ESCUELA SUPERIOR POLITÉCNICA AGROPECUARIA DE MANABÍ MANUEL FÉLIX LÓPEZ

CARRERA DE COMPUTACIÓN

INFORME DE TRABAJO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

MECANISMO: SISTEMATIZACIÓN DE EXPERIENCIAS PRÁCTICAS DE INVESTIGACIÓN Y/O INTERVENCIÓN

TEMA:

DESARROLLO DE UN ASISTENTE VIRTUAL EMPLEANDO TÉCNICAS DE COMPRENSIÓN DE LENGUAJE NATURAL

AUTORES:

JESÚS STEFANO CAJAPE BRAVO
SANDRO ANTONIO PALAU DELGADO

TUTOR:

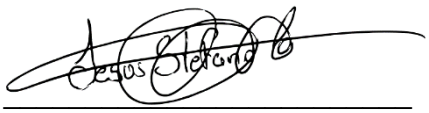
ING. FERNANDO RODRIGO MOREIRA MOREIRA, MGTR.

CALCETA, OCTUBRE DE 2023

DECLARACIÓN DE AUTORÍA

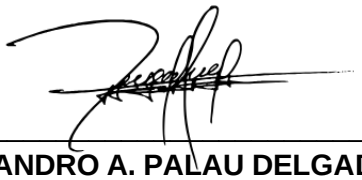
Nosotros **JESÚS STEFANO CAJAPE BRAVO** y **SANDRO ANTONIO PALAU DELGADO**, con cédulas de ciudadanía 1312189895 y 1311118440 respectivamente, declaramos bajo juramento que el Trabajo de Integración Curricular titulado: "**DESARROLLO DE UN ASISTENTE VIRTUAL EMPLEANDO TÉCNICAS DE COMPRENSIÓN DE LENGUAJE NATURAL**" es de nuestra autoría, que no ha sido previamente presentado para ningún grado o calificación profesional, y que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración, concedemos a favor de la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos, conservando a nuestro favor todos los derechos patrimoniales de autor sobre la obra, en conformidad con el Artículo 114 del Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación.



JESÚS S. CAJAPE BRAVO

CC:1312189895



SANDRO A. PALAU DELGADO

CC:1311118440

AUTORIZACIÓN DE PUBLICACIÓN

JESÚS STEFANO CAJAPE BRAVO y SANDRO ANTONIO PALAU DELGADO, con cédulas de ciudadanía 1312189895 y 1311118440 respectivamente, autorizamos a la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López, la publicación en la biblioteca del Trabajo de Integración Curricular titulado: "**DESARROLLO DE UN ASISTENTE VIRTUAL EMPLEANDO TÉCNICAS DE COMPRENSIÓN DE LENGUAJE NATURAL**", cuyo contenido, ideas y criterios son de nuestra exclusiva responsabilidad y total autoría.

JESÚS S. CAJAPE BRAVO

CC:1312189895

SANDRO A. PALAU DELGADO

CC:1311118440

CERTIFICACIÓN DEL TUTOR

MTR. FERNANDO RODRIGO MOREIRA MOREIRA, certifica haber tutelado el Trabajo de Integración Curricular titulado: "**DESARROLLO DE UN ASISTENTE VIRTUAL EMPLEANDO TÉCNICAS DE COMPRENSIÓN DE LENGUAJE NATURAL**", que ha sido desarrollada por **JESUS STEFANO CAJAPE BRAVO** y **SANDRO ANTONIO PALAU DELGADO**, previo la obtención del título de Ingeniero en Ciencias de la Computación de acuerdo al **REGLAMENTO DE LA UNIDAD DE INTEGRACIÓN CURRICULAR DE CARRERAS DE GRADO** de la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López.

Firmado electrónicamente por:
FERNANDO RODRIGO MOREIRA MOREIRA

MGTR. FERNANDO R. MOREIRA
CC: 1311726689
TUTOR

APROBACIÓN DEL TRIBUNAL

Los suscritos integrantes del Tribunal correspondiente, declaramos que hemos **APROBADO** el Trabajo de Integración Curricular titulado: "**DESARROLLO DE UN ASISTENTE VIRTUAL EMPLEANDO TÉCNICAS DE COMPRENSIÓN DE LENGUAJE NATURAL**", que ha sido desarrollado por **JESUS STEFANO CAJAPE BRAVO** y **SANDRO ANTONIO PALAU DELGADO**, previo a la obtención del título de **INGENIERO EN CIENCIAS DE LA COMPUTACIÓN**", de acuerdo al **REGLAMENTO DE LA UNIDAD DE INTEGRACIÓN CURRICULAR DE CARRERAS DE GRADO** de la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López.

Firmado electrónicamente por:
LUIS CRISTOBAL CEDENO VALAREZO

MGTR. LUIS C. CEDEÑO VALAREZO
CC: 1306246651
PRESIDENTE DEL TRIBUNAL

ANGEL ALBERTO VELEZ MERO
Firma digital.
avelez@espam.edu.ec

MGTR. ÁNGEL A. VÉLEZ MERO
CC: 1308648565
MIEMBRO DEL TRIBUNAL

Firmado digitalmente por ALFONSO TOMAS LOOR VERA

MGTR. ALFONSO T. LOOR VERA
CC: 1311655938
MIEMBRO DEL TRIBUNAL

AGRADECIMIENTO

A la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Feliz López y a la carrera de computación por habernos nutrido de conocimiento en el trascurso de nuestra educación superior, la misma que ha sido de una calidad notable, en la cual nos hemos formado como profesionales.

A nuestro tutor Ing. Fernando Moreira Moreira por habernos guiado en el trascurso del trabajo de integración curricular, y brindado su apoyo en el momento más difícil del transcurso del mismo.

Al Ing. Javier Hernán López Zambrano y al Ing. Yimmy Salvador Loor Vera, por darnos la oportunidad de participar en sus proyectos y desarrollo en la UDIV de infraestructura nuestro trabajo de integración curricular

A los docentes que nos impartieron su conocimiento en el trascurso de nuestra trayectoria académica, lo mismo que nos brindaron apoyo emocional y personal.
A nuestros compañeros haciendo mención especial al grupo “Amaya”.

LOS AUTORES

DEDICATORIA

Este trabajo de titulación tiene una dedicatoria especial a dos personas que lo hicieron posible: mi mamá, Karen Bravo, y mi abuelita, Noemy Murillo. Gracias a la motivación, apoyo y valores impartidos por estas maravillosas mujeres, puedo convertirme en el profesional que siempre desearon que fuera. Por esta razón, dedico este trabajo de titulación como muestra de mi agradecimiento por todo lo que han hecho por mí.

Quiero expresar mi agradecimiento también a mi padre, Charles Cajape, y a mi abuelita, Gevid Ávila, por el apoyo que me brindaron durante mis últimos años de educación superior. Asimismo, quiero agradecer al Ing. Fernando y Dra. Maryuri por su valiosa ayuda en momentos difíciles.

No quiero dejar de mencionar a mi gata Lucy y mi gato Fortuna, quienes han estado conmigo desde que llegaron a mi vida. Siempre me han acompañado en los momentos difíciles y de alguna manera siempre están presentes para motivarme y alegrar mis días. También quiero agradecer a One Piece, especialmente a los Mugiwaras, quienes me han enseñado el valor de seguir adelante sin importar las adversidades. Su historia y determinación han sido una fuente de inspiración para mí.

