

PROYECTO: OPTIMIZACIÓN DE DIÁLOGOS DST PARA EXPERIENCIA DEL CLIENTE EN E-CONTACT

DDSDI - N°18



En la cúspide de la revolución conversacional, el proyecto de e-Contact abre caminos hacia una interacción cliente-servicio sin precedentes. Centrándonos en el flujo comunicativo entre la inteligencia artificial y los usuarios finales, este informe despliega el marco completo de un modelo conversacional avanzado. Destacamos el 'modelo BERT multitarea para el seguimiento del estado del diálogo guiado por esquemas', una arquitectura que redefine la agilidad y precisión en el seguimiento de diálogos.



ÍNDICE

Resumen ejecutivo

Se detalla la introducción y el contexto a implementar en el proyecto.

3

Flujo del Modelo Conversacional y Definiciones Claves

4

Modelo Seleccionado: BERT

5

Arquitectura del Modelo

6

Estructura de Datos

8

Despliegue Y Dependencias

10-11

Desarrollo del Modelo

12-25

Bibliografía

Listado completo de todos los materiales empleados para realizar un trabajo

26

RESUMEN EJECUTIVO

E. Contact

Introducción

El avance vertiginoso en la inteligencia artificial y el procesamiento del lenguaje natural ha impulsado el desarrollo de sistemas de Dialog State Tracking (DST), una piedra angular en la mejora de la experiencia del cliente en plataformas digitales. La capacidad de gestionar y dar seguimiento a los estados de un diálogo en tiempo real representa un desafío crítico y una oportunidad para empresas que buscan optimizar sus interacciones con los clientes. Este informe destaca los esfuerzos y descubrimientos realizados en el proyecto "Estudio del Estado del Arte y Propuesta de un Pseudo-Modelo de Diálogos Inteligentes en DST".

Objetivo

El propósito principal del proyecto es investigar las metodologías y tecnologías actuales en DST y desarrollar un pseudomodelo innovador que mejore significativamente la eficiencia y la calidad del servicio al cliente para e-Contact. Se busca crear un sistema que no solo responda con precisión a las consultas de los usuarios sino que también proporcione una experiencia de usuario fluida y personalizada.

Conclusion

El análisis sistemático de la literatura y el estudio detallado de modelos como el propuesto por Kapelonis et al. han culminado en la construcción de un pseudomodelo de DST sólido y multifuncional. Este modelo incorpora la eficiencia y la precisión necesarias para mejorar la experiencia del cliente en e-Contact, demostrando ser una herramienta poderosa para el seguimiento de diálogos. Con la capacidad de generalizar a nuevos dominios y servicios, el pseudomodelo se sitúa como un facilitador clave para la expansión y sostenibilidad del servicio al cliente en la era digital, marcando un avance significativo en el campo y estableciendo un nuevo estándar en la atención al cliente automatizada.

Breve Historia

La necesidad de mejorar la interacción entre los clientes y los servicios de atención al cliente ha conducido a la exploración de modelos DST avanzados. La revisión de 15 artículos académicos seleccionados de bases de datos prestigiosas ha proporcionado un compendio de conocimientos que abarcan desde algoritmos y modelos hasta herramientas y frameworks eficaces para la implementación de DST. Un hito destacado en este viaje fue el análisis del artículo "Un modelo BERT multitarea para el seguimiento del estado del diálogo guiado por esquemas" de Kapelonis et al., que ofreció una metodología de vanguardia aplicable a nuestro pseudomodelo.

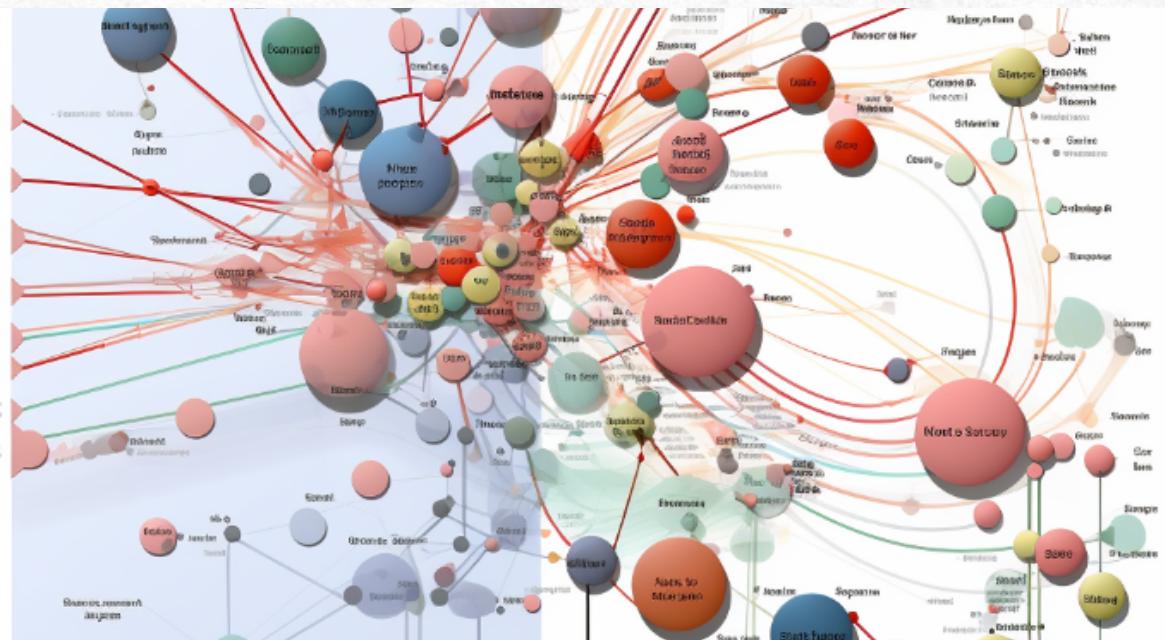
FLUJO DEL MODELO CONVERSACIONAL Y DEFINICIONES CLAVES

E.Contact

El flujo del modelo conversacional en nuestro proyecto se estructura sobre la premisa de interpretar y gestionar dinámicamente las necesidades del usuario a través de un diálogo fluido y contextual. En este marco, se destacan varias definiciones clave:

- **Dialog State Tracking (DST):** Se refiere al proceso de seguir el estado de un diálogo, es decir, mantener un registro de las intenciones del usuario y los parámetros relevantes a lo largo de una conversación.
 - **Intención del Usuario (User Intent):** Es el objetivo o la necesidad que el usuario busca satisfacer a través del diálogo, como reservar una mesa o comprar un boleto.
 - **Espacios Solicitados (Requested Slots):** Son las piezas específicas de información que el usuario necesita, tales como la hora de una reserva o el número de boletos a comprar.
 - **Llenado de Espacios (Slot Filling):** Es el acto de capturar y almacenar los valores proporcionados por el usuario para los espacios solicitados.
 - **Transferencia de Espacios (Slot Carryover):** Es la habilidad del sistema para mantener la relevancia de los espacios a lo largo de varios turnos de diálogo o incluso a través de diferentes dominios.
 - **Zero-Shot Learning:** La capacidad del modelo para comprender y actuar en dominios para los cuales no ha sido explícitamente entrenado, lo que permite una mayor generalización y escalabilidad.

Fig. 1 Referencia de esquemas multitask



MODELO SELECCIONADO: BERT MULTITAREA PARA EL SEGUIMIENTO DEL ESTADO DEL DIÁLOGO GUIADO POR ESQUEMAS

E.Contact

El modelo seleccionado para nuestro pseudomodelo de DST es el 'modelo BERT multitarea para el seguimiento del estado del diálogo guiado por esquemas'. Este modelo representa un enfoque integrado que maneja simultáneamente las tareas de predicción de intención, predicción de espacios solicitados y llenado de espacios. La eficiencia del modelo se ve mejorada gracias a una codificación parsimoniosa de la historia del diálogo y los esquemas de servicios, lo cual mejora tanto el rendimiento como la eficiencia computacional. Evaluado sobre el conjunto de datos Schema-Guided Dialogue (SGD), el modelo ha demostrado superar a sistemas previos en términos de precisión y eficiencia computacional. Su arquitectura basada en BERT, reconocida por su capacidad de procesamiento contextual del lenguaje natural, es particularmente adecuada para la tarea de DST, ya que puede capturar la complejidad y los matices de los diálogos humanos. Con esto, el modelo seleccionado no sólo avanza en el estado del arte sino que también impulsa la funcionalidad y la adaptabilidad en el ámbito de los sistemas de diálogo orientados a tareas.

Fig. 2 Cada turno de usuario se calcula el estado del diálogo para cada servicio implicado.



ARQUITECTURA DEL MODELO MULTITASK BERT

E.Contact

La arquitectura del modelo multitarea BERT (Multi-Task BERT Model) que se propone para el seguimiento del estado del diálogo (DST) se caracteriza por su capacidad para realizar simultáneamente varias tareas esenciales para la comprensión y la gestión efectiva de los diálogos orientados a tareas. A continuación, se detalla la arquitectura de este modelo:

Componentes Clave de la Arquitectura:

- **Cabezas de Clasificación (Classification Heads):**

El modelo incluye varias cabezas de clasificación para las tareas distintas de DST:

- Predicción de Intención (Intent Prediction): Determina la intención activa del usuario.
- Predicción de Espacios Solicitados (Requested Slot Prediction): Identifica qué espacios de información el usuario está solicitando.
- Llenado de Espacios (Slot Filling): Asigna valores a los espacios basándose en la entrada del usuario.
- Transferencia de Espacios (Slot Carryover): Conserva la información relevante de espacios a través de turnos en el diálogo.

- **Codificación de Entradas (Input Encoding):**

La entrada al modelo BERT base y a las cabezas de clasificación se codifica de manera que refleja la historia del diálogo y la secuencia de la interacción actual:

- Codificación por Colores: Se utiliza una codificación por colores en las entradas para facilitar la correspondencia con las partes de la secuencia de entrada.
- Historia del Diálogo: Se incluye la codificación de la historia del diálogo para proporcionar contexto a las predicciones del modelo.

- **Consideraciones de Diseño:**

- Eficiencia: La arquitectura está diseñada para ser eficiente, minimizando la necesidad de procesar toda la historia del diálogo para cada predicción.
- Generalización: El modelo está diseñado para adaptarse a nuevos dominios y servicios, lo que permite una generalización y escalabilidad más amplias.

