***Summary of the solution pipeline being proposed and how it is better from the current and why it is needed.***

Current state:

* There is limited scope of transferring the learning from one client to another based on the client similarity quantum because of non-modularity of code and non-standard pipeline.
* Onboarding a new client will take a decent amount of time.
* The current solution is not equipped enough to get maximum business accuracy.
* System is not equipped enough to maintain the ML models at scale from the engineering side.

Challenges in using the current pipeline for scaling are:

* Pertaining to the variation of nature of data availability (Integrata vs Geneva).
* Lack of standardisation of input and output.
* Pre-processing inefficiency in the pipeline to handle high volume clients.
* Non-modularity of the code structure - Because of this, any client specific changes is a challenge to implement.
* High memory + compute required for current cartesian product architecture.

Current challenges from the engineering side:

* Users can’t see intermittent updates on ML engine progress.
* Change management dictated by the user involves a lengthy process of agreement over change (meeting), communication (via emails), change made by programmer (both Epowerx and Intertrust side), deployment, testing and verification.
* DB data type change dictated by Intertrust takes time on the Epowerx side to implement
* Intermittent logging not visible (covered earlier + Intertrust will give us a DB connection where we will upload all fully completed progress by any microservice, the DB in turn will point to the UI for user to see ML model logs)
* Moving from one server to another needs Epowerx help (Dockerization + Epowerx will provide a list of steps to follow to transfer server + codebase + dependency).
* Unreleased memory because of tightly coupled workers.

Solution we propose –

* The idea is to build an ML service which is scalable and easy to maintain from the engineering standpoint and can be used with minor tweaks by Intertrust after the end of this phase.
* We propose to split the solution into 2 parts.
  + Part 1 - To enhance the current ML pipeline (end to end) to boost the further usability of the already built models in production. In this part, we will build the pipeline with modular structure and robust standardisation covering solutions to all the challenges that we are currently facing for scaling.
  + Part 2 - Testing, validation and improvement of the above pipeline for the clients covered in Phase 1 and making it robust for scaling by targeting maximum business accuracy possible so that the service can be used extensively by the end user in production.

Note - We are targeting the maximum business accuracy for the clients of Phase 1 with building the robust pipeline first. This will make sure that we have a general pipeline which behaves most similar to similar clients and can be used easily with minor tweaks to onboard any new/existing client.

***Timelines and methodology to onboard new clients - both existing ITG client’s where past Recon data exists and completely new clients where we don’t have past data.***

For existing client –

* After creating a robust pipeline, an estimate of onboarding an existing client with minor tweaks is around 1-2 weeks.

For new client –

* Here we don’t have any history. So there are two possible paths –
  + 1st – Wait for 1 month to get the enough history to train any model.
  + 2nd – If we know beforehand that this client is similar to some of the clients on which ML was already trained then the it will take 1 week to get a decent break prediction model (around 70% accuracy). For commenting – Since there is no history, we can only give predictions if we know the most similar client to this new client. Only then predictions can be made. Else, we will have to wait for 1 month to get enough user history.

Process to make onboarding faster

***Handling exceptions completely new type of clients for onboarding which don’t match to Preclassied models + Onboarding process of clients with the help of business users***

For existing client –

* Knowing which all categories users use while doing reconciliation like UMR, UMT, UMB, and UCB?
* For commenting – Categorising historical comments into pre-defined categories.
* Information from the business team - if this client is similar to any other client business wise.

For new client –

* Knowing which all categories users are going to use while doing reconciliation like UMR, UMT, UMB, and UCB?
* For commenting – Since there is no history, we can only give predictions if we know the most similar client to this new client. Only then predictions can be made. Else, we will have to wait for 1 month to get enough user history.
* Information from the business team - if this client is similar to any other client business wise.

***How much models will learn from similar classified models and how much customisation as per business discussion.***

* Based on our experience, we had Soros and Weiss as the most similar clients. The learning was transferable at around 80 % variance. The only 20% drift was because of the client specific customization.
* For clients which are not very similar, for example Oaktree and Weiss, the transferable learning quantum was at around 50-60% variance.
* And for totally different clients, the models have to be totally different. Example Lombard and Schonfeld.

***Why single model is not feasible?***

* Because of so many variations in the input variables across clients and the nature of output variable (UMT, UMR, UMB, UCB), a single model is not feasible and also not efficient at all.
* Dividing the clients into clusters based on similarity is highly efficient for a large client base and building robust pipelines for those clusters is what we propose. Building individual models from those pipelines with minimum tweaking will serve as a scaling solution.
* Similarly for comments, the nature of input variables varies by a huge degree and a single model is unrealistic.