No puedo dejar de mencionar a Gema Intriago, quien me acompañó durante todo el proceso de este trabajo de titulación. Sus ánimos y apoyo incondicional fueron fundamentales, especialmente en los momentos más difíciles de mi vida. Agradezco también a mis amigos, quienes conocí a lo largo de mi trayectoria académica especialmente al grupo “Punto” y al grupo “Amaya”, por su apoyo emocional en el logro de este objetivo.

JESÚS STEFANO CAJAPE BRAVO

DEDICATORIA

Dedico este trabajo de integración curricular a las personas que más amo en mi vida; mi papá, Sandro Palau, y mi mamá, Eloísa Delgado, por haber sido la mayor fuente de apoyo y descanso mental durante la ejecución del proyecto, por sus consejos, preocupación constante, y por ser de las pocas personas conscientes de mi esfuerzo.

A mi hermana Paula Palau, por siempre estar en los momentos más complicados y en los más alegres también. Es un gran ejemplo para mí, cada día me enseña algo nuevo que me ayuda a complementar mi formación integral, y también por ser mi par perfecto, siempre que estamos juntos siento que somos inseparables.

A mi abuelita Cecilia García, que la quiero mucho, gracias por su apoyo y por hablar conmigo la mayoría de los días y escapar aunque sea un momento de la monotonía de las actividades cotidianas. También a Hachi, por ser el más expresivo y cariñoso, y porque independientemente de la situación nunca me abandona.

A todas las personas que conocí en mi época estudiantil y que los considero mis amigos, a los chicos del pensionado, a los “Mosqueteros”, al grupo “Punto” y al grupo “Amaya”, estoy convencido de que los buenos amigos también son parte de la familia, un saludo muy afectuoso a todos ellos. No puedo olvidarme de la Ing. Aura Zambrano, gracias por su constante preocupación por nuestro futuro académico y laboral, es algo que valoro mucho y un gesto que reconoceré siempre.

Una mención especial al Sr. Julio Alvia por su incansable servicio, siempre desinteresado y humilde, y también por hacer las veces de amigo, es algo que nunca lo voy a olvidar. Finalmente quiero agradecer la familia Basurto por acogerme en su domicilio, siempre les estaré agradecido.

SANDRO ANTONIO PALAU DELGADO

CONTENIDO GENERAL

CONTENIDO DE FIGURAS

CONTENIDO DE GRÁFICOS

CONTENIDO DE TABLAS

RESUMEN

El objetivo del presente trabajo de titulación fue desarrollar un asistente virtual basado en comprensión de lenguaje natural (NLU) en la Unidad de Docencia, Investigación y Vinculación (UDIV) de Infraestructura para brindar soporte a las aplicaciones de la carrera de Computación de la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López (ESPAM MFL). El trabajo se fundamentó con el Proceso Estándar Inter-Industrial para Minería de Datos (CRISP-DM), en primera instancia se logró recopilar las principales técnicas de NLU aplicadas en el desarrollo de asistentes virtuales, de esta forma se escogió a Rasa NLU como el framework de desarrollo del asistente virtual. Posteriormente, se realizó un análisis exploratorio de datos, que sirvió para delimitar las funciones del asistente virtual. El conjunto de datos de entrenamiento se formó a partir de la estructura del bloque NLU que ofrecía el framework de Rasa, se definieron las historias, el dominio, acciones, entidades e intenciones. Luego, se detallaron los componentes para ensamblar el modelo NLU y se definieron las fases del entrenamiento. Finalmente, se realizaron simulaciones de interacción para evaluar el modelo. Después de varias pruebas, el modelo logró alcanzar un alto rendimiento con métricas como un F1 Score de 0.89, una precisión de 0.91 y una exactitud de 0.88 en la clasificación de intenciones y entidades. Finalmente se implementó una Interfaz de Programación de Aplicaciones (API) que marcó la integración de los servicios del asistente virtual con una interfaz web conversacional.

PALABRAS CLAVE

Chatbot, Framework, Procesamiento, Acciones, Bloque NLU, Intenciones, APIs

ABSTRACT

The objective of this degree work was to develop a virtual assistant based on natural language understanding (NLU) in the Teaching, Research and Liaison Unit (UDIV) of Infrastructure to provide support to the applications of the Computer Science career at Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López (ESPAM MFL). The work was based on the Cross-Industry Standard Process for Data Mining (CRISP-DM). First, the main NLU techniques applied in the development of virtual assistants were compiled and Rasa NLU was chosen as the framework for the development of the virtual assistant. Subsequently, an exploratory data analysis was performed, which served to delimit the functions of the virtual assistant. The training data set was formed from the NLU block structure offered by the Rasa framework, the stories, domain, actions, entities, and intentions were defined. Then, the components to assemble the NLU model were detailed, and the training phases were defined. Finally, interaction simulations were performed to evaluate the model. After several tests, the model achieved a high performance with metrics such as an F1 Score of 0.89, a precision of 0.91 and an accuracy of 0.88 in the classification of intentions and entities. Finally, an Application Programming Interface (API) was implemented that marked the integration of the virtual assistant services with a conversational web interface.

KEYWORDS

Chatbot, Framework, Processing, Actions, NLU Block, Intentions, APIs

CAPITULO I. ANTECEDENTES

1.1. DESCRIPCIÓN DE LA INSTITUCIÓN

La Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López ESPAM MFL (2016), se constituyó como tal un 29 de abril de 1999, con la intención de participar junto a la compañía de otras instituciones para ayudar al progreso y desarrollo del país a través de la enseñanza universitaria y la investigación científica.

La misión de la ESPAM MFL es formar profesionales pertinentes con compromisos ético y social garantizando desde la calidad de funciones sustantivos. Y su visión, ser un centro de referencia en la capacitación de profesionales que contribuyan al desarrollo agropecuario regional". Una vez que se haya logrado el objetivo de su misión a través de los esfuerzos académicos, la Institución se convertirá en un referente académico dentro de la cual se educarán varias generaciones de profesionales de la zona cuatro y el país, por lo que su visión es "ser un centro de referencia en la formación de profesionales que contribuyan al desarrollo agropecuario regional" (ESPAM MFL, 2019).

La carrera de Computación, que forma parte de la oferta académica de la ESPAM MFL, tiene como misión la "formación de profesionales íntegros que conjuguen ciencia, tecnología y valores en su accionar, comprometidos con la comunidad en el manejo adecuado de programas y herramientas computacionales de última generación" y como visión: "ser referentes en la formación de profesionales de prestigio en el desarrollo de aplicaciones informática y soluciones de hardware" (ESPAM MFL, 2022).

Dentro de la carrera de Computación se encuentra la Unidad de Docencia, Investigación y Vinculación (UDIV). La UDIV de Infraestructura es la encargada de ofrecer una formación sólida, teórica, metodológica y práctica a los estudiantes de la ESPAM MFL en el análisis de problemas relacionados con los procesos del computador, auditoría y redes (UDIV de Infraestructura, 2022). De esta forma, la UDIV de Infraestructura acogerá este proyecto que se fundamenta en temas relacionados con el soporte de aplicaciones de la Carrera de

Computación mediante la vinculación de técnicas de inteligencia artificial para automatizar esta clase de procesos.