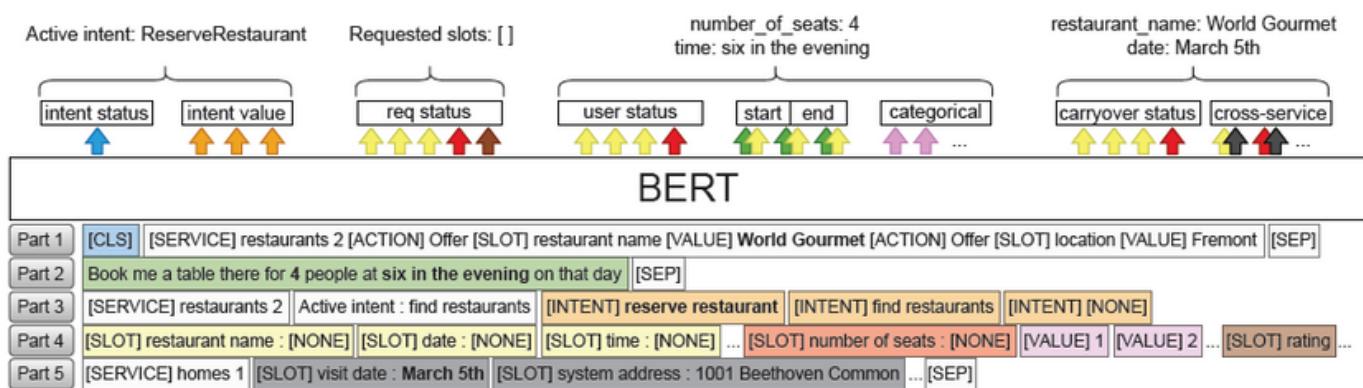
ARQUITECTURA DEL MODELO MULTITASK BERT

E.Contact

- **Ejemplo Práctico:**

- Servicio de Restaurantes (Restaurants 2): El modelo procesa un diálogo donde el usuario cambia de la intención de "Encontrar Restaurantes" (FindRestaurants) a "Reservar Restaurante" (ReserveRestaurant).
- Transferencia de Espacios en el Sistema: El modelo reconoce y acepta el valor proporcionado por el sistema ("World Gourmet") para el nombre del restaurante, exemplificando el mecanismo de transferencia de espacios.
- Entrada del Usuario: El usuario proporciona valores para espacios no categóricos (como la hora) y categóricos (como el número de asientos).
- Transferencia de Espacios entre Servicios: Se mantiene la información relevante, como la fecha de una reserva, incluso si no se menciona explícitamente en el último turno, utilizando información de diálogos anteriores.

Fig. 3 Modelo BERT multitarea propuesto



E.Contact

La estructura de los datos en el contexto del sistema de diálogo basado en el modelo BERT multitarea se organiza meticulosamente para evaluar la capacidad del modelo para rastrear el estado de los diálogos. La configuración experimental utiliza el conjunto de datos SGD, que proporciona un conjunto diverso de diálogos en varios dominios y servicios. A continuación, se detalla esta estructura:

Conjunto de Datos (Dataset):

- **SGD (Schema-Guided Dialogue):**

Contiene 21,106 diálogos que abarcan 20 dominios y 45 servicios, con divisiones estándar de entrenamiento, desarrollo y prueba. El conjunto de pruebas incluye diálogos de dominio único y múltiple, con un alto porcentaje de turnos que presentan servicios no vistos en el conjunto de entrenamiento.

- **Adquisición de Etiquetas (Label Acquisition):**

Para adquirir etiquetas para el estado del usuario y la transferencia de información (carryover), se analizan las acciones del usuario y se buscan en turnos y estados de diálogo anteriores para identificar la fuente de la información para el espacio.

- **Configuración de Entrenamiento (Training Setup):**

Se utiliza la implementación de BERT en minúsculas de Huggingface. Se emplea un tamaño de lote de 16 y una tasa de abandono de 0.3 para las cabezas de clasificación, con el optimizador AdamW y un calentamiento lineal del 10% de los pasos de entrenamiento.

- **Carga de Esquemas y Diálogos (Loading Schemas and Dialogues):**

Los esquemas y diálogos se cargan desde el disco utilizando un script de Python que detalla las funciones y clases necesarias para procesar y organizar los datos.

- **Estructura de Datos (Data Structure):**

Se define una serie de clases para representar diferentes aspectos del diálogo, como Service, Intent, Slot, State, Action, Frame, Turn, y Dialogue, cada una con atributos específicos que permiten una representación detallada y estructurada de los componentes de los diálogos.

ESTRUCTURA DE DATOS

E.Contact

- **Proceso de Carga y Conversión (Loading and Conversion Process):**

- Las funciones de conversión y carga, como **convert_task_name** y **get_file_range**, manejan la selección y carga de los archivos JSON del diálogo, basándose en la configuración del conjunto de datos y la tarea específica.
- La función **load_schema_from_disk** carga la estructura de los esquemas de servicios, definiendo los slots, intenciones y descripciones asociadas a cada servicio.
- La función **load_dialogues_from_disk** carga los diálogos y los procesa, extrayendo y organizando la información de cada turno y marco de diálogo.

- **Utilización de Datos Durante la Evaluación (Data Usage During Evaluation):**

- Durante la evaluación, el modelo utiliza los estados de diálogo previos de la verdad de campo para entrenamiento y los estados predichos previamente durante la evaluación.

- **Resultados Esperados (Expected Outcomes):**

- Se espera que el modelo, con esta estructura de datos y configuración de entrenamiento, exhiba un rendimiento superior en las métricas definidas como la Precisión de la Meta Conjunta (JGA), la Precisión de la Meta Promedio (Avg GA), la Precisión de la Intención y el F1 de los Espacios Solicitados.

Fig. 4 Estructura de los datos

```

    "city": [
      "Milpitas",
      "milpitas"
    ],
    "cuisine": [
      "Take-out"
    ],
    "restaurant_name": [
      "Olive garden Italian Restaurant"
    ]
  }
],
"speaker": "USER",
"utterance": "Yes, it seems good for me"
},
{
  "frames": [
    {
      "actions": [
        {
          "act": "OFFER_INTENT",
          "canonical_values": [
            "ReserveRestaurant"
          ],
          "slot": "intent",
          "values": [
            "ReserveRestaurant"
          ]
        }
      ],
      "service": "Restaurants_1",
      "slots": []
    }
  ],
  "speaker": "SYSTEM",
  "utterance": "shall i reserve a table here for you?"
},
{
  "frames": [
    {
      "actions": [
        {
          "act": "AFFIRM_INTENT",
          "canonical_values": [],
          "slot": "",
          "values": []
        }
      ]
    }
  ]
}
  
```

E.Contact

Para desplegar dependencias y el modelo utilizando se usarán los archivos proporcionados en el proyecto, se debe seguir una secuencia estructurada que garantice la correcta inicialización y funcionamiento del sistema. A continuación se detalla cada paso y su propósito en el despliegue:

Despliegue de Dependencias:

Antes de ejecutar cualquier script, es fundamental asegurarse de que todas las dependencias necesarias están instaladas. Esto suele realizarse mediante un archivo **requirements.txt** que lista todas las bibliotecas de Python necesarias. Se instalarían con un comando como:

```
bash
pip install -r requirements.txt
```

Inicialización del Módulo de Datos (datos):

1. `__init__.py`: Este archivo permite que Python trate los directorios que contienen el archivo como paquetes. Ayuda a evitar conflictos de nombres y simplifica las importaciones.
2. `classes.py`: Define las clases necesarias para manejar los diálogos, estados de diálogo, intenciones y más, estructurando la información de manera orientada a objetos.
3. `dialogue_processing.py`: Contiene las funciones para procesar los diálogos, como cargar datos, preprocessar la entrada y preparar los datos para el entrenamiento o la evaluación.
4. `eda.py` (Exploratory Data Analysis): Este script podría utilizarse para realizar un análisis exploratorio de los datos, ayudando a entender las características del conjunto de datos SGD.
5. `feature_extraction.py`: Extrae características relevantes de los diálogos que serán utilizadas por el modelo para realizar predicciones.
6. `input_sequence.py`: Prepara las secuencias de entrada que alimentarán al modelo, posiblemente codificando los diálogos y las historias de los diálogos.
7. `pytorch_dataset.py` y `sgd_dataset.py`: Estos archivos deben contener las definiciones de los conjuntos de datos específicos para PyTorch, permitiendo cargar y manipular el conjunto de datos SGD durante el entrenamiento y la evaluación.
8. `utils.py`: Proporciona utilidades y funciones auxiliares que apoyan las operaciones de los otros scripts, como la limpieza de texto, la división de casos camel y la conversión de nombres de tareas.

Inicialización del Módulo de DST Guiado por Esquemas (esquema_guiado_dst):

1. `__init__.py`: Al igual que en el módulo de datos, este archivo convierte el directorio en un paquete Python y facilita las importaciones. Programa para inicialización global (solo carga globalmente módulos).
2. `configuración.py`: Contiene las configuraciones del sistema, como rutas de acceso a archivos, hiperparámetros del modelo y cualquier otra configuración global necesaria para la ejecución de los scripts.
3. `dst.py`: Es probable que este archivo contenga la lógica principal para el seguimiento del estado del diálogo, interactuando con los modelos para predecir intenciones, espacios solicitados y realizar el llenado de espacios.
4. `modelos.py`: Define la arquitectura del modelo BERT multitarea, incluyendo las cabezas de clasificación y cualquier otra componente específica del modelo.
5. `tren.py`: Encargado de la lógica de entrenamiento, desde la inicialización del modelo, pasando por el bucle de entrenamiento, hasta la evaluación y el guardado de los pesos del modelo.

Despliegue del Modelo:

Una vez que el entorno está configurado y las dependencias están en su lugar, el proceso de entrenamiento puede comenzar ejecutando el script `tren.py`. Este script manejará la carga de los datos, la inicialización del modelo, el entrenamiento iterativo y la evaluación del rendimiento. La configuración del entrenamiento se detalla en `configuración.py`, y el modelo y los datos procesados se definen y manipulan a través de los scripts en los paquetes datos y `esquema_guulado_dst`.

