Modelo de predicción en VertexAI

1. Entrenamiento de los modelos

El modelo de predicción de la secuencia de actividades y el de la duración se entrenan y se despliegan por separado.

1.1. Preprocesamiento y preparación de los datos:

Se asume que los datos de partida están en el mismo formato que el de trace_activities.csv obtenido del export metrics desde inverbis. El preprocesamiento consiste de los siguientes pasos:

1. Conversión de las columnas 'start' y 'end' a datetime.
2. Obtención de la **duración de las actividades y los arcos**.
3. Obtención de **parámetros** y **atributos**: número máximo de actividades en una traza, número de actividades únicas, atributos booleanos y categóricos.
4. **Agrupación** por 'traceId' y creación de **listas de actividades** (**secuencia**) y duraciones.
5. **Truncamiento** de las trazas en puntos aleatorios y división entre traza truncada y traza restante.
6. Obtención de las duraciones totales de la parte truncada y la parte restante
7. **Tokenización** de las actividades y de las transiciones.
8. **Tokenización** de los atributos seleccionados
9. **Padding** de todas las secuencias (se rellenan de 0s hasta el número máximo de actividades en una traza)

Una vez procesados los datos han de almacenarse los Tokenizadores, y los valores del número de actividades únicas y la longitud máxima de las secuencias, así como los atributos seleccionados, ya que se hará uso de ellos en la parte de predicción una vez el modelo esté entrenado.

El resultado es un dataframe con las siguientes columnas (más los atributos tokenizados en cada caso), se muestra un ejemplo de fila en cada caso:

- **traceId**: ‘01be9719-6b2f-4200-b72c-54d085f8af1c’
- **sequence_tokenized**: [1, 2, 3, 4, 5]
- **activity_durations**: [54, 50, 188, 110, 57]
- **transition_durations**: [0, 0, 0, 21256, 0]
- **total_duration**: 21715
- **truncated_tokenized**: [1, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- **remaining_tokenized**: [4, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- **trunc_point**: 3

- **truncated_durations**: [54, 50, 188]
- **remaining_durations**: [110, 57]
- **truncated_transitions**: [0, 0]
- **remaining_transitions**: [0, 21256, 0]
- **truncated_total_duration**: 292
- **remaining_total_duration**: 21423

Posteriormente se dividen los datos en train/test con un ratio de 80/20 y se preparan las columnas que alimentarán el modelo:

Entrada: truncated_tokenized + atributos seleccionados tokenizados
Label (**target**): remaining_tokenized para el modelo de predicción de secuencia o remaining_total_time para el modelo de predicción de la duración

Se convierten en **numpy arrays** y (para el caso de la predicción de la secuencia) se hace un **one-hot encoding de las secuencias restantes**.

