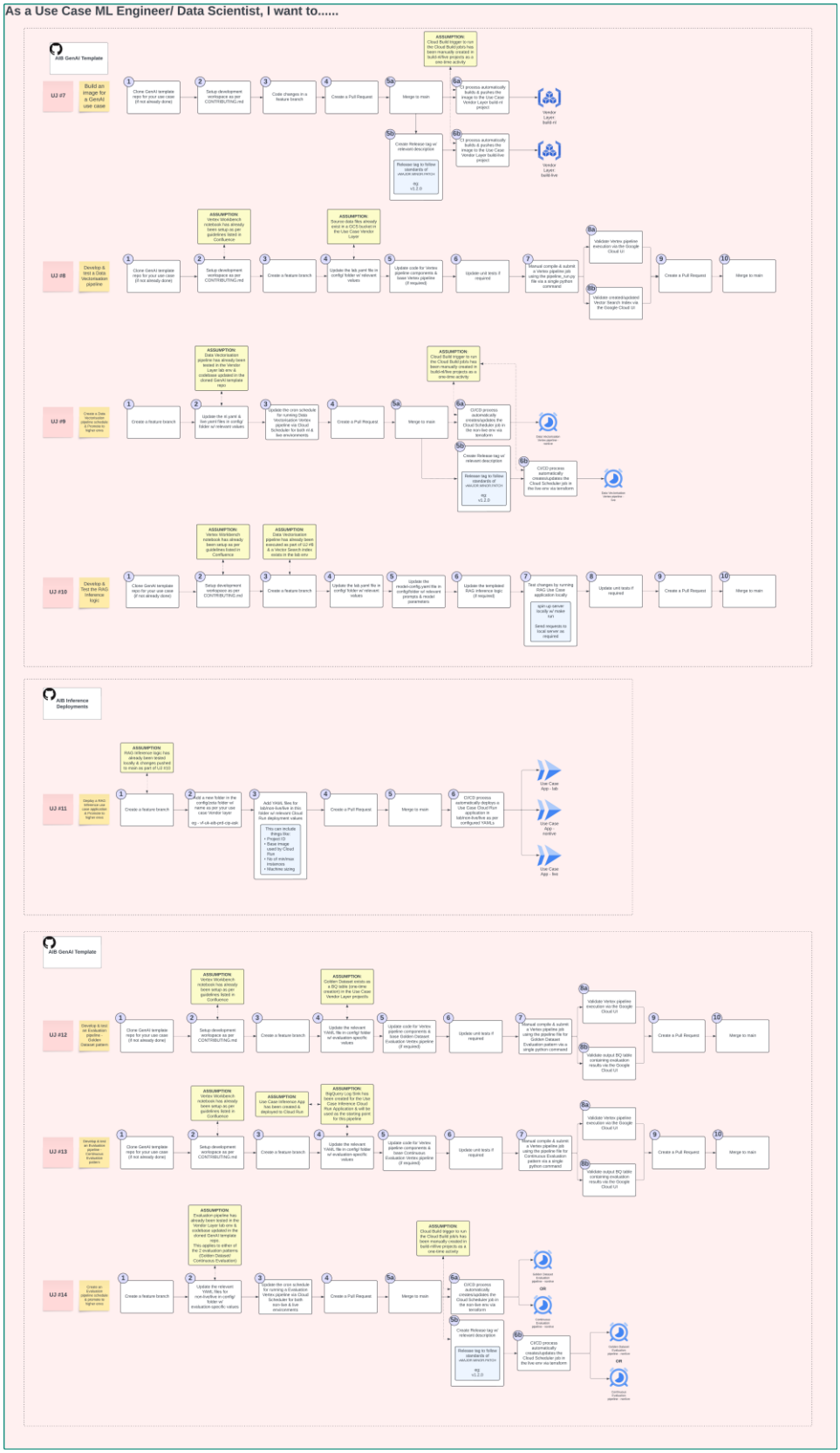
Key User Journeys summarized below capture how different personas can leverage the 4 Git Repos mentioned above to successfully complete a relevant user journey

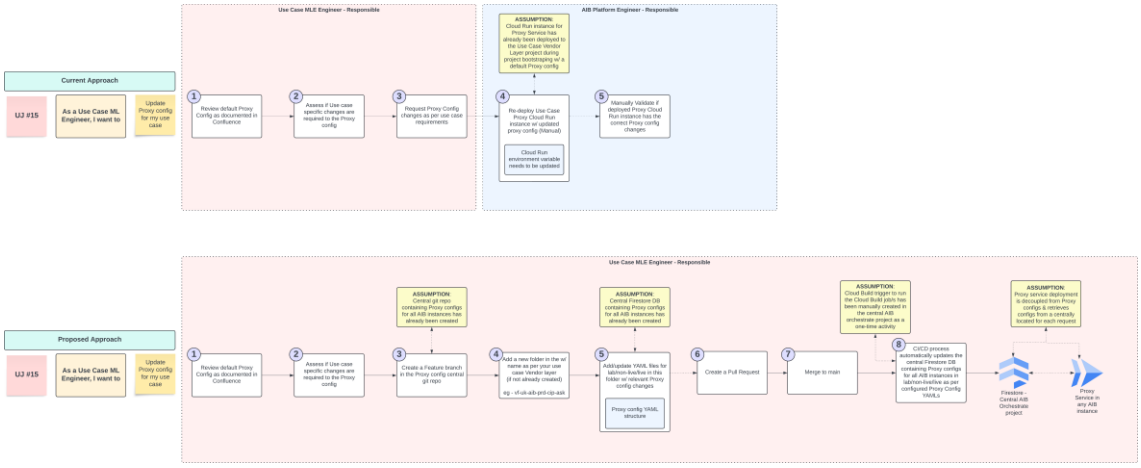
#		Persona		User Journey	Applicable Git Repo
1	As a/an	AIB ML Engineer	,I want to	Update the Responsible AI git repo	Responsible AI Service
2				Add a new Responsible AI module	
3				Deploy a new Responsible AI service image version	
4				Update the Proxy service git repo	LLM Proxy Service
5				Add a new Proxy router	
6				Deploy a new Proxy service image version	
7		Use Case ML Engineer/ Data Scientist		Build an image for a GenAI use case	AIB GenAI Template
8				Develop & test a Data Vectorisation pipeline	
9				Create a Data Vectorisation pipeline schedule & Promote to higher environments	
10				Develop & test the RAG inference logic	
11				Deploy a RAG inference use case application & Promote to higher environments	AIB Inference Deployments
12				Develop & Test an Evaluation pipeline - Golden Dataset pattern	AIB GenAI Template
13				Develop & Test an Evaluation pipeline - Continuous evaluation pattern	
14				Create an Evaluation pipeline schedule & Promote to higher environments	
15				Update Proxy config for my use case	

Workflows for these 15 User Journeys can be summarized in the designs below w/ further details on each User Journey captured in subsequent sub-pages:

As an AIB ML Engineer, I want to.....