Fig. 5 Despliegue de Dependencias

The screenshot shows a Jupyter Notebook interface with the following details:

- File Menu:** Archivo, Editar, Ver, Insertar, Entorno de ejecución, Herramientas, Ayuda.
- Toolbar:** Comentar, Compartir, Settings, G.
- Code Cell:**

```
(cl_intent_value): ClassificationHead(
    (linear1): Linear(in_features=768, out_features=128, bias=True)
    (dropout): Dropout(p=0.3, inplace=False)
    (linear2): Linear(in_features=128, out_features=2, bias=True)
)
(cl_usr_status): ClassificationHead(
    (linear1): Linear(in_features=768, out_features=16, bias=True)
    (dropout): Dropout(p=0.3, inplace=False)
    (linear2): Linear(in_features=22, out_features=3, bias=True)
)
(cl_copy_status): ClassificationHead(
    (linear1): Linear(in_features=768, out_features=16, bias=True)
    (dropout): Dropout(p=0.3, inplace=False)
    (linear2): Linear(in_features=22, out_features=4, bias=True)
)
(cl_req_status): ClassificationHead(
    (linear1): Linear(in_features=768, out_features=16, bias=True)
    (dropout): Dropout(p=0.3, inplace=False)
    (linear2): Linear(in_features=22, out_features=2, bias=True)
)
)
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:306: FutureWarning: This implementation of Ada
warnings.warn(
== TRAINING ==
Script called with args:
Namespace(load_checkpoint_path=None, num_epochs=5, num_total_epochs=5, num_steps_logging=4000)
== TRAINING ==
Epoch: 1
 4% 404/10987 [06:47<3:08:15, 1.07s/it]
```
- File Explorer:** Shows files like LICENSE.txt, README.md, dstc8.md, schema_guided_overview.md, f1gs, mtsgdst, data, schema_guided_dst, __init__.py, config.py, dst.py, models.py, train.py, multitask-schema-guided-overview.md, pickles, best.pt, and latest.pt.
- System Status:** RAM: 40.02 GB disponibles.
- Execution Status:** Ejecutando (10 min, 4 s) <cell line: 1> system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()

Configuración de Hiperparámetros:

Programa config.py

Los Hiperparámetros más importantes son:

Parametros:

1-Maximum sequence length for input text

MAX_SEQ_LEN = 512

2-Drop Dropout

WORD_DROPOUT=0.1

3-Schema_augment_Prob

SCHEMA_AUGMENT_PROB=0.1

4-Batch Size

BATCH_SIZE=16

5-Learning Rate

LEARNING_RATE=2e-5

6-MAX_INTENTS = 5

7-MAX_CAT_VALUES=11

8-MAX_SLOTS=17

9-MAX_SLOTS_OTHER_SERVICE=40

10-MAX_VALUES_PER_SERVICE=23

11-NUM_BIN_FEATS=6

Para Embeddings

Para Entrenamiento

Para Setting del Modelo

Flags:

1-Use Service History = TRUE

2-Use Dialogue History = TRUE

3-Use Full Slot Descriptions = TRUE

4-Use Natural Sys Utr = TRUE

(Utr: Utterance)

Se refiere a capacidades
a habilitar del modelo

```

config.py
1 import torch
2
3
4 MODEL_NAME = 'bert-base-uncased'
5 DEVICE = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
6 TASK_NAME = 'all'
7 MAX_SEQ_LEN = 512
8 USE_SERVICE_HISTORY = True
9 USE_DIALOGUE_HISTORY = True
10 USE_FULL_SLOT_DESCRIPTIONS = False
11 WORD_DROPOUT = 0.1
12 SCHEMA_AUGMENT_PROB = 0.1
13 USE_NATURAL_SYS_UTR = False
14
15 BATCH_SIZE = 16
16 LEARNING_RATE = 2e-5
17 LR_SCHEDULER = 'constant'
18 DROPOUT = 0.3
19 USE_BIN_FEATS = 1
20
21 DATASET_PATH = 'dstc8-schema-guided-dialogue'
22
23 MAX_INTENTS = 5
24 MAX_CAT_VALUES = 11
25 MAX_SLOTS = 17
26 MAX_SLOTS_OTHER_SERVICE = 40
27 MAX_VALUES_PER_SERVICE = 23
28 NUM_BIN_FEATS = 6
29
  
```

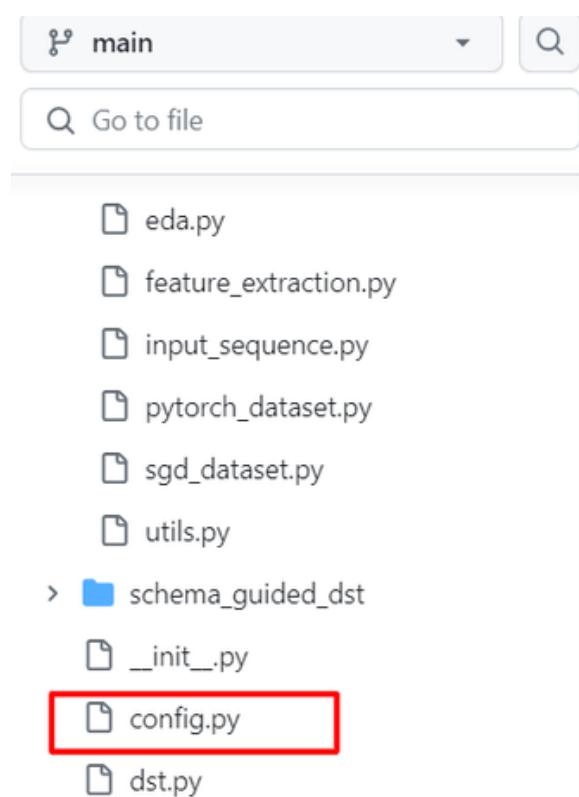
Paths:

1-DATASET_PATH

De donde toma la información del dataset para el entrenamiento y test

Conclusión :

Esta configuración de hiperparámetros muestra un enfoque equilibrado y bien pensado para entrenar un modelo de DST robusto y adaptable. Resalta la importancia de considerar tanto la estructura del modelo como las características de los datos al diseñar un sistema de diálogo eficiente. La configuración refleja un compromiso entre precisión, generalización y eficiencia computacional, preparando el terreno para un sistema de DST capaz de abordar de manera efectiva una amplia gama de situaciones y contextos de diálogo.



Configuración de Hiperparámetros:

Programa dst.py

- Carga del Modelo Entrenado (get_trained_model):
 - Inicializa y carga el estado del modelo MultitaskModel desde un checkpoint previamente entrenado, ajustando el modelo a la configuración del dispositivo (GPU o CPU).
- Procesamiento de Diálogos (process_dialogues):
 - Carga y procesa los diálogos del conjunto de datos SGD utilizando el modelo entrenado.
 - Genera predicciones para cada turno del diálogo, como intenciones, estados de usuario y valores de espacios.
- Predicciones de Intención y Espacios (Ej.: intent_prediction, slot_predictions):
 - Realiza predicciones específicas de intención y espacios basándose en los outputs del modelo.
- Evaluación del Modelo (evaluate):
 - Utiliza métricas específicas para evaluar la precisión y efectividad del modelo en el seguimiento de diálogos.
- Función Principal (main):
 - Coordina el flujo del proceso, desde el procesamiento de diálogos hasta la evaluación final del modelo.

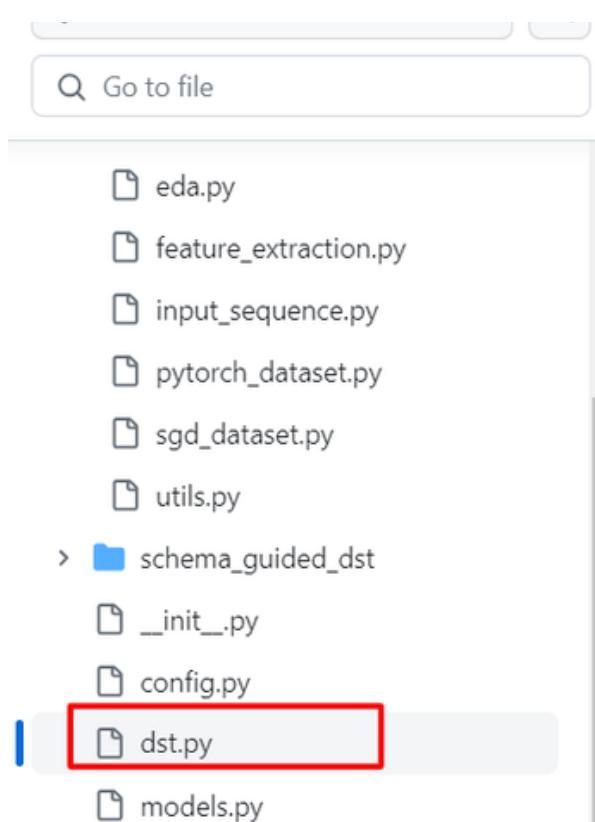
```

159 def process_dialogues(dataset_split, task_name, checkpoint_path):
160     model = get_trained_model(checkpoint_path)
161     with open(f'pickles/dataset_{dataset_split}_{task_name}_eval.pkl', 'rb') as f:
162         val_dataset = pickle.load(f)
163     dataloader = DataLoader(val_dataset, batch_size=1, collate_fn=collate_fn)
164
165     all_predictions = {}
166     with torch.no_grad():
167         for x, _ in dataloader:
168             x = utils.move_to_gpu(x)
169
170             dial_id = x['dial_id'][0]
171             turn_id = x['turn_id'][0].item()
172             service_name = x['service_name'][0]
173             service = val_dataset.schema.services[service_name]
174
175             logits = {k: v[0] for k, v in model(x).items()}
176             model_outputs = get_model_outputs(logits)
177
178             hist = val_dataset.histories[(dial_id, turn_id)]
179             hist_pred = val_dataset.running_hist_pred[dial_id]
180
181             pred_intent = intent_prediction(service, hist_pred, model_outputs)
182             other_service_info = [(v[0][0], v[1][0]) for v in x['other_service_info']]
183             req_slots, slot_status, slot_value = slot_predictions(
184                 service, hist, hist_pred, model_outputs, other_service_info, x['utter'][0])
185
186             updated_sv_pairs = {k: [v] for k, v in slot_value.items() if v}
187             dialogue_processing.get_new_service_hist_after_user_turn(
188                 hist_pred.services[service_name], pred_intent, updated_sv_pairs)
189
190             all_predictions[(dial_id, str(turn_id).zfill(2), service_name)] = {
191                 'active_intent': pred_intent,
192                 'req_slots': req_slots,
193                 'slot_status': slot_status,
194                 'slot_value': slot_value
195             }
196
197     return all_predictions

```

Conclusión :

El script dst.py demuestra un enfoque sofisticado y bien estructurado para implementar un modelo de DST basado en BERT multitarea. La combinación de carga eficiente del modelo, procesamiento detallado de diálogos y evaluación rigurosa subraya la capacidad del sistema para manejar de manera efectiva las complejidades inherentes al seguimiento de estados en conversaciones. La inclusión de funciones especializadas para predecir intenciones, estados de usuario y valores de espacios refleja una comprensión profunda de las necesidades de un sistema de DST efectivo. La capacidad del modelo para generalizar a través de múltiples dominios, como se evidencia en el conjunto de datos SGD, lo posiciona como una herramienta poderosa en la arena de sistemas de diálogo orientados a tareas. Este script, con su enfoque metódico y su integración de componentes de DST, es un testimonio de la innovación técnica y la eficiencia computacional en la implementación de sistemas de diálogo avanzados.