1.2 Modelo de predicción de secuencia

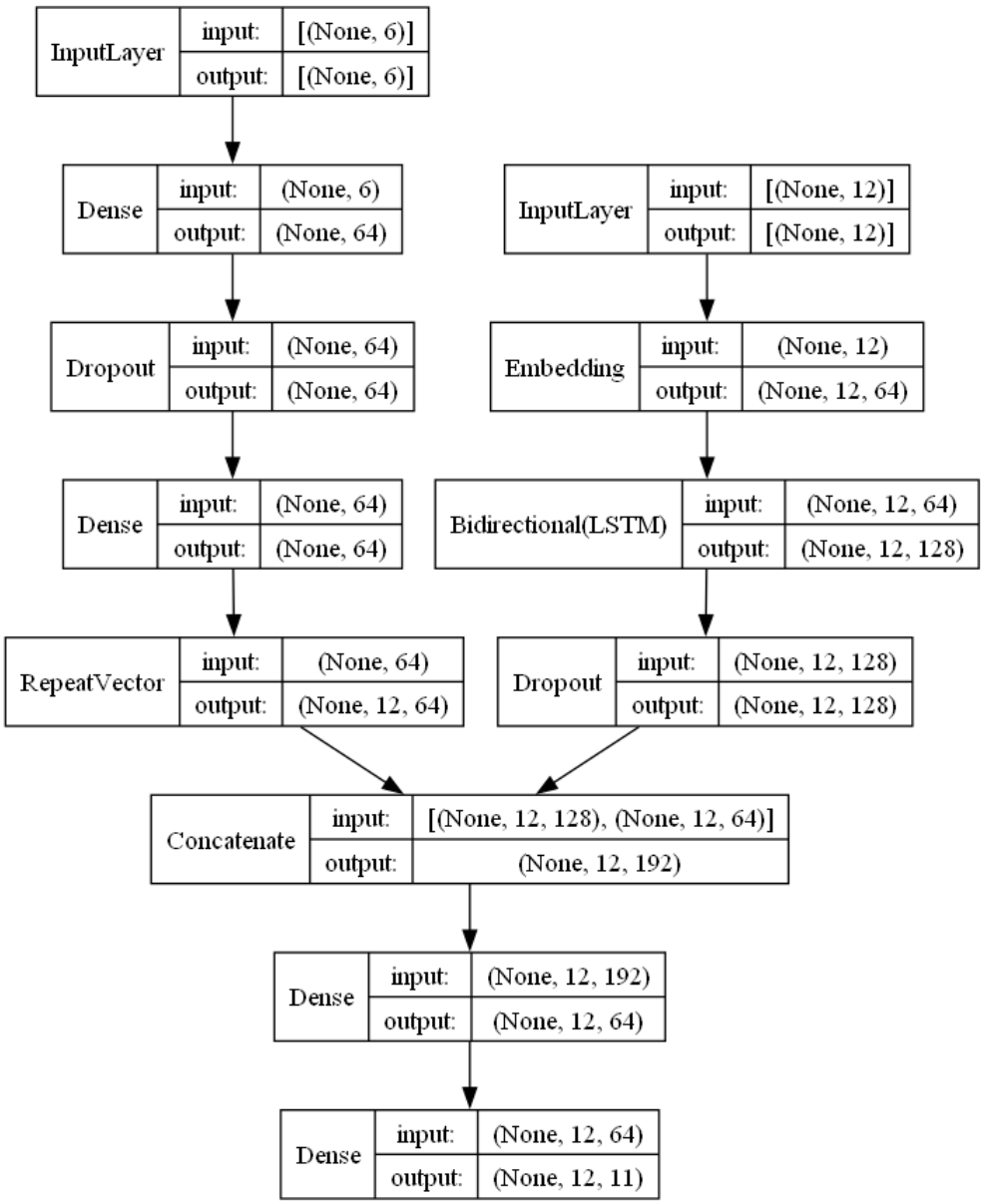
Para el modelo de predicción de secuencias se definen dos métricas personalizadas:

- **mask_acc**: calcula la precisión en las posiciones donde el valor verdadero no es cero, es decir, excluye las posiciones de "padding" en la secuencia.
- **seq_acc**: calcula la precisión de la secuencia completa; cuántas veces toda la secuencia predicha es igual a la secuencia real.

La arquitectura del modelo utilizado es la siguiente:

- **Input de Secuencia**:
    - **Entrada**: truncated_tokenized ya procesado
    - Capa de Incrustación (**Embedding**): Convierte los tokens de entrada en vectores de dimensión 64.
    - **LSTM Bidireccional**: Capa LSTM con 64 unidades, que devuelve secuencias. Esto permite que la LSTM tenga en cuenta tanto el contexto pasado como el futuro.
    - **Dropout**: Una capa de dropout con una tasa de 0.15 para regularizar y prevenir el sobreajuste.
- **Input de Atributos:**
    - **Input**: Atributos tokenizados ya procesados
    - **Capas Densas**: Dos capas densas, ambas con 64 unidades y activación ReLU.
    - **Dropout**: Después de la primera capa densa, hay una capa de dropout con una tasa de 0.15.