1.2. DESCRIPCIÓN DE LA INTERVENCIÓN

En el ámbito del desarrollo de asistentes virtuales, se enfrenta a un desafío fundamental relacionado con la naturaleza de la información no estructurada o cruda presente en el lenguaje natural. En lugar de encontrar predominantemente datos estructurados, es mucho más común encontrarse con información no estructurada en diversos contextos. En efecto, en escenarios de estudio reales, la principal fuente de información se encuentra dispersa en correos electrónicos, documentos de texto masivos y hasta en conversaciones y publicaciones cotidianas (Cortés y Valverde 2020). De acuerdo con Jiao (2020) esta característica del lenguaje natural, donde la información se presenta en forma desorganizada y sin un formato predefinido, plantea un desafío significativo para el procesamiento y análisis eficiente de los datos. En este sentido, el desarrollo progresivo de asistentes virtuales ha promulgado la aplicación de distintas técnicas de comprensión de lenguaje natural, que se convierten en piezas clave para abordar esta problemática.

Con el avance de los algoritmos en los procesos de inteligencia artificial, y aprovechándose de la necesidad de comprender el lenguaje natural, se ha creado una gran variedad de sistemas realistas de comprensión de conversaciones para varias aplicaciones de diversos dominios que son útiles para realizar tareas diarias de los usuarios finales (Jose & Lakshmi, 2018) (Duncan y Olmsted 2018).

Estos sistemas hacen referencia a los asistentes virtuales inteligentes y, de forma general, se encargan de tareas como la clasificación de dominios, llenado de espacios y predicción de intenciones. En el ámbito académico, esta clase de asistentes pueden modernizar la gestión institucional mediante la integración de diversos modelos adaptativos para la autorregulación de procesos, con el fin de contribuir al direccionamiento de información de la institución (Manjarrés-Betancur & Echeverri-Torres, 2020). Las arquitecturas más populares abarcan sistemas modulares, y dentro de sus elementos básicos, el módulo común

presente es el de comprensión del lenguaje natural o también denominado (NLU) (Rychalska et al., 2018).

La comprensión del lenguaje natural es un aspecto clave en los asistentes virtuales. Gracias a esto, un determinado agente puede entender distintas expresiones en el contexto del lenguaje común de una conversación promedio (Ramesh 2017) (Ait-Mlouk & Jiang 2020) (Namazifar et al. 2021) (Ramesh & Kolonin 2020) (Rychalska, Glabska, & Wroblewska 2018).

Para recapitular, un asistente virtual permite crear un agente, el cual recibe solicitudes por parte del usuario y reconoce a través de *keywords* (denominados entidades) la intención a la que debe ser dirigida la solicitud para generar una respuesta coherente (Betancur, 2020). Esta ventaja trajo consigo la propuesta de facilitar la consulta de información acerca de determinadas aplicaciones que son administradas por la UDIV de infraestructura. Dicho esto, los usuarios obtienen información de las aplicaciones por medio del asistente virtual, sin necesidad de interactuar directamente con la mismas, claramente agregando una capa intuitiva de consulta.

En el presente trabajo de titulación se desarrolló un asistente virtual que brinde soporte a las aplicaciones de la UDIV de infraestructura. El asistente virtual fue capaz comprender el lenguaje natural de los inputs de los usuarios, con el fin de gestionar la información de las aplicaciones y responder las solicitudes de los usuarios de forma rápida y precisa. Cabe mencionar que este trabajo de titulación fue desarrollado en paralelo con el proyecto “Desarrollo de una interfaz conversacional integrada a un asistente virtual” donde Jéssica Johana Montes Vera y Carlos Pierre Quijije Vera son los postulantes responsables del mismo.

El desarrollo del asistente virtual funcionó como un complemento del trabajo previamente mencionado. Este fue capaz de detectar intenciones y entidades específicas que, de acuerdo con Jiménez (2019) y Virkar, Honmane, & Rao (2019), esta característica se forma durante la fase de entrenamiento del modelo del asistente virtual, en esta ocasión, a partir de datos relacionados a las aplicaciones de la UDIV de la infraestructura.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Desarrollar un asistente virtual basado en NLU en la UDIV de Infraestructura para brindar soporte a las aplicaciones de la carrera de Computación de la ESPAM MFL.

1.3.2. OBJETIVOS ESPECÍFICOS

- Revisar información bibliográfica acerca de las técnicas de comprensión de lenguaje natural implementadas en el desarrollo de asistentes virtuales.
- Definir técnicas de entrenamiento del modelo de comprensión de lenguaje natural.
- Evaluar el análisis de comprensión de lenguaje natural del asistente virtual con respecto a las solicitudes de los usuarios finales.
- Proveer la API de servicio del asistente virtual para establecer conexión con la interfaz conversacional y las aplicaciones de la UDIV de infraestructura.

CAPITULO II. DESARROLLO METODOLÓGICO DE LA INTERVENCIÓN

2.1. CRISP-DM

En este trabajo de titulación se entrenó un modelo a partir de la definición de técnicas de comprensión de lenguaje natural, por esta razón, se utilizó la metodología CRISP-DM (Cross Industrie Standard Process for Data Mining), que consiste en llevar una orientación en los trabajos de minería de datos (Valderrama-Chauca et al., 2022). Esta metodología incluye descripciones de las fases normales de un proyecto, las tareas necesarias en cada fase y una explicación de las relaciones entre las tareas (Carlos et al., 2015). A continuación, se definen las fases de la metodología CRISP-DM.

2.1.1 COMPRENSIÓN DEL NEGOCIO

Se utilizó la revisión bibliográfica para identificar las técnicas de NLU más utilizadas en el desarrollo de asistentes virtuales, de acuerdo con Higinio et al, (2021), este método permite analizar el estado más actual de la información, aprovechando los resultados y lecciones aprendidas de otros investigadores para evitar la duplicación innecesaria de esfuerzos y recursos.

En este punto, se asociaron plataformas y herramientas de desarrollo de chatbots más complejas; por ejemplo, se encontraron frameworks dedicados a la creación de asistentes virtuales que integraban por defecto distintas técnicas independientes de entrenamiento de modelos basados en comprensión de lenguaje natural. También se evidenció que la base de un asistente virtual consta de un bloque NLU formado por un ensamblaje de modelos con el objetivo de mejorar el rendimiento y precisión del mismo.

Se determinó que el marco de aplicación tenía que seguir una línea de combinación de técnicas aisladas de NLU. De esta forma se aprovechó la información recopilada, para extraer los frameworks de desarrollo de asistentes virtuales, en caso de ser aplicados. Esta recopilación de información ayudó en la toma de decisiones para seleccionar el framework en el cual se desarrolló el asistente virtual.

2.1.2 COMPRENSIÓN DE LOS DATOS

En esta fase se determinó el alcance del modelo a entrenar, así como también se limitaron las funciones y servicios específicos de los que debía disponer el asistente virtual. La función principal fue establecida como la respuesta hacia las solicitudes de información de las aplicaciones de la UDIV de infraestructura, sin embargo, se realizó el entrenamiento de otros campos de la conversación relacionados a la comunicación básica del usuario, y otras excepciones fuera de la regla con respecto a su función principal; todos estos factores que fueron considerados como requisitos básicos, se detallaron más adelante.