As a Use Case ML Engineer/ Data Scientist, I want to.....
AIB GenAI Template
UJ #7
Build an image for a GenAI use case
UJ #8
Develop & test a Data Vectorisation pipeline
UJ #9
UJ #10
Develop & Test the RAG inference logic
AIB Inference Deployments
UJ #11
AIB GenAI Template
UJ #12
UJ #13
UJ #14



1.2.1 User Journeys - Persona: AIB ML Engineer

1. Persona Description

As an AIB ML Engineer, responsibilities can include maintenance, refactoring & enhancements to key AIB2.0 Platform components.

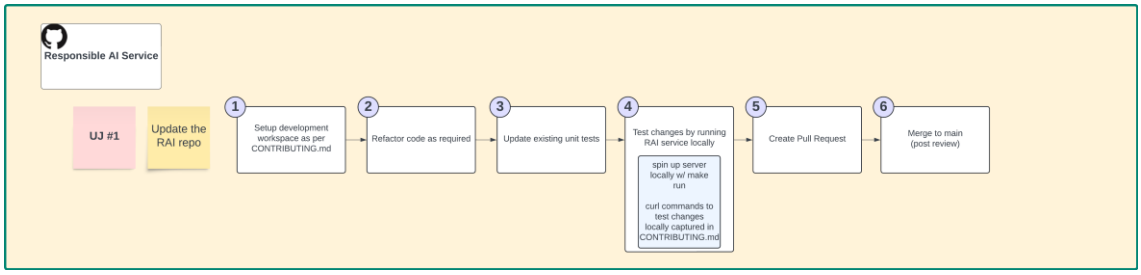
2 key components in this platform include the:

- Responsible AI service
- LLM Proxy service

6 User Journeys outlined below capture details on how to address important elements of this maintenance, refactoring & enhancements:

2. Persona Specific User Journeys

2.1. User Journey #1 - Update the Responsible AI git repo

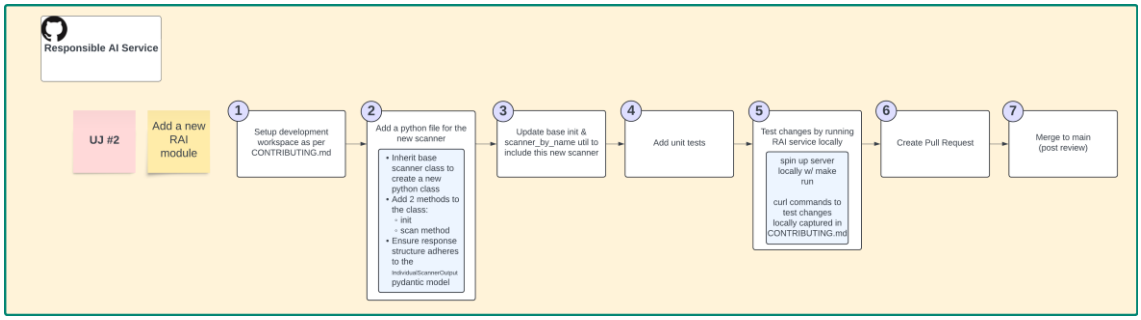


Based on the user journey outlined above, this UJ applies to the Responsible AI Service git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Terraform installation Detailed steps for the same are captured in the CONTRIBUTING.md of the repo This markdown file includes further details on testing the service locally & miscellaneous related information on unit testing etc	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster
2	Refactor code as required	Once the development workspace has been completely setup, a developer can create a feature branch, refactor code as required & commit changes to said feature branch	
3	Update existing unit tests	The repo contains unit tests related to all existing functionality in the `tests/` folder These tests can be updated based on the refactoring of functionality as required & tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
4	Test changes locally	Once required changes have been committed, the same can be tested locally by spinning up the Responsible AI Service locally using `make run` which spins up the FastAPI server on localhost:8000 by default	

#	Element	Description	Assumptions
		Furthermore example `curl` commands to send an API request to this local server have been captured in the CONTRIBUTING.md file	
5	Create a Pull request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
6	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.2. User Journey #2 - Add a new Responsible AI module



Based on the user journey outlined above, this UJ applies to the Responsible AI Service git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Terraform installation Detailed steps for the same are captured in the CONTRIBUTING.md of the repo This markdown file includes further details on testing the service locally & miscellaneous related information on unit testing etc	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster • VSCode has been connected to the created Workbench instance

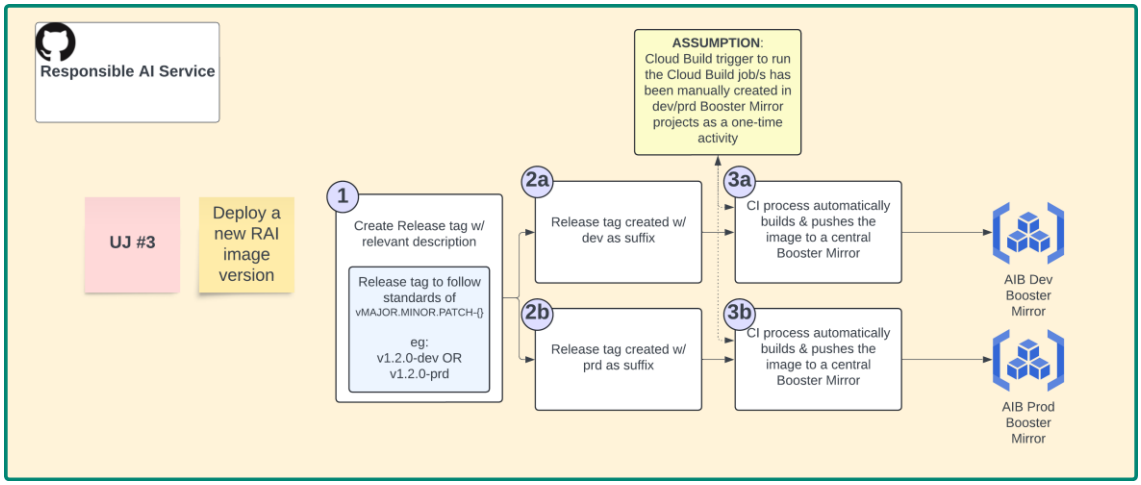
#	Element	Description	Assumptions
			using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster

2	Add a python file for the new scanner	All Responsible AI modules exist in the `/responsible_ai_modules/scanners/` folder.
		This includes existing modules for:
		• Content Moderation using the Google Cloud Natural Language API
		• Anonymization using the Google Cloud DLP API
		• Prompt Injection using an open-source model deployed to a Vertex endpoint in the AI Gateway layer
		New modules can include future roadmap functionality like topicality, embedding scans etc.
		