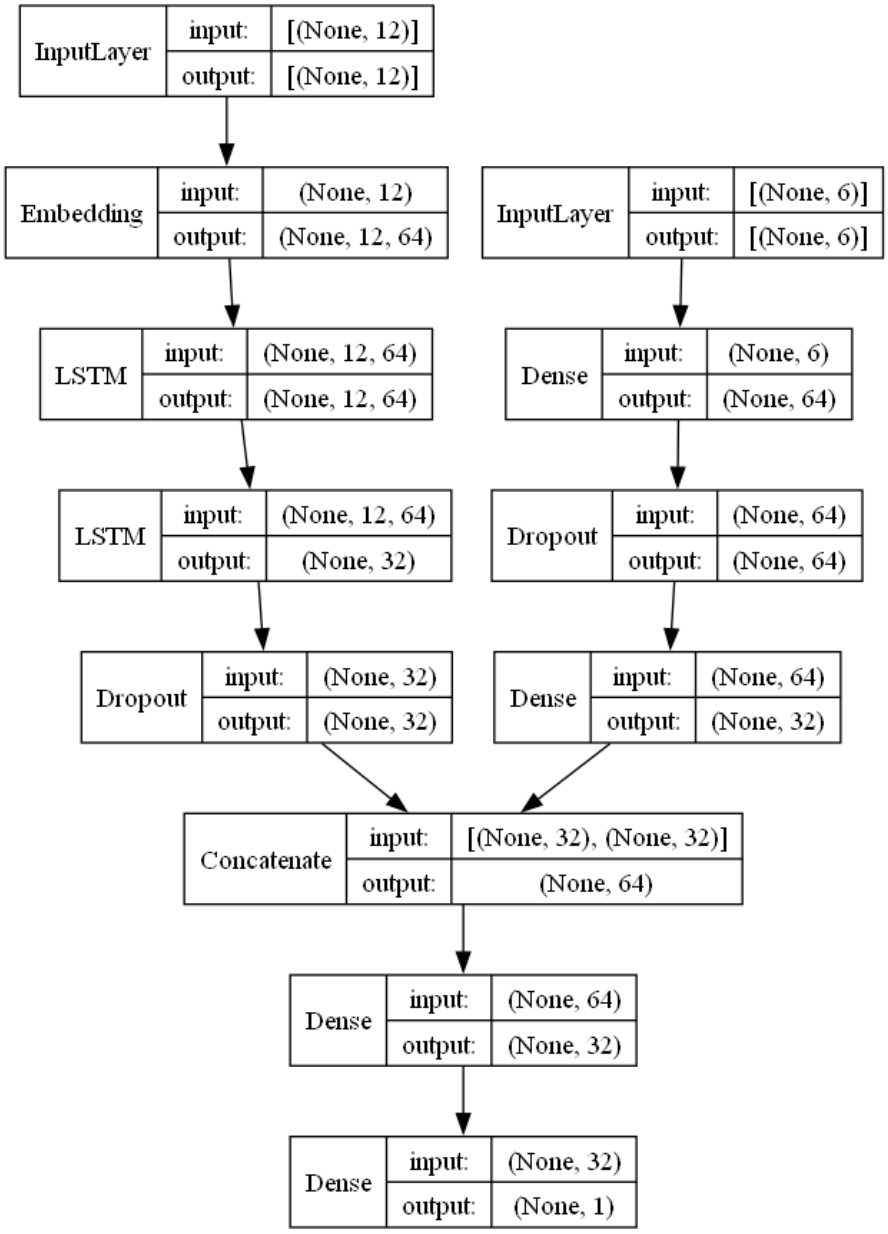
    - **RepeatVector**: Repite los atributos para que tengan la misma forma que la salida de la LSTM.
- **Combinación de las Salidas de Secuencia y Atributos:**
    - **Concatenación**: Combina la salida de la LSTM y los atributos.
    - **Capa Densa**: Una capa densa con 64 unidades y activación ReLU.
- **Output**:
    - **Densa**: Una capa densa con un número de unidades igual al número de actividades únicas y una activación softmax.
- **Compilación del Modelo:**
    - **Pérdida**: Categorical Cross Entropy.
    - **Optimizador**: Adam con una tasa de aprendizaje de 0.005.
    - **Métricas**: Las métricas personalizadas mask_acc y seq_acc.



1.3 Modelo de predicción de duración

En este caso no se utilizan métricas personalizadas, la arquitectura es la siguiente:

- **Entrada de Secuencia:**
  - **Entrada**: Una secuencia de tokens.
  - **Capa de Incrustación** (Embedding): Convierte los tokens de entrada en vectores de dimensión 64.
  - **LSTM**: Primero, una capa LSTM con 64 unidades que devuelve secuencias. Luego, otra capa LSTM con 32 unidades.
  - **Dropout**: Una capa de dropout con una tasa de 0.2 para regularizar y prevenir el sobreajuste.
- **Entrada de Atributos:**
  - **Entrada**: Una entrada de atributos adicionales.
  - **Capas Densas**: Dos capas densas, la primera con 64 unidades y activación ReLU y la segunda con 32 unidades y activación ReLU.
  - **Dropout**: Después de la primera capa densa, hay una capa de dropout con una tasa de 0.2.
- **Combinación de las Salidas de Secuencia y Atributos**:
  - **Concatenación**: Combina la salida de la LSTM y los atributos adicionales.
  - **Capa Densa:** Una capa densa con 32 unidades y activación ReLU.
- **Capa de Salida de Regresión**:
  - **Densa**: Una capa densa con 1 unidad y una activación lineal para la tarea de regresión.
- **Compilación del Modelo:**
  - **Pérdida**: Error absoluto medio.
  - **Optimizador**: Adam con una tasa de aprendizaje de 0.01.