Configuración de Hiperparámetros:

Programa models.py

Define a las dos clases principales del modelo, las cuales son:

1-Clase **ClassificationHead**: Clase que contiene la clasificación de la cabecera

2-Clase **Multitaskmodel**: Clase principal que contiene el modelo Multitask propuesto.

-Llama a la clase **ClassificationHead**

```
class ClassificationHead(nn.Module):
    def __init__(self, dropout, input_dim, hidden_dim, output_dim, num_bin_feats=0):
        super(ClassificationHead, self).__init__()
        self.num_bin_feats = num_bin_feats

        self.linear1 = nn.Linear(input_dim, hidden_dim)
        self.dropout = nn.Dropout(dropout)
        self.linear2 = nn.Linear(hidden_dim + num_bin_feats, output_dim)

    def forward(self, x):
        out = self.linear1(x['cls'])
        out = torch.relu(out)
        out = self.dropout(out)
        if self.num_bin_feats > 0:
            out = torch.cat([out, x['binary']], dim=-1)
        logits = self.linear2(out)
        return logits
```

```
class MultitaskModel(nn.Module):
    def __init__(self):
        super(MultitaskModel, self).__init__()
        self.bert = utils.get_bert_with_tokens()
        config.DROPOUT
        self.dropout1 = nn.Dropout(config.DROPOUT)

        self.cl_value = ClassificationHead(config.DROPOUT, self.bert.config.hidden_size, self.bert.config.hidden_size, self.bert.config.hidden_size)
        self.cl_start = ClassificationHead(config.DROPOUT, 2 * self.bert.config.hidden_size, self.bert.config.hidden_size, self.bert.config.hidden_size)
        self.cl_end = ClassificationHead(config.DROPOUT, 2 * self.bert.config.hidden_size, self.bert.config.hidden_size, self.bert.config.hidden_size)
        self.cl_cross = ClassificationHead(config.DROPOUT, 2 * self.bert.config.hidden_size, self.bert.config.hidden_size, self.bert.config.hidden_size)
        self.cl_intent_status = ClassificationHead(config.DROPOUT, self.bert.config.hidden_size, 128, 2)
        self.cl_intent_value = ClassificationHead(config.DROPOUT, self.bert.config.hidden_size, 128, 2)
        num_bin_feats = config.NUM_BIN_FEATS if config.US_BIN_FEATS else 0
        self.cl_usr_status = ClassificationHead(config.DROPOUT, self.bert.config.hidden_size, 16, 3, num_bin_feats)
        self.cl_copy_status = ClassificationHead(config.DROPOUT, self.bert.config.hidden_size, 16, 4, num_bin_feats)
        self.cl_req_status = ClassificationHead(config.DROPOUT, self.bert.config.hidden_size, 16, 2, num_bin_feats)

    def forward(self, x):
        logits = {}
        # y: (batch_size, seq_len, self.bert.config.hidden_size)
        # x['slot_positions']: (batch_size, MAX_SLOTS)
        # x['value_positions']: (batch_size, MAX_VALUES_PER_SERVICE)
        # slot_embeddings: (batch_size, MAX_SLOTS, self.bert.config.hidden_size)
        # value_embeddings: (batch_size, MAX_VALUES_PER_SERVICE, self.bert.config.hidden_size)
        y = self.bert(**x['tokenized']).last_hidden_state
        y = self.dropout1(y)

        slot_embeddings = torch.gather(y, 1, x['slot_positions'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))
        intent_embeddings = torch.gather(y, 1, x['intent_positions'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))
        slot_embeddings_other = torch.gather(y, 1, x['slot_positions_other_service'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))
        value_embeddings = torch.gather(y, 1, x['value_positions'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))

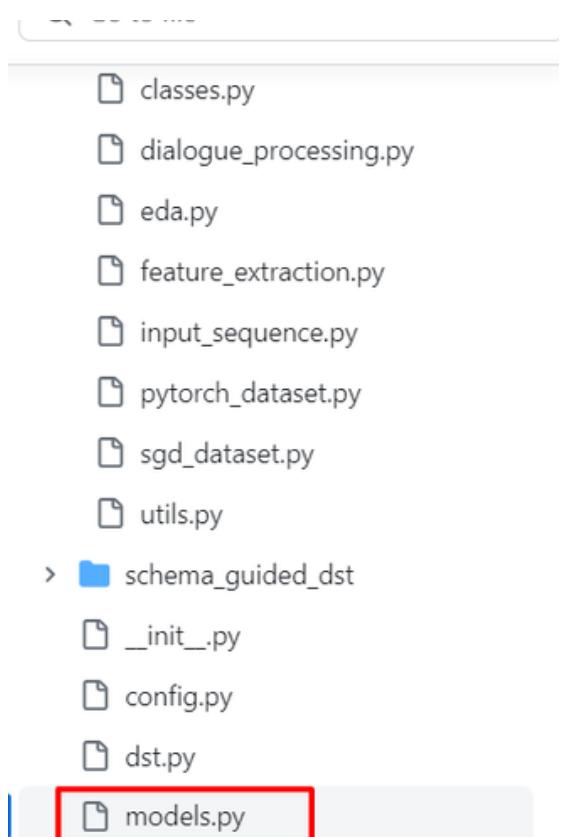
        logits['intent_status'] = self.cl_intent_status({'cls': y[:, 0, :]})
        logits['intent_values'] = self.cl_intent_value({'cls': intent_embeddings})

        slot_feats = {'cls': slot_embeddings, 'binary': x['binary']}
        logits['usr_status'] = self.cl_usr_status(slot_feats)
        logits['copy_status'] = self.cl_copy_status(slot_feats)
        logits['req_status'] = self.cl_req_status(slot_feats)
        logits['values'] = self.cl_value({'cls': value_embeddings})

        slot_embeddings_repeated = slot_embeddings.unsqueeze(2).repeat(1, 1, y.shape[1], 1)
```

Conclusión :

El archivo `models.py` refleja una implementación sofisticada y bien pensada para un sistema de DST basado en BERT multitarea. La estructura modular del modelo y el uso de cabezas de clasificación dedicadas para cada tarea de DST ilustran un enfoque detallado y especializado hacia el procesamiento del lenguaje natural en el contexto de diálogos. La integración efectiva de BERT en este marco subraya la capacidad del modelo para captar con precisión y eficacia el contexto y las complejidades inherentes a los diálogos naturales. Este enfoque no solo facilita una mayor precisión en la predicción de estados de diálogo, sino que también ofrece una flexibilidad y escalabilidad notables para adaptarse a una variedad de escenarios de diálogo. En resumen, `models.py` es un componente crucial que subyace a la robustez y eficacia del sistema de DST propuesto, destacando la importancia de una arquitectura bien diseñada en la creación de sistemas de diálogo avanzados.



Configuración de Hiperparámetros:

Programa init_.py

2. Inicialización y Configuración (Función `__init__`):

- Importa y añade al path del sistema el directorio actual, lo que facilita el acceso a los módulos y scripts dentro del mismo directorio o subdirectorios.
- Carga los parámetros desde config.py, asegurando que todas las configuraciones y constantes necesarias estén disponibles para el modelo.
- Declara constantes y parámetros relacionados con la red neuronal, como las dimensiones de las capas y las características específicas del modelo.
- Inicializa componentes del modelo, probablemente incluyendo la creación de instancias de ClassificationHead de model.py, lo que indica que el modelo utilizará estas cabezas para tareas de clasificación específicas.