Note – For the recon problem, having multiple models is extremely efficient in order to scale. Model maintenance at scale will be targeted extensively from Epowerx by building the pipeline in such a way that it can be used/tweaked easily.

***Data science team requirement from ITG’s side for the model maintaince part.***

From ITG’s side, a team of 2-3 members consisting of data scientists/data analysts/data engineer is required to maintain the pipeline in the long run. Onboarding any new client will require this team to be familiar with the pipeline so that developing custom modules will not be a roadblock. We are proposing to write an extensive underlying codebase with modules which should be able to cover all variations in a future client. But the ITG team should have enough knowledge on how and where to fit the modules in a waterfall structure of our model architecture.

***Concerns around the commenting model.***

Current state and challenges -

* Concentration of comments and cyclic nature of comments is a huge variation across clients which does not make it a straightforward replication effort.
* Variation in the values of the input variables (Transaction type, Description, Asset Type Category etc.) which are responsible to decide which comment is going to be predicted.

What we propose -

* Designing a structure for the users on how to do mapping comment categories to actual comments on the historical data which can be used directly for any new client for onboarding. This is the most important step in terms of scaling.
* Creating a robust pre-processing pipeline which can be used at scale to determine the transfer learning (if possible) across clients.
* Building a pipeline which will require very less tweaking in order to build a new model for any new client so that the onboarding time is further minimized.

***Logging and exception handling capabilities proposed in new proposal and how it make things better in maintaince and trouble shooting.***

Logging:

*Proposed state*

Separate logging handlers will be available in modules for development phase and production phase.

The development phase handler will have more information regarding :

* Db connections
* Rabbitmq messaging
* Current step of model execution
* Timestamps of each execution
* Memory used in each step
* Docker information during container spin up
* Jenkins testing logs to see code defects

The production phase logger will have information regarding :

* Data Type errors
* Jenkins testing logs to see code defects
* Db compatibility of our table to be pushed
* Check if RabbitMQ exchange is taking consumers or if the RabbitMQ exchange is down
* All exceptions raised by code execution

The logging module will have a database connection where all logs will be inputted with timestamps. This database connection will point to the UI for the user to see logs.

*Current state* and how its difficult to understand logs:

Currently, all logs are generated on the production and prod parallel server in flat files. So developers have to log in to the server to check the activity. Also, no separate handlers are made, so logs are quite exhaustive. In such a case, information extraction from logs is cumbersome. Logs also do not have a UI capability as logs are not pushed into a database where UI can pick it from. So loggin into servers is required for checking the logs. This makes the process time comsuming.

Exception handling:

*Proposed state*

Separate modules for each sub model (like separate module for closing, pairing, commenting etc) will be developed. Each module will cover all scenarios of any type of error the code can generate, along with better explanation of the reason behind the error. This will make it easier to understand exceptions. Also, if an exception occurs for empty dataframe errors, then code will continue to the next code piece, so that code breakage does not occur. Warning messages will be generated in such a case of empty dataframes which will be visible to the user on the UI.

*Current state*

The code has exceptions which make sure that the code executes for any situation, with exceptions written after exhaustive testing with business users. But the exceptions are not for separate modules. So it’s difficult to understand for a new programmer unfamiliar with the code. Custom messages explaining warnings are limited as well. Any exception is stored in flat files on server instead of being shown on the UI.

***How dockerization with Jenkins will streamline entire pipeline***

*Current state of*

1. *Dependency management*

To deploy on a new server, EpowerX data engineer has to setup the entire server along with IT department’s help, which takes half a week. The global environment in that server should not be changed to maintain a constant state of code deployment. Any major change to the global namespace can alter libraries being used by the ML model, resulting in namespace conflicts and code rot.

1. *Code deployment on new server*

To deploy on a new server, or to make new changes to the code, there is no current indicator for claiming that the code will run flawlessly due to no testing module covering code breakage scenarios. So programmer has to wait for the code to perform in the UI deployment phase, instead of knowing beforehand that the code will work.

*Proposed state of*

1. *Dependency management using Docker*

We will make separate docker containers for each model based on our researched design pattern for the code. Once a design pattern has been agreed upon and modules are containerized, then the code deployment on any server will not need the help of EpowerX data engineer to setup the server. The docker containers will be a separate container where installations will be controlled entirely by the dockerfile which will come with the docker image. So even in the case of global namespace variables and libraries being messed around with by separate users, the ML code will not break.

1. *Code deployment using Jenkins testing module*

Once the modules are covered by a testing module deployed in Jenkins, users can test if all tests are passing for sanity checks on the code in case of code changes. If tests pass, the user can be quite sure that the code will work in UI production/development environment. If a test fails, the user can correct the faulty module without waiting for UI testing. So testing time will be reduced, resulting in streamlining of the code deployment process.