Se logró identificar las funcionalidades de las aplicaciones en producción de la UDIV de Infraestructura, y también se verificó las formas específicas en las que las aplicaciones manipulan los datos que recoge, haciendo hincapié en el formato en el que devuelve los mismos. Del mismo modo, se establecieron casos posibles de interacción, que funcionaron como supuestos datos para el entrenamiento del modelo.

Otra de las tareas que se aplicó dentro de esta fase, fue un análisis exploratorio de los datos que funcionó en paralelo como un proceso de refinación, extracción de características significativas, y validación extra del conjunto de datos para entrenar el modelo en su primera iteración.

2.1.3 PREPARACIÓN DE LOS DATOS

Se analizó la estructura del framework Rasa NLU para comprender las tareas del entrenamiento de un modelo y del bloque que implica las actividades de comprensión de lenguaje natural. En consecuencia, también se reconoció la forma en que los datos tuvieron que estructurarse para que sean legibles por los pipelines de dicho framework. Estos datos estaban formados por ejemplos de interacción entre los usuarios y el asistente, junto con las respuestas apropiadas para que sean proporcionadas por el mismo.

También se detalló el funcionamiento de las acciones de Rasa, y se explicó cómo algunos datos específicos de entrenamiento fueron utilizados para influir como entidades en la interacción usuario – asistente.

2.1.4 MODELADO

En esta etapa se seleccionaron los componentes de comprensión de lenguaje natural que más se adaptaron a las características del corpus especificado en la preparación de los datos, además, se da una breve descripción da cada uno de los componentes de entrenamiento. De esta forma se dio apertura visual del bloque NLU con respecto a la estructura y orden de un proyecto creado en el framework de Rasa, también se definieron las capas de entrenamiento, es decir, se estableció el ensamblaje del modelo y las fases a las que se sometió el entrenamiento, detallando la configuración de cada uno de los hiperparámetros.

Por otra parte, también se detalló en el documento la estructura del funcionamiento del framework, incluyendo el endpoint que proporcionó el servicio del asistente virtual, y al que se conectó la interfaz web conversacional, que como se mencionó anteriormente pertenece a otro trabajo de integración curricular, que se desarrolló en simultáneo con el asistente virtual, y los endpoints que se utilizaron para acceder a las aplicaciones de la UDIV de Infraestructura.

Asimismo, se identificaron estrategias de verificación de la calidad del modelo seleccionado, se realizó la introducción del aprendizaje por refuerzo mediante la aplicación de métricas de validación, en tiempo real, de los resultados del entrenamiento del modelo.

2.1.5 EVALUACIÓN

Se explicaron todos los métodos posibles para entrenar e interactuar con un modelo dentro del framework de Rasa, además, se establecieron las métricas de evaluación del modelo de compresión de lenguaje natural; se definió una ruta de pruebas específica simulando un escenario de comunicación con el asistente para que sea evaluado el desempeño del modelo con respecto al reconocimiento de entidades e intenciones de los inputs, así como también las respuestas del asistente virtual bajo las acciones que fueron programadas previamente. Teniendo en cuenta las solicitudes propuestas en el entrenamiento por los autores y la respectiva validación del modelo utilizando solicitudes de usuarios, se pudo cuantificar la confianza con respecto a la toma decisiones y la selección de una determinada respuesta del modelo acorde al input del usuario.

2.1.6 DESPLIEGUE

Posteriormente al entrenamiento y validación del modelo de NLU aplicado, se desarrolló una interfaz de programación de aplicación (API), cuyo objetivo fue liberar los servicios del asistente virtual para que fuese consumido por la interfaz conversacional propuesta en fases anteriores del proyecto como parte de la integración final del servicio.

Además, se realizaron pruebas locales para validar el funcionamiento de la API que se encargó de devolver la respuesta del modelo. De esta forma, se evidenció el funcionamiento de los procesos internos de interacción del asistente, así como también se validó la operabilidad de las acciones externas del mismo, y se verificó que en cualquier instancia nueva de uso de la aplicación, el modelo que se utilice fuese el más reciente o el último que se haya lanzado a producción.

CAPITULO III. DESCRIPCIÓN DE LA EXPERIENCIA

En la presente descripción de experiencias se expusieron los resultados mediante la especificación de entregables en función de cada uno de los objetivos específicos propuestos previamente, a partir de la metodología planteada en el capítulo de desarrollo metodológico de la intervención.

3.1. COMPRENSIÓN DEL NEGOCIO

Mediante una cuidadosa selección de palabras clave, entre las que destacaron “Chatbot”, “Framework”, “Procesamiento”, “Acciones”, “Bloque NLU”, “Intenciones”, “APIs”, se pudo realizar un análisis exhaustivo del estado del arte en el campo de las técnicas de NLU aplicadas en el desarrollo de asistentes virtuales. Se logró obtener una visión actualizada de los avances más recientes, las investigaciones destacadas y los descubrimientos relevantes en el área de la comprensión del lenguaje natural. Estas palabras clave fueron utilizadas para seleccionar artículos específicos que otorgaron información de una determinada herramienta utilizada para el desarrollo de estos asistentes. El objetivo principal de esta recopilación fue aportar a la selección del framework, técnicas y componentes NLU para el desarrollo del asistente virtual.

Chatbot: Es un programa de software diseñado para simular una conversación humana y responder preguntas o realizar acciones específicas a través de una interfaz de chat. Los chatbots pueden utilizar técnicas de procesamiento del lenguaje natural, para comprender el lenguaje humano y generar respuestas relevantes.

Framework: Un framework es un conjunto de herramientas, bibliotecas y componentes que proporcionan una estructura básica para el desarrollo de software. Estas herramientas facilitan la creación y el mantenimiento de aplicaciones al ofrecer una base sólida y predefinida en la que los desarrolladores pueden construir sus proyectos

Procesamiento: El procesamiento, en el contexto de la informática, es el conjunto de operaciones que se realizan sobre los datos con el fin de obtener un resultado deseado. En el ámbito del procesamiento del lenguaje natural, implica la manipulación y análisis de texto o lenguaje humano.

Acciones: En el contexto de los chatbots y la inteligencia artificial, las acciones se refieren a las respuestas o tareas que un sistema puede llevar a cabo en respuesta a las solicitudes de los usuarios.

Bloque NLU: El bloque NLU (Understanding Natural Language Understanding) es un componente o módulo dentro de un sistema de chatbot o asistente virtual que se encarga de comprender y procesar el lenguaje humano.

Intenciones: En el contexto del procesamiento del lenguaje natural (NLP), las intenciones son los objetivos o propósitos que subyacen en el lenguaje utilizado por un usuario. En un sistema de chatbot, se busca identificar las intenciones del usuario para poder proporcionar respuestas o realizar las acciones adecuadas.

APIs: Son conjuntos de reglas y protocolos que permiten a diferentes aplicaciones comunicarse y compartir datos entre sí. Las APIs definen cómo se deben solicitar y enviar datos entre diferentes sistemas, lo que facilita la integración y la interacción entre ellos.

Tabla 1: Listado de artículos recopilados.