2. Función de Procesamiento (Función `forward`):

- La descripción sugiere que la función forward no está definida en `__init__.py`, sino más bien en otros archivos como model.py. Sin embargo, si estuviera presente, su función sería procesar los datos a través del modelo.

```
def forward(self, x):
    logits = {}
    # y: (batch_size, seq_len, self.bert.config.hidden_size)
    # x['slot_positions']: (batch_size, MAX_SLOTS)
    # x['value_positions']: (batch_size, MAX_VALUES_PER_SERVICE)
    # slot_embeddings: (batch_size, MAX_SLOTS, self.bert.config.hidden_size)
    # value_embeddings: (batch_size, MAX_VALUES_PER_SERVICE, self.bert.config.hidden_size)
    y = self.bert(**x['tokenized']).last_hidden_state
    y = self.dropout(y)

    slot_embeddings = torch.gather(
        y, 1, x['slot_positions'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))
    intent_embeddings = torch.gather(
        y, 1, x['intent_positions'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))
    slot_embeddings_other = torch.gather(
        y, 1, x['slot_positions_other_service'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))
    value_embeddings = torch.gather(
        y, 1, x['value_positions'].unsqueeze(-1).repeat(1, 1, self.bert.config.hidden_size))

    logits['intent_status'] = self.cl_intent_status({'cls': y[:, 0, :]})
    logits['intent_values'] = self.cl_intent_value({'cls': intent_embeddings})

    slot_feats = {'cls': slot_embeddings, 'binary': x['binary']}
    logits['usr_status'] = self.cl_usr_status(slot_feats)
    logits['copy_status'] = self.cl_copy_status(slot_feats)
    logits['req_status'] = self.cl_req_status(slot_feats)
    logits['values'] = self.cl_value({'cls': value_embeddings})

    slot_embeddings_repeated = slot_embeddings.unsqueeze(2).repeat(1, 1, y.shape[1], 1)
    tokens_repeated = y.unsqueeze(1).repeat(1, slot_embeddings_repeated.shape[1], 1, 1)
    slots_and_tokens = torch.cat([slot_embeddings_repeated, tokens_repeated], 3)

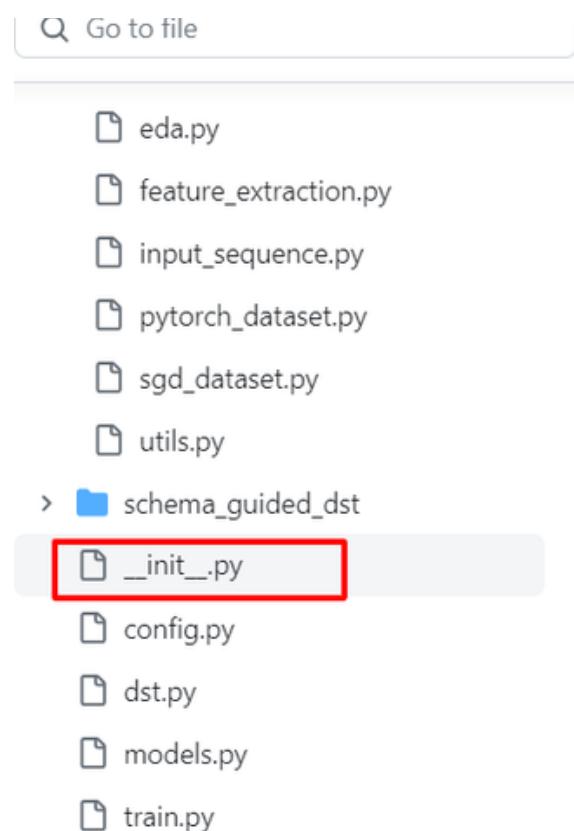
    logits['start'] = self.cl_start({'cls': slots_and_tokens}).squeeze(-1)
    logits['end'] = self.cl_end({'cls': slots_and_tokens}).squeeze(-1)

    usr_utr_mask = x['usr_utr_mask']
    usr_utr_mask[0] = 1
    usr_utr_mask = usr_utr_mask.unsqueeze(1)

    logits['start'] = torch.where(usr_utr_mask == 1, logits['start'], torch.Tensor([-1e9]).to(config.DEVICE))
    logits['end'] = torch.where(usr_utr_mask == 1, logits['end'], torch.Tensor([-1e9]).to(config.DEVICE))
```

Conclusión :

El archivo `__init__.py` juega un papel crítico en la estructuración y la configuración inicial del proyecto de DST. Al establecer las conexiones necesarias entre diferentes módulos y configurar los parámetros iniciales, este archivo asegura que el entorno de trabajo del modelo esté correctamente organizado y listo para su ejecución. La posible ausencia de una función `forward` en este archivo es típica, ya que las operaciones de procesamiento de datos suelen estar contenidas en módulos específicos del modelo, como `model.py`. En general, `__init__.py` es un componente esencial para la coherencia y eficiencia del proyecto, asegurando que todas las partes del modelo estén adecuadamente interconectadas y que las configuraciones sean coherentes a lo largo del proceso de desarrollo y entrenamiento del modelo de DST.



Configuración de Hiperparámetros:

Programa train.py

Define dos funciones principales:

1- Función de Inicialización `get_loss()`

Función que calcula las perdidas entre `y_pred` y `y_real`

2- Función `train()`

- Configura el **path resultados** donde dejará los resultados de la evaluación
- Setea la red neuronal previo al entrenamiento
- Setea el modelo (Llama al **modelo**)
- Ejecuta el entrenamiento
- Evalua el modelo (Llama a **evaluación**)
- Revisa las métricas resultantes
- Guarda los resultados en el **path de resultados**

Conclusión :

El script `train.py` representa un enfoque completo y sistemático para el entrenamiento de modelos complejos de DST. La meticulosa atención a la función de pérdida y el proceso detallado de entrenamiento y validación demuestran un entendimiento profundo de los requisitos para el entrenamiento efectivo de modelos de aprendizaje profundo. La integración de técnicas de optimización avanzadas, junto con la gestión cuidadosa de los checkpoints y la evaluación periódica, subraya un enfoque equilibrado que busca optimizar el rendimiento del modelo mientras se controla el sobreajuste. Este script es un ejemplo excelente de cómo manejar múltiples aspectos del entrenamiento de modelos en proyectos de procesamiento de lenguaje natural, proporcionando una base sólida para el desarrollo de sistemas de diálogo eficientes y precisos.

```

dst.main('dev', 'checkpoints/latest.pt')

with open('metric_file.json', 'r') as f:
    json_metric_file = json.load(f)
    val_acc = json_metric_file['ALL_SERVICES']['joint_goal_accuracy']
    val_unseen_acc = json_metric_file['UNSEEN_SERVICES']['joint_goal_accuracy']
    print(f'Validation accuracy: {val_acc}')
    print(f'Validation unseen accuracy: {val_unseen_acc}')

if val_acc > max_val_acc:
    max_val_acc = val_acc
    save_checkpoint('checkpoints/best.pt')
    save_checkpoint('checkpoints/latest.pt')

    train_loss = running_train_loss / len(train_dataloader)
    print(f'\ntrain loss: {train_loss}\n')

    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(range(1, len(val_losses) + 1), val_losses, label='validation')
    plt.legend()
    plt.savefig('figs/losses.png')

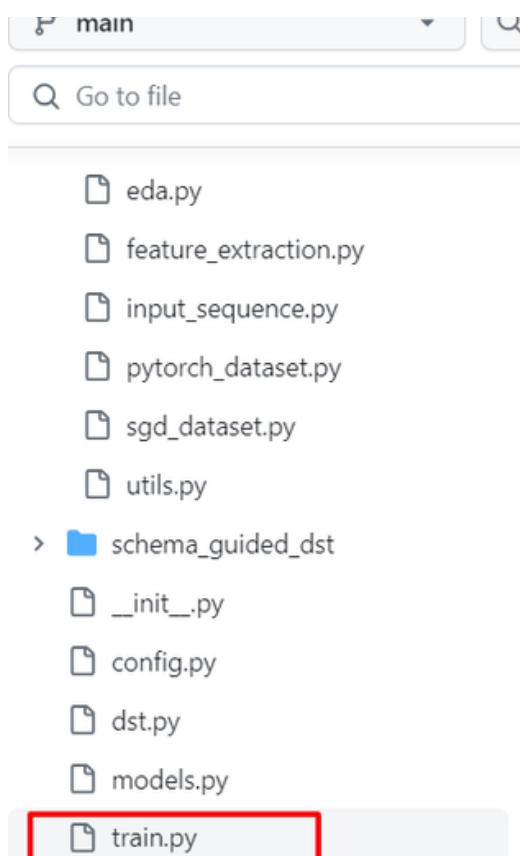
if __name__ == '__main__':
    utils.create_directories()
    parser = argparse.ArgumentParser(description='Train the model.')
    parser.add_argument('--load_checkpoint_path')
    parser.add_argument('--num_epochs', default=1, type=int)
    parser.add_argument('--num_total_epochs', default=3, type=int)
    parser.add_argument('--num_steps_logging', type=int)
    args = parser.parse_args()

    model = MultitaskModel()
    print(model)

    with open('pickles/dataset_train_{config.TASK_NAME}.pkl', 'rb') as f:
        train_dataset = pickle.load(f)
    with open('pickles/dataset_dev_{config.TASK_NAME}.pkl', 'rb') as f:
        val_dataset = pickle.load(f)

    train(model)

```



Revisión PruebasModelo

Multi-Task Bert Model para DST

Las métricas que se utilizará para la evaluación del modelo son las siguientes:

1.active_intent_accuracy: La precisión de la predicción de la intención activa. Mide con qué precisión el modelo predice la intención principal de la solicitud del usuario.

2.average_cat_accuracy: La precisión promedio para ranuras categóricas. Mide la precisión de la predicción para los slots cuyos valores son categóricos.

3.average_goal_accuracy : La precisión promedio de la predicción del objetivo. Evalúa cuánto se acerca el modelo a predecir correctamente los objetivos de los usuarios.

4.average_noncat_accuracy : La precisión promedio para slots no categóricas. Similar a average_cat_accuracy, pero para slots cuyos valores no son categóricos

5.joint_cat_accuracy : La precisión conjunta para slots categóricas. Mide la precisión global para las slots categóricas.

6.joint_goal_accuracy : La precisión conjunta para los objetivos. Indica la precisión global para la predicción de objetivos.

7.joint_noncat_accuracy: La precisión conjunta para slots no categóricas. Mide la precisión global para los slots no categóricas.

8.requested_slots_f1, requested_slots_precision, requested_slots_recall: Métricas relacionadas con la identificación de slots solicitadas. F1-score, precisión y recall para determinar si las slots solicitadas por el usuario se identificaron correctamente.

```
Alarm_1": {
    "active_intent_accuracy": 0.9432314410480349,
    "average_goal_accuracy": 0.8271103896103897,
    "average_noncat_accuracy": 0.8271103896103897,
    "joint_goal_accuracy": 0.8431441048034933,
    "joint_noncat_accuracy": 0.8431441048034933,
    "requested_slots_f1": 0.9912663755458515,
    "requested_slots_precision": 0.9912663755458515,
    "requested_slots_recall": 1.0
},
"Banks": {
    "active_intent_accuracy": 0.9435933147632312,
    "average_cat_accuracy": 0.8909620817270624,
    "average_goal_accuracy": 0.890341814443588,
    "average_noncat_accuracy": 0.9059865900383142,
    "joint_cat_accuracy": 0.865598887938719,
    "joint_goal_accuracy": 0.8316110027855154,
    "joint_noncat_accuracy": 0.9425668523676879,
    "requested_slots_f1": 0.9623955431754875,
    "requested_slots_precision": 0.9965181058495822,
    "requested_slots_recall": 0.9637883008356546
},
Banks_2": {
    "active_intent_accuracy": 0.9435933147632312,
    "average_cat_accuracy": 0.8909620817270624,
    "average_goal_accuracy": 0.890341814443588,
    "average_noncat_accuracy": 0.9059865900383142,
    "joint_cat_accuracy": 0.865598887938719,
    "joint_goal_accuracy": 0.8316110027855154,
    "joint_noncat_accuracy": 0.9425668523676879,
    "requested_slots_f1": 0.9623955431754875,
    "requested_slots_precision": 0.9965181058495822,
    "requested_slots_recall": 0.9637883008356546
},
```

Conjunto de Datos a Evaluar con las métricas antes descritas

Estas métricas se presentan para diferentes conjuntos de datos que son:

#ALL_SERVICES,
#SEEN_SERVICES,
#UNSEEN_SERVICES

y para servicios individuales

Alarm,
Banks,
Buses,
Events,
Flights,
Homes,
Hotels, entre otros.