2. Predicción en trazas incompletas

Para la predicción en trazas incompletas los datos se procesan de la misma manera que en la fase de entrenamiento salvo por el truncamiento de trazas, ya que partimos de las trazas truncadas. Para ello se hace uso de los **Tokenizadores** para las **actividades** y los **atributos** obtenidos en la fase de entrenamiento así como de los valores de los parámetros necesarios (**número de actividades únicas y longitud máxima de secuencia**) y los **atributos seleccionados.**

Una vez obtenidas las predicciones de la secuencia restante se utiliza el Tokenizador de las actividades para deshacer el encodeado y obtener el nombre de las actividades.

Para las predicciones de la duración, primero se obtiene la duración total (*t*) para cada traza incompleta y posteriormente se calcula el riesgo (*r)* de incumplimiento del SLA (por defecto se toma la mediana del proceso). El riesgo se divide en tres niveles dependiendo de cuánto se desvíe la duración predicha del SLA:

- $r = 1$ si $t \in [SLA,\ 1.5\ \cdot\ SLA]$
- $r = 2$ si $t \in [1.5\ \cdot\ SLA,\ 2\ \cdot\ SLA]$
- $r = 3$ si $t \in [2\ \cdot\ SLA,\ \infty]$.

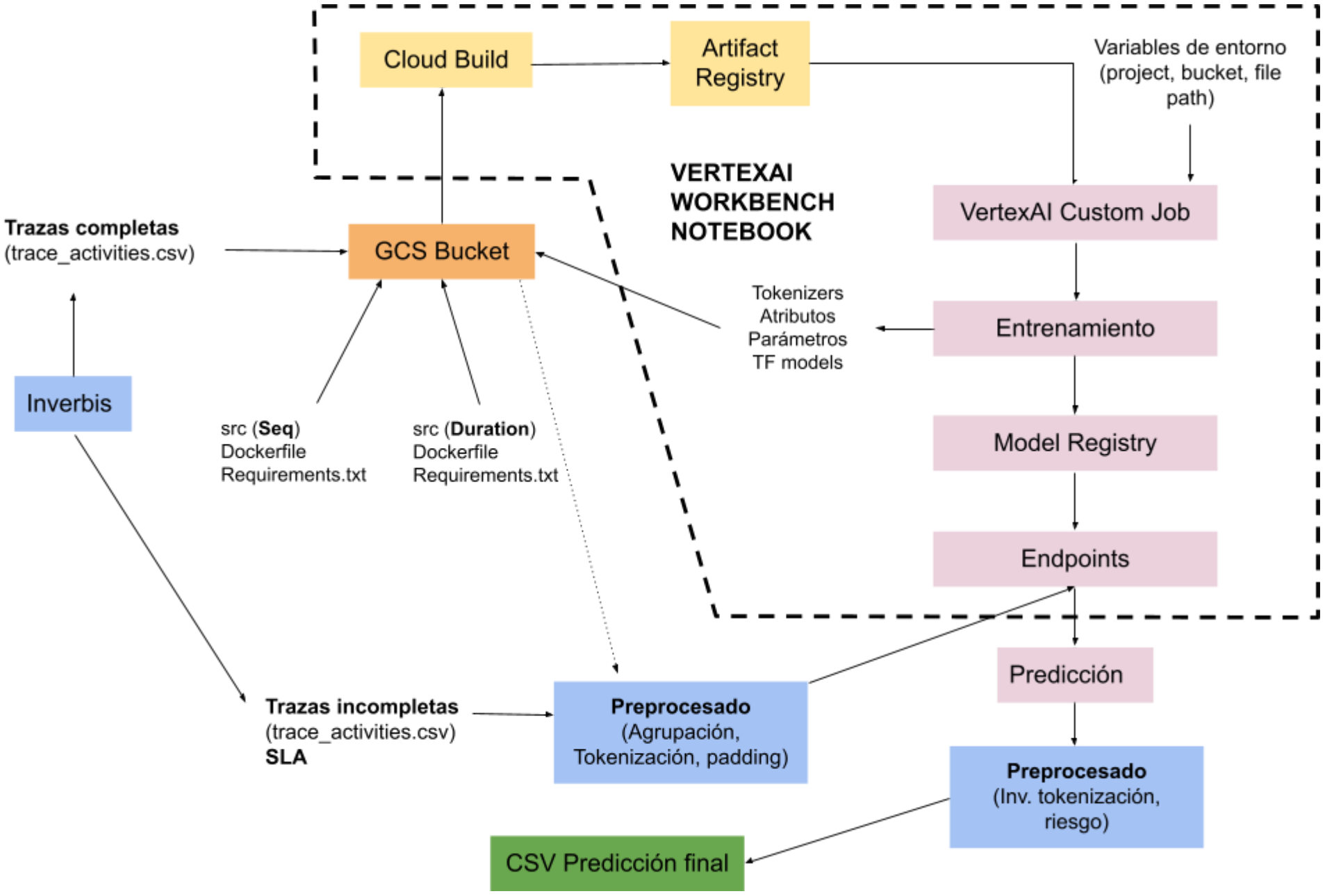
El riesgo predicho de esta manera se compara con el riesgo 'real' de las trazas truncadas, teniendo en cuenta la duración real de la parte restante y se utiliza como métrica para el modelo la accuracy en la predicción del riesgo.