Nombre	Autor	Año	Técnica implementada	Clasificación
Chatbol, A Chatbot For The Spanish “La Liga”	Carlos Segura, Alex Palau, Jordi Luque, Marta R. Costa-Juss ` a and Rafael E Banchs	2018	Rasa NLU	Bloque NLU
Using A Multiplatform Chatbot As An Online Tutor In A University Course	Lap-Kei Lee, Yin-Chun Fung, Yau-Wai Pun, Ka-Kin Wong, Maverick Tai-Yin Yu	2020	DialogFlow	Agente externo
A Chatbot For Changing Lifestyle In Education	E.Kasthuri Dr.S.Balaji	2021	Algoritmo LSTM (RNN)	Deep learning

Language Chatbot–The Design And Implementation Of English Language Transfer Learning Agent Apps	Nuobei Shi, Qin Zeng, Raymond Lee	2021	BERT (Google) GPT-2 (Open AI) (Componente gratuito)	Agente externo
Sogo: A Social Intelligent Negotiation Dialogue System	Ran Zhao, Oscar J. Romero, Alex Rudnicky	2018	GRU (RNN)	Deep learning
An Intelligent Chatbot System Based On Entity Extraction Using RASA NLU And Neural Network	Anran Jiao	2020	Rasa NLU	Bloque NLU
Humanizing The Chatbot With Semantics Based Natural Language Generation	Mayuresh Virkar, Vikas Honmane, S. Upendra Rao	2019	Alogitmo LSTM (Post respuesta del chatbot)	Deep learning
Goal-Oriented Modelling For Virtual Assistants	Jonathan Leung, Zhiqi Shen, Chunyan Miao	2019	Herramienta gráfica para la creación de asistentes virtuales basado en Goal Net	Agente externo
Askco: A Multi-Language And Extensible Smart Virtual Assistant	Mattia Atzeni and Maurizio Atzori	2019	AskCO (Agente externo de NLU)	Agente externo
Kbot: A Knowledge Graph Based Chatbot For Natural Language Understanding Over Linked Data	Addi Ait-Mlouk , Lili Jiang	2020	Máquinas de soporte de vectores - Reconocimiento de entidades nombradas	Bloque NLU

A New Chatbot For Customer Service On Social Media	Anbang Xu, Zhe Lie, Yufan Guo, Vibha Sinha , Rama Akkiraju	2017	Algoritmo LSTM (RNN)	Deep learning
Superagent: A Customer Service Chatbot For E-Commerce Websites	Lei Cui , Shaohan Huang , Furu Wei, Chuanqi Tan, Chaoqun Duan, and Ming Zhou	2017	Hechos pregunta-respuesta, búsqueda pregunta-respuesta, minería de opinión, conversación chit-chat	Bloque NLP
Pharmabot: A Pediatric Generic Medicine Consultant Chatbot	Benilda Eleonor V. Comendador, Bien Michael B. Francisco, Jefferson S. Medenilla, Sharleen Mae T. Nacion, and Timothy Bryle E. Serac	2015	Análisis sintático izq-der	Bloque NLU
Virtual Assistant Prototype For Managing Medication Using Messaging Platgorms	Surya Roca, Jose Garcia, Jorge Sancho, Alvaro Alesanco	2019	Lenguaje de marcado de inteligencia artificial (Lenguaje independiente dedicado a la creación de asistentes virtuales)	Lenguaje de programación independiente
Alime Chat: A Sequence To Sequence And Rerank Based Chatbot Engine	Minghui Qiu, Feng-Lin Li, Siyu Wang, Xing Gao, Yan Chen, Weipeng Zhao, Haiqing Chen, Jun Huang, Wei Chu	2017	Hechos pregunta-respuesta, modelo de recuperación de información, modelo seq2seq	Bloque NLU
Multi-Intent Hierarchical Ntural Language Understanding For Chatbots	Barbara Rychalska, Helena Glabska, Anna Wroblewska	2018	Modelo de multi-intenciones: Modelo Wit, CRF, Redes neuronales recurrentes	Bloque NLU
Program Synthesis From Natural Language	Xi Victoria Lin, Chenglong Wang, Deric Pang, Kevin Vu,	2020	Redes neuronales recurrentes	Deep learning

Using Recurrent Neural Networks	Luke Zelemoyer, Michael D. Ernst			
Language Model Is All You Need: Natural Language Understanding As Question Answerning	Mahdi Namazifar, Alexandros Papangelis, Gokhan Tur, Dilek Hakkani-tur	2021	Respuesta a preguntas para la comprensión de lenguaje natural	Bloque NLU
A Review Of Technologies For Conversational Systems	Julia Masche and Nguyen-Thinh Le	2021	Análsis semántico latente basado en un modelo de espacio vectorial: TF-IDF	Bloque NLU
CLOUD COMPUTING 2018 Proceedings Of The Ninth International Conference On Cloud Computing, Grids, And Virtualization	Bob Duncan, Aspen Olmsted	2018	Watson NLU: Deep Learning para extraer metadatos del texto, como palabras clave, emoción y sintaxis	Agente externo
A Survey Of Desing Techniques For Conversational Agents	Kiran Ramesh, Surya Ravishankaran, Abhishek Joshi, K. Chandrasekaran	2017	Redes neuronales recurrentes - LSTM	Deep learning
Learning To Paraphrase For Question Answering	Li Dong, and Jonathan Mallinson, and Siva Reddy, and Mirella Lapata	2022	Modelo de pregunta-respuesta, generación de paráfrasis	Bloque NLU
A Methodology For Generating Natural Language Paraphrases	Isidoros Perikos, Ioannis Hatzilygeroudis	2016	Etiquetado gramatical, analisis sintácticatico, reconicimiento de entidades nombradas	Bloque NLU
Detailed Study Of Deep Learning Models For Natural Language Processing	Megha Gupta, Shailesh Kumanr Verma, Priyanshu Jain	2020	Redes neuronales recurrentes y recursivas	Deep learning

Basic Dependency Parsing In Natural Language Inference	Aleshinloye Abass Yusuf, Nnanna Agwu Nwojo, Moussa Mahamat Boukar	2017	Normalización del análisis sintáctico	Bloque NLU
Enhancing Rasa NLU Model For Vietnamese Chatbot	Nguyen Thi Mai Trang, Maxim Shcherbakov	2021	Rasa NLU	Bloque NLU
Voice-Based Road Navigation System Using Natural Language Processing (NLP)	Pooja Withanage, Tharaka Liyanegem Naditha Deeyakaduwe, Eshan Dias, Samantha Thelijjagoda	2018	Modelo de reconocimiento automático del habla y CoreNLP framework	Bloque NLP
Interpretable Natural Language Segmentation Based On Link Grammar	Vignav Ramesh, Anton Kolonin	2020	Modelo de segmentación de texto y oraciones	Bloque NLU
Joint Slot Filling And Intent Predictionfor Natual Language Understanding In Frames Dataset	Jose K J, Lakshmi K S	2018	Redes neurorales recurrentes - LSTM	Deep learning
Aplicación Para Crear Chatbots Y Asistentes Virtuales Inteligentes	BUENO JIMÉNEZ Adrián	2019	Snips NLU: Motor de comprensión de lenguaje natural desarrollado en python	Bloque NLU