Proporcionan una visión detallada del rendimiento del modelo en varios aspectos del procesamiento del lenguaje natural relacionado con el diálogo y la tarea específica que está abordando el modelo.

```
metric_file.json
1  {
2      "#ALL_SERVICES": {
3          "active_intent_accuracy": 0.9489588526287533,
4          "average_cat_accuracy": 0.8894785635902676,
5          "average_goal_accuracy": 0.8631229168504957,
6          "average_noncat_accuracy": 0.864777859512477,
7          "joint_cat_accuracy": 0.8694774982500927,
8          "joint_goal_accuracy": 0.6588142772585037,
9          "joint_noncat_accuracy": 0.7314695167293017,
10         "requested_slots_f1": 0.9792236179665715,
11         "requested_slots_precision": 0.9899668929196865,
12         "requested_slots_recall": 0.9835295471104805
13     },
14     "#SEEN_SERVICES": {
15         "active_intent_accuracy": 0.9432835820895522,
16         "average_cat_accuracy": 0.902397033505496,
17         "average_goal_accuracy": 0.8675905254527478,
18         "average_noncat_accuracy": 0.8677573592595321,
19         "joint_cat_accuracy": 0.896554275857762,
20         "joint_goal_accuracy": 0.662761506270692,
21         "joint_noncat_accuracy": 0.7233778840644505,
22         "requested_slots_f1": 0.9825192220714608,
23         "requested_slots_precision": 0.9885911352329263,
24         "requested_slots_recall": 0.9884667571234735
25     },
26     "#UNSEEN_SERVICES": {
27         "active_intent_accuracy": 0.9563376554644085,
28         "average_cat_accuracy": 0.8722516556291391,
29         "average_goal_accuracy": 0.8572200537546313,
30         "average_noncat_accuracy": 0.8605827301653903,
31         "joint_cat_accuracy": 0.8349569127013863,
32         "joint_goal_accuracy": 0.6536822180153479,
33         "joint_noncat_accuracy": 0.7419899952932875,
34         "requested_slots_f1": 0.9749387803105718,
35         "requested_slots_precision": 0.9917556084796095,
36         "requested_slots_recall": 0.9771103466525536
37     },
38 }
```

Entendimiento de los Servicios

Es importante entender los servicios evaluados y obtenidos en las métricas.

#ALL_SERVICES :

Resume las métricas agregadas para todos los servicios presentes en el conjunto de datos. En este contexto, se están evaluando métricas para el conjunto completo de servicios.

Es decir cuando observas métricas específicas de **#ALL_SERVICES**, estás viendo el rendimiento global del modelo en todos los servicios combinados.

Por ejemplo, **active_intent_accuracy**, **average_cat_accuracy**, **average_goal_accuracy**, etc., proporcionan estadísticas agregadas para todas las intenciones, slots categóricas y objetivos de todos los servicios.

Esta categoría es útil para obtener una visión general del rendimiento del modelo en la tarea de procesamiento de lenguaje natural basada en diálogos para todos los servicios, sin desglosar por servicio individualmente.

Es importante entender los servicios evaluados y obtenidos en las métricas.

#SEEN_SERVICES (Vistos en el entrenamiento):

Representa métricas calculadas específicamente para los servicios que ya han aparecido en el conjunto de datos. Cuando se refiere a "vistos" o "vistos previamente", se refiere a los servicios que están presentes en la partición de datos actual (por ejemplo, el conjunto de entrenamiento o el conjunto de validación).

Cuando observas las métricas dentro de **#SEEN_SERVICES**, estás viendo el rendimiento del modelo solo en los servicios que ha encontrado durante el entrenamiento o la evaluación anterior. Estas métricas son útiles para evaluar cómo se está desempeñando el modelo en servicios con los que ya ha tenido alguna interacción.

En resumen, **#SEEN_SERVICES** proporciona estadísticas específicas para los servicios que ya han sido encontrados en el conjunto de datos actual.

Es importante entender los servicios evaluados y obtenidos en las métricas.

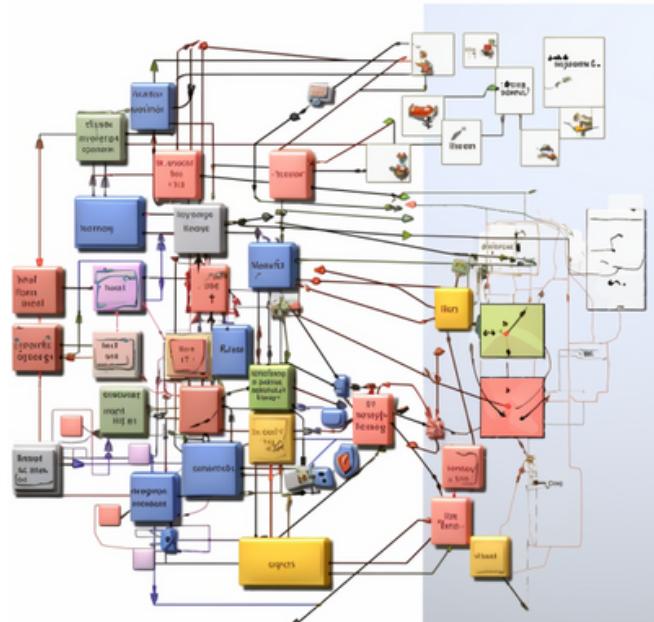
#UNSEEN_SERVICES (Vistos en el Test):

Representa métricas calculadas específicamente para los servicios que son nuevos o que no han aparecido previamente en el conjunto de datos. Cuando te refieres a "no vistos" o "no vistos previamente", se refiere a los servicios que no estaban presentes durante el entrenamiento o la evaluación anterior.

Cuando observas las métricas dentro de **#UNSEEN_SERVICES**, estás viendo el rendimiento del modelo solo en servicios que son desconocidos hasta ese momento. Estas métricas son útiles para evaluar cómo se generaliza el modelo a servicios que no ha encontrado previamente.

En resumen, **#UNSEEN_SERVICES** proporciona estadísticas específicas para los servicios que son nuevos en el conjunto de datos actual y que no estaban presentes durante el entrenamiento o la evaluación anteriores.

Fig. 6 Representación de los servicios Multitask



Relacionamiento de los Servicios en Asistente Virtual

Ejemplo: Un modelo de procesamiento del lenguaje natural diseñado para realizar tareas de asistente virtual. Supongamos que el modelo ha sido entrenado en un conjunto de datos que incluye servicios de reservas de vuelos, reservas de hoteles y reservas de restaurantes. Durante este entrenamiento, estos tres servicios específicos ("Flights", "Hotels" y "Restaurants") son considerados como **#SEEN_SERVICES**.

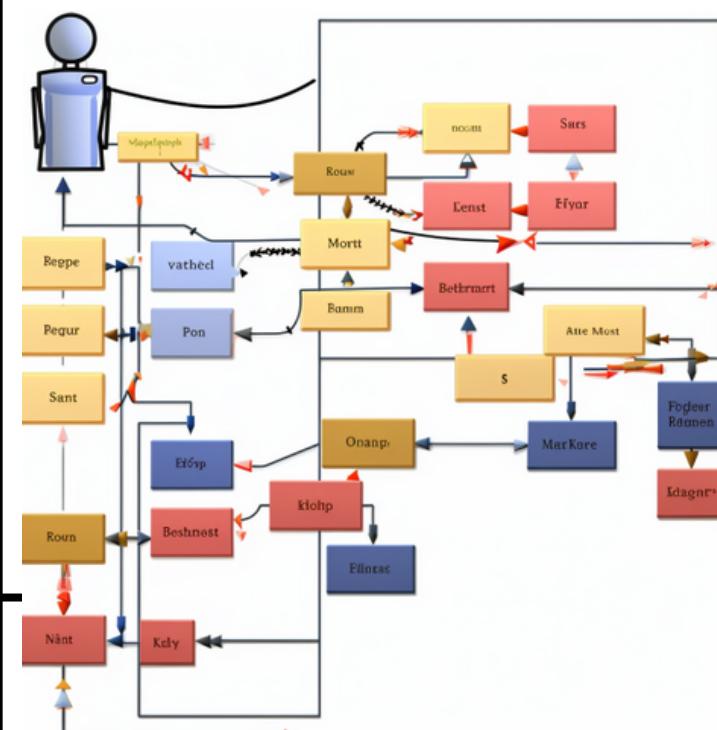
Ahora, cuando evaluamos el modelo en un conjunto de datos de prueba que incluye un nuevo servicio, por ejemplo, un servicio de compra de boletos para eventos ("Events"), este servicio específico se consideraría como **#UNSEEN_SERVICES**. El modelo no ha tenido conocimiento previo de este servicio durante el entrenamiento.

Esto nos permite entender cómo se desempeña el modelo en servicios que ya ha encontrado (vistos durante el entrenamiento) en comparación con servicios completamente nuevos que no estaban presentes durante el entrenamiento.

Estas métricas proporcionan información detallada sobre el rendimiento del modelo para cada servicio específico, incluyendo la precisión de la intención activa, la precisión promedio para categorías, objetivos y atributos no categóricos, y métricas relacionadas con la solicitud de ranuras.