		Any new module needs to exist in the above-mentioned folder w/ a base python class structure containing:
		• The new scanner class deriving from the Base Scanner class
		• 2 methods for:
		  ○ base initialisation of the class
		  ○ Scan method capturing underlying logic for the new module
		Furthermore, all scanners/modules are generalized to adhere to a pydantic model called `IndividualScannerOutput` that exists in `responsible_ai_modules/schemas.py` . Any new module needs to transform their output to adhere to this structure	

3	Update supplementary files to include this new scanner	In addition to a new scanner file, following supplementary files need to be updated to include this new module: • `responsible_ai_modules/scanners/__init__.py` • `get_scanner_by_name` function in `responsible_ai_modules/scanners/utils.py`	
4	Add unit tests	The repo contains unit tests related to all existing functionality in the `tests/` folder	

#	Element	Description	Assumptions
		This folder will need to be updated w/ new unit tests to reflect new functionality created for the new scanner. This can be tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
5	Test changes locally	Once required changes have been committed, the same can be tested locally by spinning up the Responsible AI Service locally using `make run` which spins up the FastAPI server on localhost:8000 by default Furthermore example `curl` commands to send an API request to this local server have been captured in the CONTRIBUTING.md file	
6	Create a Pull request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
7	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.3. User Journey #3 - Deploy a new Responsible AI service image version

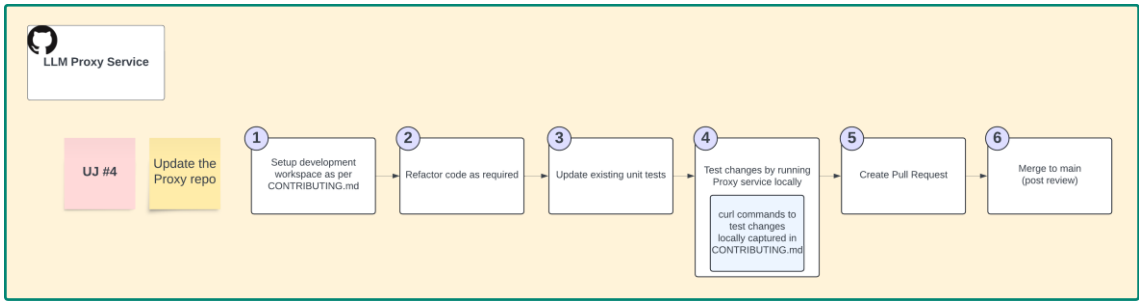


Based on the user journey outlined above, this UJ applies to the Responsible AI Service git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Create Release tag	A new release tag needs to be created w/ relevant description supporting it. This tag should conform to best practices around tags, specifically of format: vMAJOR.MINOR.PATCH-dev/prd eg - A new release tag can be:	

#	Element	Description	Assumptions
		• v1.2.0-dev OR • v1.2.0-prd	
2a	Release tag created w/ dev as suffix	If a release tag is created w/ dev as the suffix, it points to image creation in Google Cloud Artifact Registry of Dev Booster Mirror	
2b	Release tag created w/ prd as suffix	If a release tag is created w/ prd as the suffix, it points to image creation in Google Cloud Artifact Registry of Prod Booster Mirror	
3a	CI process automatically builds & pushes image to Dev Booster Mirror	Once a tag w/ dev as suffix is created, the CI pipeline is automatically triggered that runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of Dev Booster Mirror For more details on this CI pipeline run by Cloud Build, please refer to the `Dockerfile` & `cloudbuild.yaml` files in the repo	Cloud Build Trigger exists in the Dev Booster Mirror Google Cloud project to run this CI pipeline. This is a manual one-time activity & is already in place
3b	CI process automatically builds & pushes image to Prod Booster Mirror	Once a tag w/ prd as suffix is created, the CI pipeline is automatically triggered that runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of Prod Booster Mirror For more details on this CI pipeline run by Cloud Build, please refer to the `Dockerfile` & `cloudbuild.yaml` files in the repo	Cloud Build Trigger exists in the Prod Booster Mirror Google Cloud project to run this CI pipeline. This is a manual one-time activity & is already in place

2.4. User Journey #4 - Update the LLM Proxy service git repo

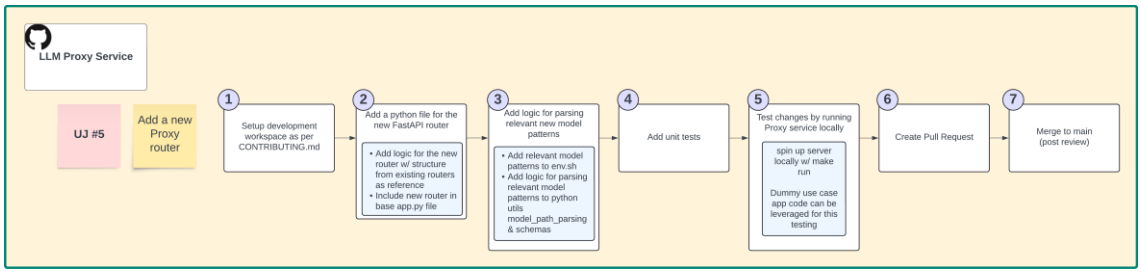


Based on the user journey outlined above, this UJ applies to the LLM Proxy Service git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Terraform installation Detailed steps for the same are captured in the CONTRIBUTING.md of the repo	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex

#	Element	Description	Assumptions
		This markdown file includes further details on testing the service locally & miscellaneous related information on unit testing etc	Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster
2	Refactor code as required	Once the development workspace has been completely setup, a developer can create a feature branch, refactor code as required & commit changes to said feature branch	
3	Update existing unit tests	The repo contains unit tests related to all existing functionality in the `tests/` folder These tests can be updated based on the refactoring of functionality as required & tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
4	Test changes locally	Once required changes have been committed, the same can be tested locally by spinning up the LLM Proxy Service locally using `make run` which spins up the FastAPI server on localhost:8000 by default Local testing can be executed in 2 ways: • `curl` directly into the local server • Spin up a dummy use case app locally using the `make run-dummy-use-case` command & update the `dummy_use_case/api/app.py` file as required The 2nd option provides more flexibility to test any LLM Proxy changes from an overall usage standpoint (calling the LLM Proxy via Langchain, Llama-Index or other LLM frameworks)	
5	Create a Pull request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
6	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.5. User Journey #5 - Add a new LLM Proxy router



Based on the user journey outlined above, this UJ applies to the LLM Proxy Service git repo & details for the same can be broken down as follows:

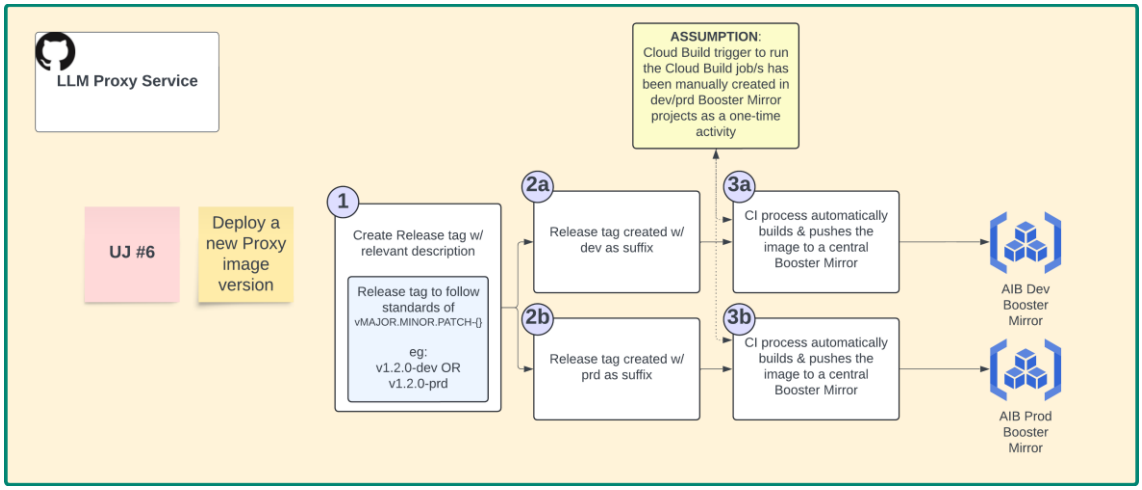
#	Element	Description	Assumptions
1	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Terraform installation Detailed steps for the same are captured in the CONTRIBUTING.md of the repo This markdown file includes further details on testing the service locally & miscellaneous related information on unit testing etc	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster

2	Add a python file for the new FastAPI	All existing LLM Proxy routers exist in the `api/routers/` folder. This currently includes logic to:
		• Access Google LLMs
		• Access external LLMs (currently tested for OpenAI LLMs hosted in an Azure Vendor project)
		  ○ This can be extended to other external LLMs like AWS Bedrock, Cohere etc (will need to test authentication mechanism)
		• Access LLMs deployed to Vertex Endpoints
		  ○ This currently has been tested for Llama & Mistral LLMs deployed to a Vertex Endpoint along w/ custom LLMS deployed	

#	Element	Description	Assumptions
		New routers can include LLMs that do not conform to the extensibility provided by above-mentioned routers. In that scenario, a new router file needs to be added to this folder w/ relevant authentication logic & API calling logic. Please refer to existing Google LLM, External LLM routers for ideas on standardized code structure Once logic for the new router has been plugged in, this router needs to be included in the base `api/app.py` file	
3	Add logic for parsing relevant new model patterns	Every LLM call via a router conforms to a generalized regex model pattern that captures all possible model paths as per the final LLM endpoint. This is currently captured in the `env.sh` file & the same is deployed with LLM Proxy Cloud Run deployment as environment variables. In addition to the above, 2 utility files capture logic to generically parse these model patterns & the same need to be amended: • api/utils/model_path_parsing.py - Need to add a new case for the new model patterns • api/utils/schema.py - Need to update the `ModelProvider` class to reflect this new model pattern	
4	Add unit tests	The repo contains unit tests related to all existing functionality in the `tests/` folder This folder will need to be updated w/ new unit tests to reflect new functionality created for the new scanner. This can be tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
5	Test changes locally	Once required changes have been committed, the same can be tested locally by spinning up the LLM Proxy Service locally using `make run` which spins up the FastAPI server on localhost:8000 by default Local testing can be executed in 2 ways: • `curl` directly into the local server • Spin up a dummy use case app locally using the `make run-dummy-use-case` command & update the `dummy_use_case/api/app.py` file as required	

#	Element	Description	Assumptions
		The 2nd option provides more flexibility to test any LLM Proxy changes from an overall usage standpoint (calling the LLM Proxy via Langchain, Llama-Index or other LLM frameworks)	
6	Create a Pull request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
7	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.6. User Journey #6 - Deploy a new LLM Proxy image version



Based on the user journey outlined above, this UJ applies to the LLM Proxy Service git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Create Release tag	A new release tag needs to be created w/ relevant description supporting it. This tag should conform to best practices around tags, specifically of format: vMAJOR.MINOR.PATCH-dev/prd eg - A new release tag can be: • v1.2.0-dev OR • v1.2.0-prd	
2a	Release tag created w/ dev as suffix	If a release tag is created w/ dev as the suffix, it points to image creation in Google Cloud Artifact Registry of Dev Booster Mirror	
2b	Release tag created w/ prd as suffix	If a release tag is created w/ prd as the suffix, it points to image creation in Google Cloud Artifact Registry of Prod Booster Mirror	

#	Element	Description	Assumptions
3a	CI process automatically builds & pushes image to Dev Booster Mirror	Once a tag w/ dev as suffix is created, the CI pipeline is automatically triggered that runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of Dev Booster Mirror For more details on this CI pipeline run by Cloud Build, please refer to the `Dockerfile` & `cloudbuild.yaml` files in the repo	Cloud Build Trigger exists in the Dev Booster Mirror Google Cloud project to run this CI pipeline. This is a manual one-time activity & is already in place
3b	CI process automatically builds & pushes image to Prod Booster Mirror	Once a tag w/ prd as suffix is created, the CI pipeline is automatically triggered that runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of Prod Booster Mirror For more details on this CI pipeline run by Cloud Build, please refer to the `Dockerfile` & `cloudbuild.yaml` files in the repo	Cloud Build Trigger exists in the Prod Booster Mirror Google Cloud project to run this CI pipeline. This is a manual one-time activity & is already in place

1.2.2 User Journeys - Persona: Use Case ML Engineer/ Data Scientist

1. Persona Description

As a Use Case ML Engineer or a Use Case Data Scientist, key responsibilities can include leveraging the AIB GenAI Template as boilerplate code to build out relevant logic for a specific GenAI use case & productionize the same w/ components provided by the AIB2.0 Platform.

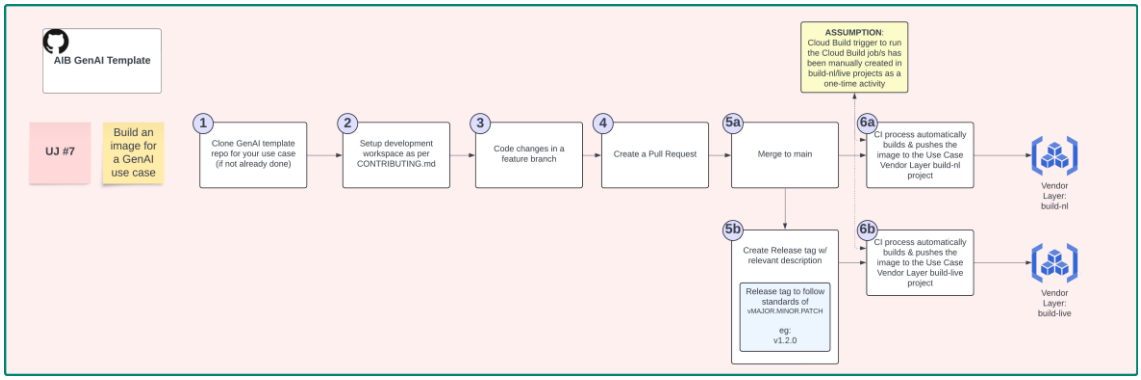
These templates capture the following essential elements of a GenAI use case:

- Data Vectorisation pipeline to ingest documents into a Vector Database to use for RAG-based retrieval during inference
- RAG inference logic to serve a GenAI application
- LLM Evaluation elements to complete the production-ready lifecycle of a GenAI use case

8 User Journeys outlined below capture details on how to address these important elements:

2. Persona-specific User Journeys

2.1. User Journey #7 - Build an image for a GenAI use case



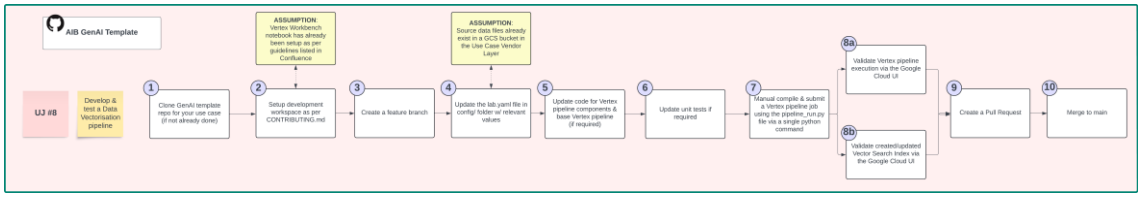
Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Clone GenAI Template repo	Clone the template repo for your GenAI use case & leverage the boilerplate code & frameworks already provided by the template	
2	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Loading environment variables Detailed steps for the same are captured in the CONTRIBUTING.md of the repo	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster
3	Code changes in a feature branch	Once the development workspace has been completely setup, a developer can create a feature branch, refactor code	