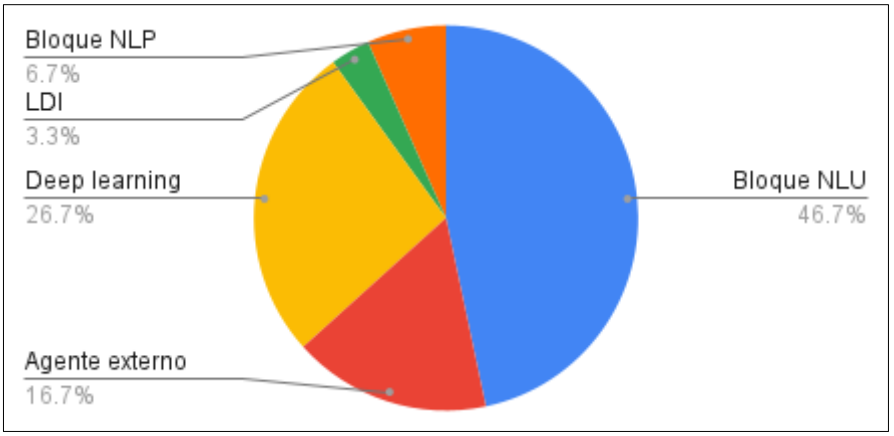
Fuente: Los Autores.

El listado que se muestra en el Tabla 1 fue el resultado de agregar filtros a los campos descriptivos del listado, entre los que inicialmente se encontraban año de publicación, autores, y título del artículo. Estos parámetros que se incluyeron después se definieron como “Técnica implementada”, haciendo referencia al nombre específico de la técnica de comprensión de lenguaje natural aplicada, y “Clasificación”, en donde se agrupan las técnicas que se detallan previamente,

según la naturaleza y procedencia de esta. En consecuencia, la clasificación de técnicas se divide en los siguientes grupos:

- **Bloque NLU:** Se aplicaron técnicas como el reconocimiento de entidades nombradas, análisis sintáctico-semántico vectorial de preguntas-respuestas, y otros modelos de segmentación de texto y oraciones en la fase del entrenamiento del modelo.
- **Bloque NLP:** Técnicas y procesos similares que fueron ejecutados en el Bloque NLU, incluyendo aplicaciones relacionadas al reconocimiento de voz.
- **Agente externo:** Se utilizaron componentes gratuitos de servicios externos para ejecutar y evaluar modelos de comprensión de lenguaje natural ej. *BERT* (Google), *GPT-2* (Open AI).
- **Deep Learning:** Implementación de redes neuronales recurrentes en procesos internos del entrenamiento del modelo de NLU.
- **Lenguaje de programación independiente:** Creación de procesos de NLU (desde cero) mediante un lenguaje dedicado a la creación de asistentes virtuales.

Gráfico 1: Clasificación de técnicas aplicadas.



Fuente: Los Autores.

Como se observa en el gráfico 1, las técnicas correspondientes al bloque de NLU predominan en el desarrollo de asistentes virtuales con un 46,7%. Se puede inferir que, este bloque es esencial para la creación de un chatbot porque en principio permite comprender el lenguaje natural utilizado por el usuario. Además, el bloque NLU ayudó a identificar la intención detrás de la solicitud del usuario.

Los estudios realizados en esta investigación también permitieron identificar varios frameworks utilizados en el desarrollo de chatbots. Entre los más destacados se encuentran Rasa NLU, Dialogflow y Microsoft Bot Framework.

Rasa NLU: Es un framework de código abierto que proporciona herramientas y bibliotecas de aprendizaje automático para construir chatbots con capacidades de procesamiento del lenguaje natural y diálogos contextualmente inteligentes. Su flexibilidad y personalización permiten adaptar el comportamiento del chatbot a necesidades específicas, y ofrece integraciones con plataformas populares.

Dialogflow: Es un framework de Google Cloud que ofrece capacidades de procesamiento del lenguaje natural para desarrollar chatbots, proporcionando una interfaz fácil de usar para crear conversaciones basadas en texto o voz.

Microsoft Bot Framework: Es un conjunto de herramientas y servicios que permiten a los desarrolladores crear y desplegar chatbots en diversas plataformas, brindando una plataforma flexible para su desarrollo.

Sin embargo, Rasa NLU se destaca por su enfoque en el contexto y los diálogos, permitiendo construir conversaciones más complejas y mantener un seguimiento del contexto durante la interacción con los usuarios. Su capacidad de entrenar modelos de lenguaje personalizados y su comunidad activa y soporte brindan recursos valiosos para el desarrollo de chatbots altamente personalizados. Por estas razones, se decidió utilizar Rasa como el framework principal en este trabajo de titulación.

3.2. COMPRENSIÓN DE LOS DATOS

Se realizó un estudio de caso utilizando el framework Rasa NLU, como se puede ver en el (Anexo I). Este framework proporciona una herramienta efectiva y fácil de usar para el desarrollo de modelos de lenguaje natural, lo que permitió lograr los objetivos del estudio de manera eficiente.

Se realizó un análisis exploratorio de datos (EDA) corpus de datos inicial seleccionado para ejecutar la primera iteración del entrenamiento del modelo. Estos ejemplos se consideraron como entradas (*inputs*) hacia el chatbot, es decir, cualquier mensaje o solicitud escrita que haya sido propuesta por el usuario. De esta forma, los ejemplos mencionados fueron clasificados según el propósito específico del asistente, que, en este caso de uso, se divide en:

- **Saludos y despedidas:** El usuario por instinto e iniciar el respectivo de consumo del asistente para recibir soporte acerca de la gestión de espacios de aulas y laboratorios iniciará la conversación con el chatbot mediante un saludo específico; de igual forma se estableció una regla en la que al iniciar sus servicios, el asistente siempre empezará con la conversación. De igual forma, para terminar con la instancia de servicio del asistente, el usuario usualmente se debería despedir o agradecer en una determinada instancia de la conversación; en caso de cumplirse esta situación, se agregó otra regla específica en la que se incluyó un *timeout* o tiempo de espera prudente hasta llegar al punto de no comunicación usuario-asistente, en el que automáticamente se cancela la instancia de servicio. En consecuencia, se entrenó al modelo con distintas variaciones de saludos y despedidas clásico ej. “hola” y “adiós”, y además se incluyeron algunos ecuatorianismos.
- **Afirmaciones:** Este campo hace referencia a algunas posibles consideraciones de *inputs* del usuario hacia el asistente, únicamente cuando esté de acuerdo con el mensaje proporcionado por el chatbot.
- **Negaciones:** Al igual que las afirmaciones, este conjunto de entradas engloban la mayor parte de respuestas posibles del usuario hacia el asistente, únicamente cuando tome una postura de desacuerdo en base al mensaje proporcionado por el chatbot.
- **Estados de ánimo:** Conjunto de posibles respuestas a una conversación trivial no referencial al tema principal del servicio (consultas sobre la gestión de espacios), el usuario puede incluir, de acuerdo con su estado de ánimo asumiendo que está hablando de forma natural con cualquier persona, el dominio consiste en dos *moods*: triste y feliz.
- **Desafíos:** Se incluye la posibilidad de interactuar con el chatbot en las que

el usuario puede preguntar por la naturaleza del asistente virtual, por lo que, el dominio de inputs fue basado en variaciones de la clásica pregunta "¿eres un bot?"