- Para el servicio "Banks":
 - `active_intent_accuracy`: 94.36%
 - `average_cat_accuracy`: 89.09%
 - `average_goal_accuracy`: 89.03%
 - `average_noncat_accuracy`: 90.60%
 - `joint_cat_accuracy`: 86.56%
 - `joint_goal_accuracy`: 83.16%
 - `joint_noncat_accuracy`: 94.26%
 - `requested_slots_f1`: 96.24%
 - `requested_slots_precision`: 99.65%
 - `requested_slots_recall`: 96.38%

Fig. 7 Representación relacionado a los servicios de asistente virtual en DST



Determinando si el Modelo Multitask es Bueno para lo Buscado

Hay que considerar 3 aspectos importantes, que son:

- **Aspecto General.**

Accuracy general (joint_goal_accuracy):

Esta métrica indica la precisión general del modelo en predecir la intención, categorías, objetivos y atributos no categóricos en todas las interacciones del conjunto de datos. Un valor más alto es mejor. Sin embargo, es importante considerar el equilibrio entre estas submétricas, ya que un alto rendimiento en una área específica podría compensar un rendimiento más bajo en otra.

- **Solicitudes de Slots**

Solicitudes de slots (requested_slots_f1, requested_slots_precision, requested_slots_recall):

Estas métricas evalúan la capacidad del modelo para entender y manejar las solicitudes de slots. Un buen modelo debería tener un equilibrio entre precisión, recall y F1-score para estas métricas.

- **Aspecto Específicos**

Métricas específicas del servicio:

Puedes evaluar el rendimiento del modelo en servicios individuales. Si el sistema está destinado a ser especializado en ciertos dominios (por ejemplo, "Banks" o "Homes"), asegúrate de que el modelo tenga un rendimiento sólido en esos servicios específicos.

Resultado Final EPOCH 3

Hay que considerar 3 aspectos importantes, que son:

Aspecto General	Solicitudes de Slots
<pre>metric_file.json</pre> <pre>{ "#ALL_SERVICES": { "active_intent_accuracy": 0.9489588526287533, "average_cat_accuracy": 0.8894785635902676, "average_goal_accuracy": 0.8631229168504957, "average_noncat_accuracy": 0.8647977859512477, "joint_cat_accuracy": 0.8694774982500927, "joint_goal_accuracy": 0.6588142772585037, "joint_noncat_accuracy": 0.7314695167293017, "requested_slots_f1": 0.9792236179665715, "requested_slots_precision": 0.9899668929196865, "requested_slots_recall": 0.9835295471104805 }, }</pre>	
joint_goal_accuracy: 0.6588	request_slots_f1: 0.97922 requested_slots_precision: 0.9899 requested_slots_recall: 0.9835
Bueno!!! (Se debe a otros aspectos específicos)	Muy bueno!!!

Aspecto Específicos

- Para el servicio "Banks":

- `active_intent_accuracy`: 94.36%
- `average_cat_accuracy`: 89.09%
- `average_goal_accuracy`: 89.03%
- `average_noncat_accuracy`: 90.60%
- `joint_cat_accuracy`: 86.56%
- `joint_goal_accuracy`: 83.16%
- `joint_noncat_accuracy`: 94.26%
- `requested_slots_f1`: 96.24%
- `requested_slots_precision`: 99.65%
- `requested_slots_recall`: 96.38%

Muy bueno!!!

Revisión Pruebas Finales

Las métricas que se utilizará para la evaluación del modelo son las siguientes:

1.active_intent_accuracy: La precisión de la predicción de la intención activa. Mide con qué precisión el modelo predice la intención principal de la solicitud del usuario.

2.average_cat_accuracy: La precisión promedio para ranuras categóricas. Mide la precisión de la predicción para los slots cuyos valores son categóricos.

3.average_goal_accuracy :La precisión promedio de la predicción del objetivo. Evalúa cuánto se acerca el modelo a predecir correctamente los objetivos de los usuarios.

4.average_noncat_accuracy :La precisión promedio para slots no categóricas. Similar a average_cat_accuracy, pero para slots cuyos valores no son categóricos.

5.joint_cat_accuracy: La precisión conjunta para slots categóricas. Mide la precisión global para las slots categóricas.

6.joint_goal_accuracy : La precisión conjunta para los objetivos. Indica la precisión global para la predicción de objetivos.

7.joint_noncat_accuracy: La precisión conjunta para slots no categóricas. Mide la precisión global para los slots no categóricas.

8.requested_slots_f1, requested_slots_precision, requested_slots_recall:

Métricas relacionadas con la identificación de slots solicitadas. F1-score, precisión y recall para determinar si las slots solicitadas por el usuario se identificaron correctamente.

Hiperparámetros para Iteración 1

De los Hiperparámetros configurados, se ha revisado y se propone modificar lo siguiente:

Se toma y se modifica este parámetro:

Bath_Size: Se intenta aumentar el batch_size para agilizar el entrenamiento (revisar sobreajuste)

EPOCH: 3

Estamos parámetros se descartan porque están dentro del rango:

Learning_rate: Se propone ajustar entre algún valor entre 1e-5 a 1e-3.

Dropout: Se propone ajustar entre algún valor entre 0.1 y 0.5

```
MODEL_NAME = 'bert-base-uncased'
DEVICE = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
TASK_NAME = 'all'
MAX_SEQ_LEN = 512
USE_SERVICE_HISTORY = True
USE_DIALOGUE_HISTORY = True
USE_FULL_SLOT_DESCRIPTIONS = False
WORD_DROPOUT = 0.1
SCHEMA_AUGMENT_PROB = 0.1
USE_NATURAL_SYS_UTTR = False

BATCH_SIZE = 16
LEARNING_RATE = 2e-5
LR_SCHEDULER = 'constant'
DROPOUT = 0.3
USE_BIN_FEATS = 1

DATASET_PATH = 'dstc8-schema-guided-dialogue'

MAX_INTENTS = 5
MAX_CAT_VALUES = 11
MAX_SLOTS = 17
MAX_SLOTS_OTHER_SERVICE = 40
MAX_VALUES_PER_SERVICE = 23
NUM_BIN_FEATS = 6
```

Resultados Finales Iteración 1: EPOCH 3 , batch size 16, 120 Archivos

Aspecto General	Solicitudes de Slots
<code>metric_file.json</code>	
{ "#ALL_SERVICES": { "active_intent_accuracy": 0.9489588526287533, "average_cat_accuracy": 0.8894785635902676, "average_goal_accuracy": 0.8631229168504957, "average_noncat_accuracy": 0.8647977859512477, "joint_cat_accuracy": 0.8694774982500927, "joint_goal_accuracy": 0.6588142772585037, "joint_noncat_accuracy": 0.7314695167293017, "requested_slots_f1": 0.9792236179665715, "requested_slots_precision": 0.9899668929196865, "requested_slots_recall": 0.9835295471104885 },	
joint_goal_accuracy: 0.6588	request_slots_f1: 0.97922 requested_slots_precision: 0.9899 requested_slots_recall: 0.9835
Bueno!!! (Se debe a otros aspectos específicos)	Muy bueno!!!

Aspecto Específicos

- Para el servicio "Banks":
 - active_intent_accuracy: 94.36%
 - average_cat_accuracy: 89.09%
 - average_goal_accuracy: 89.03%
 - average_noncat_accuracy: 90.60%
 - joint_cat_accuracy: 86.56%
 - joint_goal_accuracy: 83.16%
 - joint_noncat_accuracy: 94.26%
 - requested_slots_f1: 96.24%
 - requested_slots_precision: 99.65%
 - requested_slots_recall: 96.38%

Muy bueno!!!

Hiperparámetros para iteración 2, 3 y 4

De los Hiperparámetros configurados, se ha revisado y se propone modificar lo siguiente. Se toma y se modifica este parámetro:

Bath_Size: Se intenta aumentar el batch_size para agilizar el entrenamiento (revisar sobreajuste)

EPOCH: 3, 4 y 5

Estamos parámetros se descartan porque están dentro del rango:

Learning_rate: Se propone ajustar entre algún valor entre 1e-5 a 1e-3.

Dropout: Se propone ajustar entre algún valor entre 0.1 y 0.5

```

MODEL_NAME = 'bert-base-uncased'
DEVICE = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
TASK_NAME = 'all'
MAX_SEQ_LEN = 512
USE_SERVICE_HISTORY = True
USE_DIALOGUE_HISTORY = True
USE_FULL_SLOT_DESCRIPTIONS = False
WORD_DROPOUT = 0.1
SCHEMA_AUGMENT_PROB = 0.1
USE_NATURAL_SYS_UTTR = False

BATCH_SIZE = 24
LEARNING_RATE = 2e-5
LR_SCHEDULER = 'constant'
DROPOUT = 0.3
USE_BIN_FEATS = 1

DATASET_PATH = 'dstc8-schema-guided-dialogue'

MAX_INTENTS = 5
MAX_CAT_VALUES = 11
MAX_SLOTS = 17
MAX_SLOTS_OTHER_SERVICE = 40
MAX_VALUES_PER_SERVICE = 23
NUM_BIN_FEATS = 6

```

Resultados Finales Iteración 2: EPOCH 3 , batch size 24, 70 Archivos

Aspecto General	Solicitudes de Slots
	{'active_intent_accuracy': 0.9595045442343828, 'requested_slots_f1': 0.9922754406820825, 'requested_slots_precision': 0.9934488885480179, 'requested_slots_recall': 0.9957050274187982, 'average_goal_accuracy': 0.9461145484671118, 'average_cat_accuracy': 0.9401984712604532, 'average_noncat_accuracy': 0.9495434371844801, 'joint_goal_accuracy': 0.7980269156344671, 'joint_cat_accuracy': 0.9112282290937539, 'joint_noncat_accuracy': 0.8594990851708403}
	Saving checkpoint for epoch 3 100% 3934/3934 [1:59:13<00:00, 1.82s/it]
	Train loss: 0.01355232761144315
	'requested_slots_f1': 0.9922754406820825, 'requested_slots_precision': 0.9934488885480179, 'requested_slots_recall': 0.9957050274187982,
Bueno!!! (Se debe a otros aspectos específicos)	Muy bueno!!!