#	Element	Description	Assumptions
		as required & commit changes to said feature branch The repo also contains unit tests related to all existing functionality in the `testing/` folder These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
4	Create a pull request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
5a	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	
5b	Create a Release tag	A new release tag needs to be created w/ relevant description supporting it. This tag should conform to best practices around tags, specifically of format: vMAJOR.MINOR.PATCH eg - A new release tag can be: • v1.2.0	
6a	CI process automatically builds & pushes an image to the build-nl Use Case Vendor layer project	Once the PR has been pushed to the `main` branch (see step 5a), the CI/CD pipeline is automatically triggered that: • Runs relevant unit testing & linting checks • Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-nl project with a `latest` tag along with the relevant git commit SHA value • Compiles relevant Vertex pipelines & stores them in GCS (more details to follow in subsequent user journeys) • Create/update a Cloud Scheduler job for the compiled Vertex pipeline (more details to follow in subsequent user journeys)	Cloud Build Trigger exists in the Use Case Vendor layer build-nl project to run this CI/CD pipeline This is a one-time manual activity & needs to be implemented once for every GenAI use case

#	Element	Description	Assumptions
6b	CI process automatically builds & pushes an image to the build-live Use Case Vendor layer project	Once a new release tag has been created (see step 5b), the CI/CD pipeline is automatically triggered that: • Runs relevant unit testing & linting checks • Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-live project with the release tag attached to the image • Compiles relevant Vertex pipelines & stores them in GCS (more details to follow in subsequent user journeys) • Create/update a Cloud Scheduler job for the compiled Vertex pipeline (more details to follow in subsequent user journeys)	Cloud Build Trigger exists in the Use Case Vendor layer build-live project to run this CI/CD pipeline This is a one-time manual activity & needs to be implemented once for every GenAI use case

2.2. User Journey #8 - Develop & Test a Data Vectorisation Pipeline



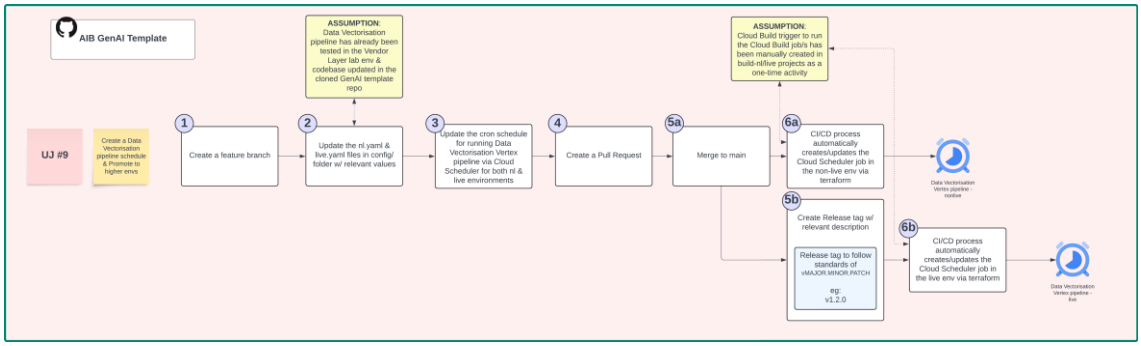
Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Clone GenAI Template repo	Clone the template repo for your GenAI use case & leverage the boilerplate code & frameworks already provided by the template	
2	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Loading environment variables Detailed steps for the same are captured in the CONTRIBUTING.md of the repo	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex

#	Element	Description	Assumptions
			Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster
3	Create a feature branch	Create a feature branch on the cloned repo to build & test relevant Data Vectorisation pipeline functionality	
4	Update the lab.yaml file in config/ w/ relevant values	In addition to boilerplate code to build a Vertex pipeline for Data Vectorisation & ingest documents to a Vector Database (Vertex Vector Search), this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances For testing purposes, the lab.yaml file needs to be updated w/ relevant values catering to: • Project ID • Region • Relevant Google Cloud Storage URIs • Relevant Vertex Pipeline configuration • Vertex Vector Search fields The template repo contains sample YAML files for reference	Source data files to be ingested into a Vector Database already exist in a relevant Google Cloud Storage bucket in the relevant Use Case Vendor layer project
5	Update code for Vertex pipeline components & the pipeline (if required)	Once the configuration-based YAML files have been updated, base logic in the `ingestion_workflow/pipeline/` folder can be updated if absolutely required. This folder contains generic logic for Vertex Pipeline components that include: • Fetching documents • Processing documents	

#	Element	Description	Assumptions
		• Creating or Updating a Vertex Vector Search index (Vector Database) • Updating pipeline details All components above contain core logic to maintain & audit the document lifecycle of processed documents	
6	Update unit tests (if required)	The repo contains unit tests related to all existing functionality in the `testing/` folder These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
7	Manually compile & submit a Vertex Pipeline job	Once all changes are in place, the Vertex pipeline can be manually submitted via a single python command - `python ingestion_workflow/pipeline/pipeline_run.py` to test updated functionality	
8a	Validate Vertex Pipeline execution	Once the above-mentioned python command has been executed in a development workspace, the Terminal will provide a link for the running pipeline OR the same can be found in the Vertex Pipeline UI Once submitted & executed, this pipeline should be validated via the UI to verify updated functionality & configuration (if any)	
8b	Validate create Vector Database	Once the Vertex pipeline has been submitted & successfully executed, a Vertex Vector Search index will be created/updated w/ embeddings of the ingested documents. The same should be validated from a use case specific standpoint	
9	Create a Pull Request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
10	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.3. User Journey #9 - Create a Data Vectorisation Pipeline Schedule & Promote to higher environments



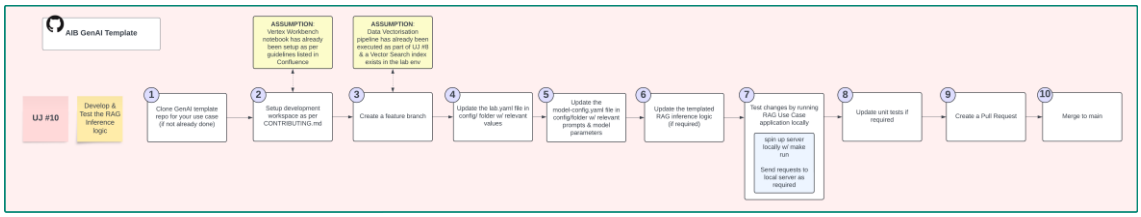
Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Create a feature branch	Create a feature branch on the cloned repo to plug in relevant configurations to enable promotion	
2	Update the nl.yaml + live.yaml files in config/ w/ relevant values	In addition to boilerplate code to build a Vertex pipeline for Data Vectorisation & ingest documents to a Vector Database (Vertex Vector Search), this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances For promotion purposes, the nl.yaml & live.yaml files need to be updated w/ relevant values catering to: • Project ID • Region • Relevant Google Cloud Storage URIs • Relevant Vertex Pipeline configuration • Vertex Vector Search fields The template repo contains sample YAML files for reference Additionally, configuration values setup in lab.yaml as part of pipeline testing in UJ #8 should be used as a reference	Data Vectorisation pipeline for the GenAI use case has already been tested as part of UJ #8 in the Use Case Vendor layer Lab project & relevant changes have already been plugged into the repo