- **Tiempo:** A partir de una conversación fuera del tema de experticia, el usuario puede solicitar la hora, por lo tanto, en esta instancia, el dominio del modelo fue considerado como la variación de la pregunta clásica "¿qué hora es?
- **Experticia:** Conversación directamente relacionada con la gestión de espacios, en este caso el dominio del modelo se estableció en el conocimiento del responsable, hora y lugar de la reserva del aula/laboratorio, el modelo fue capaz de reconocer la intención de cualquier pregunta acerca de estos temas que integren las entidades mencionadas previamente.

Tabla 2: Ejemplos de Inputs dirigidos al chatbot.

Texto de entrada						
Saludos/despedidas	**Afirmaciones**	**Negaciones**	**Estados de ánimo**	**Desafíos**	**Tiempo**	**Experticia**
Buenas noches	Así es	En realidad, no	Contento	¿Eres humano?	Dime la hora	¿Cuál es el aula libre en este momento?
Buenos días	Cierto	De ninguna manera	Estoy genial	¿Eres un bot?	¿Qué hora es?	¿Qué laboratorio está disponible?
Hola	Claro	Nel		¿Que eres?	¿Puedes	¿Está disponib

			Estoy triste		decirme que hora es?	le algún aula el viernes por la mañana ?
Adiós	Correcto	Negativo	Estoy decepcionado	¿Quién te creó?	¿Cuáles la hora actual?	¿Qué aula está disponible el martes?
Chau	Absolutamente	Nope	Me siento bien	¿Eres un ser humano o una máquina?	¿Quisieras decirme que hora es?	¿Qué aula esta libre?

Fuente: Los Autores.

En la Tabla 2 se muestra solo algunos ejemplos de entradas de texto del usuario por categoría, el dataset seleccionado para el corpus y primera iteración del entrenamiento cuenta con 133 registros del mismo estilo. Cabe recalcar que esta cantidad está aislada al análisis exploratorio de datos, y que aumentó considerablemente en las próximas iteraciones de entrenamiento, así como también en las fases de validación y prueba del modelo.

Posteriormente, se realizó el análisis exploratorio tal y como se evidencia en el **(Anexo II).** En esta etapa se incluyeron análisis del promedio de la longitud de la entrada del texto, exclusión de *stopwords* también conocidas como muletillas o palabras vacías, se revisa la repetición de palabras que aportan información relevante al modelo, definición de secuencias de palabras con su respectiva frecuencia, y la extracción de entidades con su respectiva clasificación. La estructura del análisis exploratorio de datos está compuesta a primera vista por una carpeta que contiene un script de utilitarios, el archivo de análisis (que es el

dataset referente al corpus), y un *notebook* orquestador que básicamente va llamando a las funciones necesarias para realizar todas las actividades descritas previamente. Estas funciones, y otras librerías cuya instalación es de carácter obligatorio, se almacenan dentro de un script de Python también denominados utilitarios. La secuencia de ejecución del análisis exploratorio es la siguiente:

1. Descarga e importe de módulos requeridos para el análisis.
2. Ejecución del script de utilitarios, es decir, asegurarse de que todas las funciones sean operables por el *notebook* orquestador.
3. Ejecución del *notebook* orquestador, en el que se pueden visualizar los resultados del análisis exploratorio de datos.

3.3. PREPARACIÓN DE LOS DATOS

Una vez que los campos de aplicación del asistente ya habían sido determinados con sus respectivos ejemplos de interacción, se identificaron y aplicaron las restricciones de formato necesarias para intervenir en el proceso de entrenamiento de la comprensión de lenguaje natural del modelo, como ya se explicó previamente, Rasa NLU, posee un bloque dedicado a la distribución de los datos para el entrenamiento de un modelo específico, a continuación, se muestra la respectiva división:

- **Personalización del entrenamiento de modelos específicos de comprensión de lenguaje natural:**
  Para entrenar un modelo de Rasa NLU, se necesitaron varios archivos, como el nlu.yml, rules.yml, stories.yml, y domain.yml, que describen las intenciones y entidades que el modelo debe reconocer, las reglas que debe seguir para el procesamiento de las frases, los escenarios de conversación y las acciones que el modelo debe realizar en respuesta a las solicitudes de los usuarios. A continuación, se explicará cómo funcionaron estos archivos en Rasa NLU:
    1. **nlu.yml:** Este archivo fue utilizado para definir los ejemplos de frases que el modelo debe aprender a reconocer. En este archivo,

se definen las intenciones (intentions) y las entidades (entities) que el modelo debe reconocer en las frases. Cada ejemplo se define con una frase y las intenciones y entidades asociadas a esa frase. Estos ejemplos son utilizados durante el entrenamiento del modelo para enseñarle a reconocer las intenciones y entidades específicas en diferentes contextos.

En el siguiente bloque de código, se demuestra cómo se creó una intención llamada "Saludar" y se enumeró todas las posibles entidades que el chatbot pudo reconocer como saludos. Con base en estas entidades, el modelo determinó cuándo debe saludar o llevar a cabo cualquier otra acción.

```
- intent: greet
  examples: |
    - buenas noches
    - buenas tardes
    - buenos días
```

2. **rules.yml:** Este archivo se utiliza para definir reglas que el modelo debe seguir durante el procesamiento de las frases. Estas reglas pueden ser útiles para mejorar la precisión del modelo y evitar errores comunes. Por ejemplo, se pueden definir reglas que indiquen que ciertas palabras siempre se deben tratar como entidades, o que ciertas combinaciones de palabras siempre deben indicar una intención específica, como lo podemos apreciar en el siguiente bloque de código:

```
rules:
- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye
```

3. **stories.yml:** Este archivo se utiliza para definir los escenarios de conversación que el modelo debe aprender a manejar. En este archivo, se definen las diferentes etapas de una conversación, junto con las intenciones y entidades asociadas a cada etapa.

Estas historias fueronutilizadas durante el entrenamiento del modelo para enseñarle a manejar diferentes situaciones y responder adecuadamente a las solicitudes de los usuarios. En el siguiente fragmento de código se puede apreciar cómo está estructurada una historia.

```
stories:
- story: greeting
    steps:
    - intent: greet
    - action: utter_greet
```

4. **domain.yml:** Este archivo se utilizó para definir el dominio de la aplicación, que incluye las diferentes intenciones y entidades que el modelo debe reconocer, así como las respuestas que el modelo proporcionó en función de las solicitudes de los usuarios. En este archivo, se definen las acciones que el modelo puede realizar, así como las plantillas que se utilizaron para generar respuestas personalizadas a las solicitudes de los usuarios.

```
intents:
- goodbye
- greet
- mood_great
- mood_unhappy

responses:
  utter_greet:
  - text: ¡Hola! 😀 Soy un asistente virtual para brindar soporte a las aplicaciones de la Carrera de Computación ESPAM MFL, ¿En qué puedo ayudarte hoy?
  - text: Buenos días 😁, Soy un asistente virtual para brindar soporte a las aplicaciones de la Carrera de Computación ESPAM MFL, ¿cómo puedo ser de ayuda?

actions:
```

```
- action_ask_weather
- action_show_time
- action_default_fallback
- action_process_input
```

- **Creación y mantenimiento de acciones del asistente:**

  En principio, el módulo de comprensión de lenguaje natural de Rasa no es capaz de cumplir con métodos complejos de consulta o de respuestas ligeramente más elaboradas, de hecho, su objetivo principal fue servir al entrenamiento del modelo, sin embargo, también existe la posibilidad de programar respuestas simples mediante *“utters”.* Para el presente proyecto se realizaron métodos de respuesta de mayor complejidad, y estos métodos fueron desarrollados en un script de python exclusivo para alojar estas funciones denominadas acciones

  El archivo *actions.py* fue un componente fundamental en el desarrollo de aplicaciones de este asistente virtual utilizando el modelo Rasa NLU, ya que contiene la lógica necesaria para llevar a cabo las acciones en respuesta a las solicitudes de los usuarios.