Aspecto Específicos

```
"Banks": {  
    "active_intent_accuracy": 0.963091922005571,  
    "average_cat_accuracy": 0.961449498843485,  
    "average_goal_accuracy": 0.9708885890516575,  
    "average_noncat_accuracy": 0.9797509578544061,  
    "joint_cat_accuracy": 0.9345403899721448,  
    "joint_goal_accuracy": 0.9298746518105849,  
    "joint_noncat_accuracy": 0.9882033426183844,  
    "requested_slots_f1": 1.0,  
    "requested_slots_precision": 1.0,  
    "requested_slots_recall": 1.0}
```

Muy bueno!!!

Resultados Finales Iteración 3: EPOCH 4, batch size 24 , 70 Archivos

Aspecto General	Solicitudes de Slots
	{'active_intent_accuracy': 0.957625493730107, 'requested_slots_f1': 0.9938080811954341, 'requested_slots_precision': 0.996797944548836, 'requested_slots_recall': 0.9950722859224604, 'average_goal_accuracy': 0.95681376658137, 'average_cat_accuracy': 0.938888258220759, 'average_noncat_accuracy': 0.9649726741670271, 'joint_goal_accuracy': 0.8056715143996624, 'joint_cat_accuracy': 0.8898175978918763, 'joint_noncat_accuracy': 0.8862853580166491}
	Saving checkpoint for epoch 4 100% 3934/3934 [1:59:25<00:00, 1.82s/it]
	Train loss: 0.010712403475325157
	'requested_slots_f1': 0.9938080811954341, 'requested_slots_precision': 0.996797944548836, 'requested_slots_recall': 0.9950722859224604
Muy Bueno!!! (Se debe a otros aspectos específicos)	Muy bueno!!!

Aspecto Específicos

```
"Banks": {  
    "active_intent_accuracy": 0.9428969359331476,  
    "average_cat_accuracy": 0.981881264456438,  
    "average_goal_accuracy": 0.97666730917501927,  
    "average_noncat_accuracy": 0.9490517241379309,  
    "joint_cat_accuracy": 0.9596100278551533,  
    "joint_goal_accuracy": 0.6355571030640668,  
    "joint_noncat_accuracy": 0.661566852367688,  
    "requested_slots_f1": 0.9923398328690808,  
    "requested_slots_precision": 0.9905988857938719,  
    "requested_slots_recall": 0.9986072423398329  
}
```

Muy bueno!!!

Resultados Finales Iteración 4: EPOCH 5, batch size 24, 70 Archivos

Aspecto General	Solicitudes de Slots
	{'active_intent_accuracy': 0.955209571653181, 'requested_slots_f1': 0.9932328616533088, 'requested_slots_precision': 0.9965614654037401, 'requested_slots_recall': 0.9944203704413851, 'average_goal_accuracy': 0.9488725344863085, 'average_cat_accuracy': 0.9370719340068868, 'average_noncat_accuracy': 0.9549390523582864, 'joint_goal_accuracy': 0.8035043179813629, 'joint_cat_accuracy': 0.9008934821097706, 'joint_noncat_accuracy': 0.8746721207194079}
	Saving checkpoint for epoch 5 100% 3934/3934 [1:59:23<00:00, 1.82s/it] Train loss: 0.008833289376339162
joint_goal_accuracy': 0.8035043179813629	'requested_slots_f1': 0.9932328616533088, 'requested_slots_precision': 0.9965614654037401, 'requested_slots_recall': 0.9944203704413851,
Muy Bueno!!! (Se debe a otros aspectos específicos)	Muy bueno!!!

Aspecto Específicos

```
"Banks": {
  "active_intent_accuracy": 0.9428969359331476,
  "average_cat_accuracy": 0.981881264456438,
  "average_goal_accuracy": 0.9766730917501927,
  "average_noncat_accuracy": 0.9490517241379309,
  "joint_cat_accuracy": 0.9596100278551533,
  "joint_goal_accuracy": 0.6355571030640668,
  "joint_noncat_accuracy": 0.661566852367688,
  "requested_slots_f1": 0.9923398328690808,
  "requested_slots_precision": 0.9905988857938719,
  "requested_slots_recall": 0.9986072423398329
},
```

Muy bueno!!!

Conclusiones del Modelo

- **Optimización del Batch Size:**
 - Aumentar el tamaño del lote de 16 a 24 resultó en una mejora significativa en la eficiencia del entrenamiento, proporcionando un equilibrio entre la aceleración del proceso y la capacidad de la GPU.
 - Se encontró un límite en la capacidad de la GPU al intentar aumentar el tamaño del lote más allá de 24, indicando que este es el máximo soportable por la configuración actual de hardware.
- **Uso de GPU T4 y RAM:**
 - La GPU T4 demostró ser la más adecuada para este modelo, particularmente después de ampliar su RAM. Esta configuración ofreció un rendimiento óptimo, facilitando un procesamiento más eficiente.
- **Mejora en Métricas de Rendimiento con Aumento de Epochs:**
 - Al incrementar el número de epochs hasta 4, se observaron mejoras significativas en las métricas de precisión de slots (F1, precisión y recall).
 - El **joint_goal_accuracy**, una métrica crucial para la efectividad del modelo DST, también mejoró al aumentar los epochs a 4. Sin embargo, no se observaron beneficios adicionales al incrementar a 5 epochs, lo que sugiere que 4 epochs es el número óptimo para este modelo.
- **Impacto de la Reducción del Dataset:**
 - Reducir el tamaño del dataset de 127 a 70 archivos no afectó negativamente las métricas de rendimiento ENSEEN en las pruebas, indicando que el modelo puede mantener su precisión incluso con un conjunto de datos más pequeño.
- **Análisis de la Pérdida de Entrenamiento:**
 - El modelo mostró una pérdida de entrenamiento de casi el 1% a partir del cuarto epoch, lo cual es un indicador de un buen ajuste del modelo sin signos evidentes de sobreajuste o subajuste.

Puntos adicionales del Modelo

1. Documentación y Acceso:

- La documentación completa del modelo está disponible en GitHub, proporcionando una guía detallada y recursos para su implementación. Puede accederse en: [Multitask Schema-Guided DST](#).

• Idioma y Dataset:

- El dataset utilizado actualmente está en inglés. Aunque no se encontraron datasets equivalentes en español, existe la posibilidad de traducir el dataset existente usando un traductor de alta calidad. Sin embargo, esto requiere una validación cuidadosa para asegurar que la traducción no altere significativamente el contexto o la precisión de los datos.

• Dependencias y Librerías:

- Antes de implementar el modelo, es esencial instalar las dependencias listadas en el archivo **requirements.txt**. Algunas de las librerías clave incluyen:
 - absl-py>=0.7.0**: Una librería que proporciona abstracciones de Python usadas por TensorFlow.
 - fuzzywuzzy>=0.17.0**: Útil para comparaciones de texto basadas en la similitud.
 - numpy>=1.16.1**: Fundamental para el manejo de matrices y operaciones matemáticas.
 - six>=1.12.0**: Proporciona compatibilidad entre Python 2 y 3.
 - tensorflow>=2.6.3**: Requerido para ciertas operaciones y modelos en el contexto de procesamiento de lenguaje natural.
 - transformers==4.18.0**: Proporciona acceso a modelos preentrenados como BERT y herramientas para trabajar con ellos.
 - torch==1.11.0**: PyTorch, esencial para la construcción y entrenamiento de modelos de aprendizaje profundo.
 - tqdm**: Para visualizar el progreso de las operaciones en bucles.
 - scikit-learn, matplotlib, nltk**: Utilizadas para procesamiento de datos, análisis y visualización.

Conclusión General:

Los puntos adicionales proporcionan información crucial para la correcta implementación y optimización del modelo de DST. La disponibilidad de documentación detallada en GitHub facilita la comprensión y el acceso al modelo. Mientras que el dataset se encuentra en inglés, la opción de traducirlo a español abre la posibilidad de expandir su aplicabilidad, aunque esto requiere pruebas rigurosas para garantizar la integridad del contexto. Finalmente, la lista de dependencias y librerías esencialmente subraya la importancia de preparar el entorno de desarrollo adecuadamente antes de proceder con la implementación del modelo, asegurando así una base sólida para su funcionamiento efectivo.

Bibliografía

- Lefteris. (n.d.). GitHub - lefteris12/multitask-schema-guided-dst: Code for "A Multi-Task BERT Model for Schema-Guided Dialogue State Tracking." GitHub. <https://github.com/lefteris12/multitask-schema-guided-dst/blob/main/mtsgdst/train.py>
- Kumar, A. (2020, 7 febrero). MA-DST: Multi-Attention Based Scalable Dialog State Tracking. arXiv.org. <https://arxiv.org/abs/2002.08898>
- Li, S. (2021b, enero 20). Zero-shot generalization in dialog state tracking through generative question answering. arXiv.org. <https://arxiv.org/abs/2101.08333>
- Lizi Liao, Le Hong Long, Yunshan Ma, Wenqiang Lei (2021). Dialogue State Tracking.
Enlace: https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00384/101875/Dialogue-State-Tracking-with-Incremental-Reasoning
- Guan-Lin Chao, Jan Lane. BERT-DST: Scalable End-to-End Dialogue State Tracking with Bidirectional Encoder Representations from Transformer.
Enlace: <https://arxiv.org/pdf/1907.03040.pdf>
- An, J., Kim, M., Cho, S., & Bang, J. (2022). Domain-Slot relationship modeling using a Pre-Trained Language encoder for Multi-Domain Dialogue state tracking. IEEE/ACM transactions on audio, speech, and language processing, 30, 2091-2102. <https://doi.org/10.1109/taslp.2022.3181350>
- Kapelonis, E., Georgiou, E., & Potamianos, A. (2022b). A Multi-Task BERT model for Schema-Guided dialogue state tracking. Interspeech 2022. <https://doi.org/10.21437/interspeech.2022-10852>
- Song, L., Yao, M., Bi, Y., Wu, Z., Wang, J., Xiao, J., Juan, W., & Yu, X. (2021). LS-DST: Long and Sparse Dialogue State Tracking with Smart History Collector in Insurance Marketing. SIGIR 2021 – Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. <https://doi.org/10.1145/3404835.3463058>
- Brabra, H., Báez, M., Benatallah, B., Gaaloul, W., Bouguelia, S., & Zamanirad, S. (2022). Dialogue Management in Conversational Systems: A review of approaches, challenges, and opportunities. IEEE Transactions on Cognitive and Developmental Systems, 14(3), 783-798. <https://doi.org/10.1109/tcds.2021.3086565>