#	Element	Description	Assumptions
3	Update the cron schedule for the Cloud Scheduler job	In addition to relevant configuration-based YAML files, separate files contain logic to configure the cron schedule of the Cloud Scheduler job to be created/updated cron schedules for both non-live & live Use Case Vendor layer projects will need to be updated separately	
4	Create a Pull Request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
5a	Merge to main	Post review & approval of the config-update/feature branch functionality, the same can be merged into the `main` branch	
5b	Create a Release tag	A new release tag needs to be created w/ relevant description supporting it. This tag should conform to best practices around tags, specifically of format: vMAJOR.MINOR.PATCH eg - A new release tag can be: • v1.2.0	
6a	CI process automatically creates/updates the Cloud Scheduler job in the non-live Use Case Vendor layer project	Once the PR has been pushed to the `main` branch (see step 5a), the CI/CD pipeline is automatically triggered that: • Runs relevant unit testing & linting checks • Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-nl project with a `latest` tag along with the relevant git commit SHA value • Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the git commit SHA value in the build-nl Use Case Vendor layer project • Create/update a Cloud Scheduler job for the compiled Vertex pipeline in the non-live Use Case Vendor layer project ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the	Cloud Build Trigger exists in the Use Case Vendor layer build-nl project to run this CI/CD pipeline This is a one-time manual activity & needs to be implemented once for every GenAI use case

#	Element	Description	Assumptions
		git commit SHA value attached to the image + the compiled pipeline GCS location from the previous stage	
6b	CI process automatically creates/updates the Cloud Scheduler job in the live Use Case Vendor layer project	Once a new release tag has been created (see step 5b), the CI/CD pipeline is automatically triggered that: • Runs relevant unit testing & linting checks • Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-live project with the release tag attached to the image • Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the release tag in the build-live Use Case Vendor layer project • Create/update a Cloud Scheduler job for the compiled Vertex pipeline in the live Use Case Vendor layer project ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the release tag attached to the image + the compiled pipeline GCS location from the previous stage	Cloud Build Trigger exists in the Use Case Vendor layer build-live project to run this CI/CD pipeline This is a one-time manual activity & needs to be implemented once for every GenAI use case

2.4. User Journey #10 - Develop & Test the RAG Inference logic



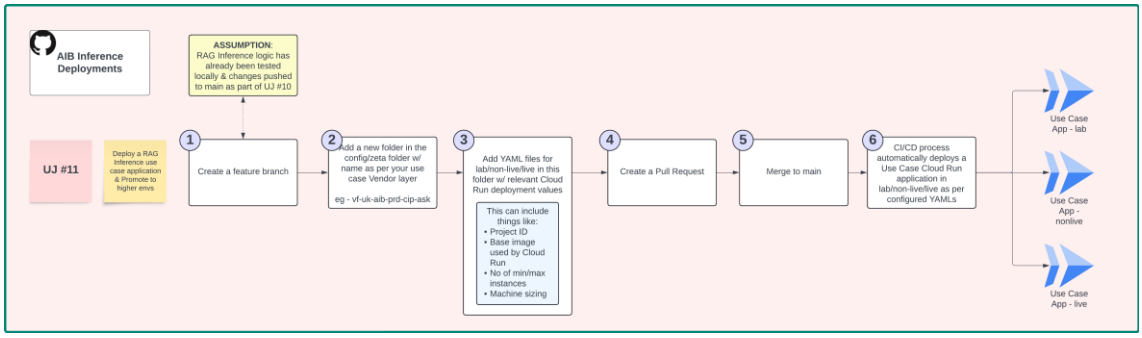
Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Clone GenAI Template repo	Clone the template repo for your GenAI use case & leverage the boilerplate code & frameworks already provided by the template	

#	Element	Description	Assumptions
2	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Loading environment variables Detailed steps for the same are captured in the CONTRIBUTING.md of the repo	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster
3	Create a feature branch	Create a feature branch on the cloned repo to build & test relevant RAG inference logic	Data Vectorisation pipeline has already been successfully executed as part of UJ #8 & a Vector Search index exists containing embeddings of the ingested documents
4	Update the lab.yaml file in config/ w/ relevant values	In addition to boilerplate FastAPI code to build out RAG inference logic, this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances For testing purposes, the lab.yaml file needs to be updated w/ relevant values catering to: • Project ID • Region • Relevant Google Cloud Storage URIs • Relevant Vertex Pipeline configuration • Vertex Vector Search fields The template repo contains sample YAML files for reference	
5	Update the model-config.yaml file	Furthermore, for inference purposes, a separate model-config YAML file captures inference-based configurations. This includes:	

#	Element	Description	Assumptions
	in config/ w/ relevant values	• LLM parameters around temperature, top_k, top_p etc • LLM Prompts as YAML fields • ... The template repo contains sample YAML files for reference	
6	Update the templated RAG inference logic	This template repo contains generic code to run a single LLM as part of its logic If a use case requires multiple LLM calls or any additional processing (reranking etc), the same needs to be reflected in the: • inference_workflow/src/ folder (contains source scripts for inference & retrieval) • inference_workflow/api/routers/ folder (contains FastAPI logic which imports source scripts & executes them in a predict router	
7	Test changes by running locally	Once required changes have been committed, the same can be tested locally by spinning up the Use Case Application locally using `make run` which spins up the FastAPI server on localhost:8000 by default Local testing can be executed by running a `curl` command to send API calls directly into the local server	
8	Update unit tests if required	The repo contains unit tests related to all existing functionality in the `testing/` folder These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
9	Create a Pull Request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
10	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.5. User Journey #11 - Deploy a RAG Inference Use Case Application & Promote to higher environments

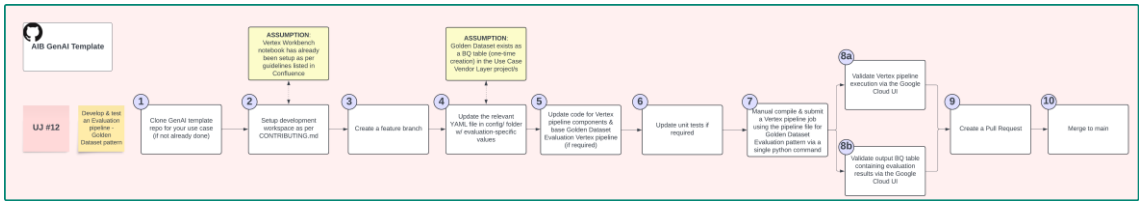


Based on the user journey outlined above, this UJ applies to the AIB Inference Deployments git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Create a feature branch	Create a feature branch on the cloned repo to plug in relevant configurations required for promotion	RAG inference logic has already been tested locally & changes have been pushed to the main branch as part of UJ #10
2	Add a new folder in the config/zeta folder	The base config/zeta folder needs a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances	
3	Add YAML files for relevant environments w/ relevant Cloud Run deployment values	Once the folder is created as part of step 2, relevant YAML files need to be added catering to separate environments (lab/non-live/live) These YAML files can include configuration values like: • Project ID • Image used by Cloud Run • No of min/max instances required by Cloud Run • Machine sizing of the Cloud Run instance An example YAML file has been included in the repo for reference	
4	Create a Pull Request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
5	Merge to main	Post review & approval of the feature branch configurations, the same can be merged into the `main` branch	
6	CI process automatically deploys a Use Case	Once the PR has been pushed to the `main` branch, the central CI/CD	Cloud Build Trigger exists in the AIB GCP