3. Despliegue de los modelos en GCP VertexAI

El despliegue del modelo se hace desde el Workbench de VertexAI utilizando un notebook. Inicialmente necesitamos subir todos los archivos del modelo (scripts .py, requirements.txt y Dockerfile) en Bucket del proyecto. Desde el notebook se realizan los siguientes pasos:

1. **Configuración Inicial**: se determinan y configuran variables esenciales: PROJECT_ID, BUCKET, REGION, BUCKET_path y SERVICE_ACCOUNT utilizando comandos de gcloud.
2. **Configuración del Repositorio**: Se establece la dirección del repositorio Docker en Artifact Registry con la variable REPOSITORY.
3. **Construcción de la Imagen Docker:** se configura y utiliza la API de Cloud Build de Google Cloud para crear una imagen Docker que realiza:
    a. Recuperación del código fuente desde Google Cloud Storage.
    b. Construcción de la imagen Docker en el repositorio de Artifact Registry.
    c. Etiquetado de la imagen construida.

    Se envía la tarea de construcción y se obtiene una respuesta.
4. **Verificación del Estado de la Construcción**: se verifica el estado de la operación de construcción y se visualizan los artefactos generados.
5. **Inicialización de Vertex AI**: Se inicializa Vertex AI con el PROJECT_ID y REGION.
6. **Configuración de Variables de Entorno**: Se configuran variables de entorno GCS_BUCKET, PROJECT_ID y DATA_PATH para su uso en el entrenamiento.
7. **Definición del Trabajo de Entrenamiento**: Se define un trabajo de entrenamiento utilizando CustomContainerTrainingJob en Vertex AI.
    Se configura el trabajo de entrenamiento con la URI del contenedor, una imagen para servir el modelo y un bucket de almacenamiento temporal.
8. **Lanzamiento del Trabajo de Entrenamiento**: Se lanza el trabajo de entrenamiento en Vertex AI con las configuraciones definidas anteriormente.
9. **Creación de un Endpoint y Despliegue del Modelo**: Se crea un nuevo endpoint en Vertex AI. Se despliega el modelo entrenado en el endpoint creado.


Cloud Build
Artifact Registry
Variables de entorno (project, bucket, file path)
VERTEXAI WORKBENCH NOTEBOOK
Trazas completas (trace_activities.csv)
GCS Bucket
VertexAI Custom Job
Tokenizers
Atributos
Parámetros
TF models
Entrenamiento
Inverbis
src (Seq)
Dockerfile
Requirements.txt
src (Duration)
Dockerfile
Requirements.txt
Model Registry
Endpoints
Predicción
Trazas incompletas (trace_activities.csv) SLA
Preprocesado (Agrupación, Tokenización, padding)
Preprocesado (Inv. tokenización, riesgo)
CSV Predicción final

4. Consideraciones

- Como alternativa se pueden almacenar los datos de trazas completas/incompletas en Google **BigQuery** (Data Warehouse)
- Hay que pensar dónde se realizarán el procesado de las trazas incompletas y de las predicciones
- Trasladar Workbench - Ejecutable
- Antes de poner en producción hay que un testeo a fondo de los modelos para evitar posibles errores (trazas inacabadas más largas que la longitud máxima?). Unit Tests etc.
- ¿Devolver las **métricas** del proceso para que el usuario sepa el grado de fiabilidad de la predicción?