#	Element	Description	Assumptions
	Cloud Run application in lab/non-live/live environments of the Use Case Vendor layer	pipeline is automatically triggered that deploys a Use Case Cloud Run application for inference. This deployment depends on which YAML file/s have been pushed to main. eg - addition of live.yaml to vf-uk-aib-prd-cip-ask folder will deploy said Cloud Run app to the live project for AskHR	Orchestrate project to run this CI/CD pipeline This is a one-time manual activity & is already in place

2.6. User Journey #12 - Develop & Test an Evaluation Pipeline - Golden Dataset pattern



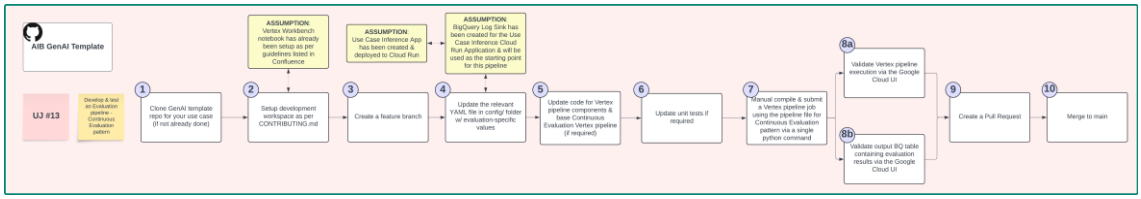
Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Clone GenAI Template repo	Clone the template repo for your GenAI use case & leverage the boilerplate code & frameworks already provided by the template	
2	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Loading environment variables Detailed steps for the same are captured in the CONTRIBUTING.md of the repo	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster
3	Create a feature branch	Create a feature branch on the cloned repo to build & test relevant LLM Evaluation - Golden Dataset pattern functionality	
4	Update the relevant YAML file/s in config/	This template repo contains generic code to execute LLM evaluation catering to 2 patterns:	Golden Dataset w/ relevant prompt-expected response

#	Element	Description	Assumptions
	w/ evaluation-based values	• LLM evaluation on a pre-set Golden Dataset (prompt + expected response) • Continuous LLM evaluation based on actual prompt-response pairs generated by the RAG Use Case Application In addition to this generic code to build Vertex pipelines for the above-mentioned 2 patterns, this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances For testing purposes, the relevant YAML file needs to be updated w/ evaluation-specific values catering to: • Project ID • Region • Evaluation metrics • Input Golden Dataset BQ location • Expected output BQ location to store evaluation results The template repo contains sample YAML files for reference	pairs already exist in a BQ table as a one-time activity
5	Update code for Vertex pipeline components & the pipeline (if required)	Once the configuration-based YAML files have been updated, base logic in the `evaluation/pipeline/` folder can be updated if absolutely required. This folder contains generic logic for Vertex Pipeline components that include: 1. Extracting data from a BQ table 2. Querying a Use Case app to generate relevant responses 3. Running LLM Evaluation using the Google GenAI Evaluation service 4. Storing evaluation results in the relevant output BQ table For the Golden Dataset Evaluation pattern, all 4 components apply & have been plugged into a generic Vertex pipeline	

#	Element	Description	Assumptions
6	Update unit tests (if required)	The repo contains unit tests related to all existing functionality in the `testing/` folder These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
7	Manually compile & submit a Vertex Pipeline job	Once all changes are in place, the Vertex pipeline can be manually submitted via a single python command to test updated functionality	
8a	Validate Vertex Pipeline execution	Once the above-mentioned python command has been executed in a development workspace, the Terminal will provide a link for the running pipeline OR the same can be found in the Vertex Pipeline UI Once submitted & executed, this pipeline should be validated via the UI to verify updated functionality & configuration (if any)	
8b	Validate output BQ table	Once the Vertex pipeline has been submitted & successfully executed, an output BQ table will be created containing relevant evaluation results The same should be validated from a use case specific standpoint	
9	Create a Pull Request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
10	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.7. User Journey #13 - Develop & Test an Evaluation Pipeline - Continuous Evaluation pattern



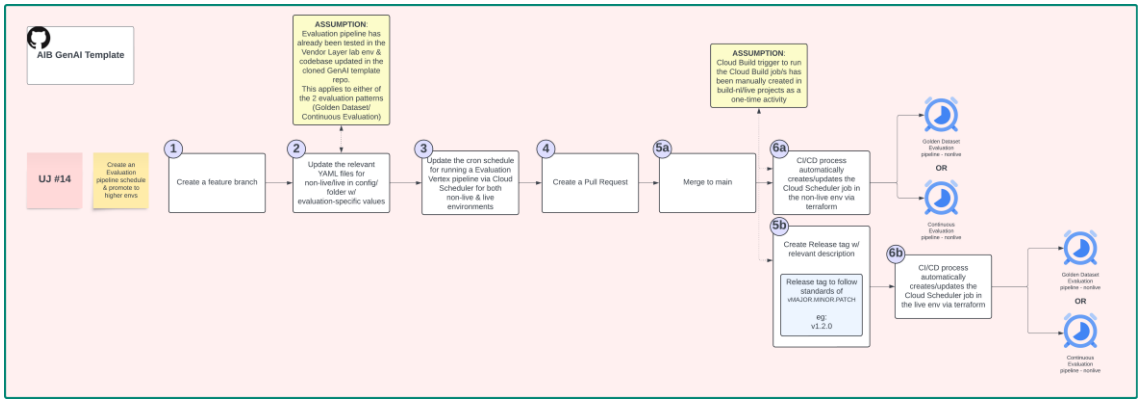
Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

#	Element	Description	Assumptions
1	Clone GenAI Template repo	Clone the template repo for your GenAI use case & leverage the boilerplate code & frameworks already provided by the template	
2	Setup a development workspace	Once the assumptions have been met, further setup is required. This includes: • Python package installation using poetry as a package manager • Loading environment variables Detailed steps for the same are captured in the CONTRIBUTING.md of the repo	Development workspace has been setup by: • Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster • VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster
3	Create a feature branch	Create a feature branch on the cloned repo to build & test relevant LLM Evaluation - Continuous Evaluation functionality	
4	Update the relevant YAML file/s in config/ w/ evaluation-based values	This template repo contains generic code to execute LLM evaluation catering to 2 patterns: • LLM evaluation on a pre-set Golden Dataset (prompt + expected response) • Continuous LLM evaluation based on actual prompt-response pairs generated by the RAG Use Case Application In addition to this generic code to build Vertex pipelines for the above-mentioned 2 patterns, this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances For testing purposes, the relevant YAML file needs to be updated w/ evaluation-specific values catering to: • Project ID • Region	• Use Case application has been created as part of UJ #10 + #11 & deployed as a Cloud run instance • BigQuery log sink has been created for the Use Case Inference Cloud Run application & contains information on the actual prompt-response pairs generated by the app

#	Element	Description	Assumptions
		• Evaluation metrics • Input BQ location containing actual prompt-response pairs generated by a Use Case Application • Expected output BQ location to store evaluation results The template repo contains sample YAML files for reference	
5	Update code for Vertex pipeline components & the pipeline (if required)	Once the configuration-based YAML files have been updated, base logic in the `evaluation/pipeline/` folder can be updated if absolutely required. This folder contains generic logic for Vertex Pipeline components that include: 1. Extracting data from a BQ table 2. Querying a Use Case app to generate relevant responses 3. Running LLM Evaluation using the Google GenAI Evaluation service 4. Storing evaluation results in the relevant output BQ table For the Continuous Evaluation pattern, components 1,3,4 apply & have been plugged into a generic Vertex pipeline	
6	Update unit tests (if required)	The repo contains unit tests related to all existing functionality in the `testing/` folder These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command. This Makefile abstracts away execution of key commands & can be referred for other aspects as well	
7	Manually compile & submit a Vertex Pipeline job	Once all changes are in place, the Vertex pipeline can be manually submitted via a single python command to test updated functionality	
8a	Validate Vertex Pipeline execution	Once the above-mentioned python command has been executed in a development workspace, the Terminal will provide a link for the running pipeline OR the same can be found in the Vertex Pipeline UI Once submitted & executed, this pipeline should be validated via the UI to verify updated functionality & configuration (if any)	
8b	Validate output BQ table	Once the Vertex pipeline has been submitted & successfully executed, an	

#	Element	Description	Assumptions
		output BQ table will be created containing relevant evaluation results The same should be validated from a use case specific standpoint	
9	Create a Pull Request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
10	Merge to main	Post review & approval of the feature branch functionality, the same can be merged into the `main` branch	

2.8. User Journey #14 - Create an Evaluation Pipeline Schedule & Promote to higher environments



Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

(This process applies generically to either of the 2 LLM Evaluation patterns)

#	Element	Description	Assumptions
1	Create a feature branch	Create a feature branch on the cloned repo to plug in relevant configurations to enable promotion	
2	Update the nl.yaml + live.yaml files in config/ w/ relevant values	This template repo contains generic code to execute LLM evaluation catering to 2 patterns: • LLM evaluation on a pre-set Golden Dataset (prompt + expected response) • Continuous LLM evaluation based on actual prompt-response pairs generated by the RAG Use Case Application In addition to this generic code to build Vertex pipelines for the above-mentioned 2 patterns, this template abstracts relevant updates to a	LLM Evaluation pipelines for either of the 2 patterns have already been tested as part of UJ #12 or UJ #13 in the Use Case Vendor layer Lab project & relevant changes have already been plugged into the repo

#	Element	Description	Assumptions
		configuration-based YAML file which exists in the **config/** folder This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances For promotion purposes, the relevant YAML files for non-live & live need to be updated w/ evaluation-specific values catering to: • Project ID • Region • Evaluation metrics • Input BQ location containing actual prompt-response pairs generated by a Use Case Application • Expected output BQ location to store evaluation results The template repo contains sample YAML files for reference Additionally, configuration values setup in the lab YAML as part of pipeline testing in UJ #13 should be used as a reference	
3	Update the cron schedule for the Cloud Scheduler job	In addition to relevant configuration-based YAML files, separate files contain logic to configure the cron schedule of the Cloud Scheduler job to be created/updated cron schedules for both non-live & live Use Case Vendor layer projects will need to be updated separately	
4	Create a Pull Request	Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review	
5a	Merge to main	Post review & approval of the config-update/feature branch functionality, the same can be merged into the `main` branch	
5b	Create a Release tag	A new release tag needs to be created w/ relevant description supporting it.	

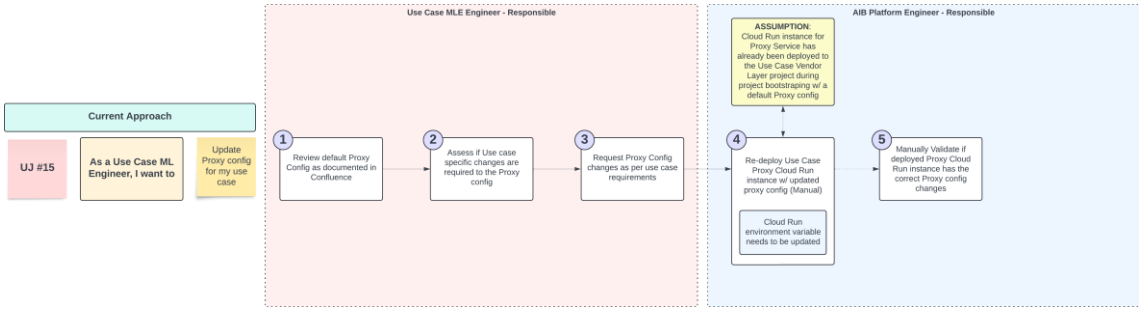
#	Element	Description	Assumptions
		This tag should conform to best practices around tags, specifically of format: vMAJOR.MINOR.PATCH eg - A new release tag can be: • v1.2.0	
6a	CI process automatically creates/updates the Cloud Scheduler job in the non-live Use Case Vendor layer project	Once the PR has been pushed to the `main` branch (see step 5a), the CI/CD pipeline is automatically triggered that: • Runs relevant unit testing & linting checks • Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-nl project with a `latest` tag along with the relevant git commit SHA value • Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the git commit SHA value in the build-nl Use Case Vendor layer project • Create/update a Cloud Scheduler job for the compiled Vertex pipeline/s in the non-live Use Case Vendor layer project ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the git commit SHA value attached to the image + the compiled pipeline GCS location from the previous stage	Cloud Build Trigger exists in the Use Case Vendor layer build-nl project to run this CI/CD pipeline This is a one-time manual activity & needs to be implemented once for every GenAI use case
6b	CI process automatically creates/updates the Cloud Scheduler job in the live Use Case Vendor layer project	Once a new release tag has been created (see step 5b), the CI/CD pipeline is automatically triggered that: • Runs relevant unit testing & linting checks • Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-live project	Cloud Build Trigger exists in the Use Case Vendor layer build-live project to run this CI/CD pipeline This is a one-time manual activity & needs to be implemented once for every GenAI use case

#	Element	Description	Assumptions
		with the release tag attached to the image • Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the release tag in the build-live Use Case Vendor layer project • Create/update a Cloud Scheduler job for the compiled Vertex pipeline/s in the live Use Case Vendor layer project ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the release tag attached to the image + the compiled pipeline GCS location from the previous stage	

1.2.3 User Journeys - Proxy Config Update

1. User Journey-specific Background & Context

2.1. User Journey #15 - Update Proxy Config for my use case (Current Approach)



2.2. User Journey #15 - Update Proxy Config for my use case (Future Roadmap)