  Para crear una nueva acción personalizada en Rasa, es necesario buscar el archivo actions.py en la carpeta correspondiente, generalmente denominada "actions". A continuación, se debe crear una clase con el nombre de la acción personalizada, siguiendo el siguiente formato:

```
from typing import Any, Text, Dict, List
from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher

class ActionName(Action):
    def name(self) -> Text:
        # Define el nombre de la acción
        return "action_name"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        # Aquí es donde se define la lógica de la acción
```

```
        dispatcher.utter_message(text="¡Hola! Esta es una
respuesta de prueba de la acción.")
        return []
```

Definición de la lógica de la acción: En el método run de la clase que se acaba de crear, es donde se define la lógica de la acción. En este ejemplo, se ha utilizado el método utter_message del objeto dispatcher para enviar un mensaje al usuario. Se puede agregar cualquier lógica que necesite este método, como realizar consultas a una base de datos o llamar a una API externa.

Luego se debe agregar la acción en el archivo stories.yml: Por último, se agrega la acción en el archivo stories.yml para que el modelo de Rasa NLU sepa cuándo debe ejecutar la acción. Se puede agregar la acción en una historia existente o crear una nueva historia.

```
- story: example_story
  steps:
  - intent: greet
  - action: action_name
```

Por último, se debe agregar el nombre de la acción dentro del archivo domain:

```
actions:
- action_name
```

3.4. MODELADO

Para iniciar un proyecto en el framework Rasa NLU, es necesario realizar varias configuraciones. En primer lugar, se requiere tener instalada una versión de Python 3.7 o superior. Para verificar la versión de Python instalada, se puede ejecutar el comando "pip --version" o "python --version" en la terminal. Si se requiere, se puede actualizar la versión de Pip a la más reciente mediante el comando "pip install -U pip".

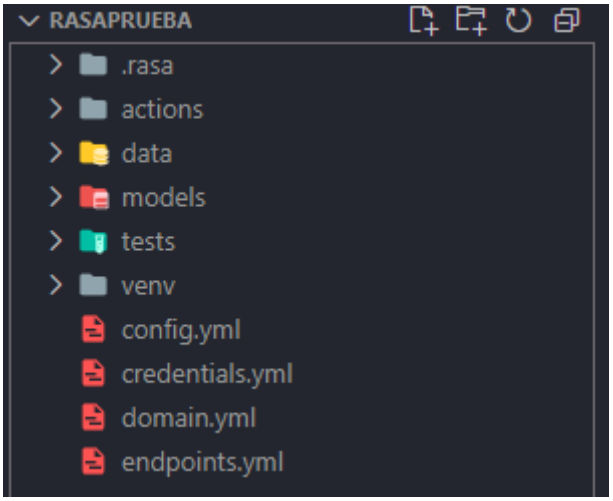
Una vez realizadas estas configuraciones, se procede a instalar el entorno virtual. Es recomendable utilizar un entorno virtual de Python para evitar conflictos y mantener un entorno de desarrollo limpio y aislado para el proyecto de Rasa. Un entorno virtual es una copia aislada del intérprete de Python, lo que permite instalar paquetes de Python y sus dependencias de forma separada para cada proyecto. De esta manera, las dependencias y paquetes instalados para el proyecto de Rasa no entrarán en conflicto con otras versiones de paquetes instalados en el sistema. Para instalar el entorno virtual, se debe ejecutar el siguiente comando en la terminal: "python -m venv ./venv".

Una vez creado el entorno virtual, se debe activarlo con el comando ".\venv\Scripts\activate". Luego, se procede a instalar Rasa y todas sus dependencias mediante el comando "rasa init". Este comando crea automáticamente una estructura de directorios y archivos que incluye un archivo de configuración de Rasa, una plantilla de archivo de entrenamiento de Rasa NLU, una plantilla de archivo de entrenamiento de Rasa Core, y otros archivos necesarios para el proyecto.

Una vez instalado Rasa y sus dependencias, se debe buscar la ruta en la que se ha guardado la carpeta de Rasa con el proyecto iniciado y abrirla en el editor de código de preferencia. En este caso, para el presente trabajo de titulación, se utilizó Visual Studio Code (VSC).

Al abrir el proyecto en VSC, se observó que al ejecutar el comando "rasa init", se crean automáticamente carpetas y archivos predeterminados. Estos archivos y carpetas se pueden apreciar en la imagen que se muestra a continuación:

Figura 1: Estructura básica de un proyecto de Rasa.



Fuente: Los Autores.

Cada una de estas carpetas y archivos tiene una importante funcionalidad en el proyecto:

- **actions:** Un directorio para almacenar los archivos de acción personalizados, que se utilizan para implementar la lógica personalizada de la conversación.
- **data:** Un directorio para almacenar los archivos de entrenamiento y validación para el modelo de Rasa.
- **nlu.yml:** Un archivo que contiene ejemplos de oraciones etiquetadas con intenciones y entidades para que Rasa pueda aprender a identificarlas.
- **stories.yml:** Un archivo que contiene historias de conversación, que describen cómo un usuario interactúa con el chatbot en diferentes escenarios.
- **rules.yml:** El archivo de reglas se llama "rules.md" y se utiliza para definir reglas de conversación más simples y directas, en contraposición a las historias de conversación más complejas que se definen en el archivo stories.md.
- **models:** Un directorio donde se guardarán los modelos entrenados de Rasa.
- **tests:** Un directorio para almacenar pruebas automatizadas.
- **config.yml:** Un archivo de configuración que contiene las opciones de configuración para el modelo de Rasa, como el pipeline del modelo y las políticas de entrenamiento

- **credentials.yml:** Un archivo donde se pueden almacenar credenciales y configuraciones para servicios externos que el bot puede necesitar, como un proveedor de autenticación o una plataforma de chat.
- **domain.yml:** Un archivo que define la estructura de la conversación, incluyendo las intenciones, entidades y acciones disponibles para el bot.
- **endpoints.yml:** Un archivo que contiene la configuración de los endpoints para los servicios externos que el bot puede necesitar, como un servidor de acción personalizado o una plataforma de chat.

Se exploró una de las ventajas ofrecidas por Rasa NLU que consistía en su capacidad para ser utilizado como servicio en la creación de chatbots, lo que permitió la gestión integral de todas las funcionalidades del entorno. En este sentido, se aprovecharon funcionalidades específicas de esta herramienta:

- **Gestión de pipelines de configuración de las políticas y ejemplos del entrenamiento del modelo:** Generalmente cuando se habla de pipelines, se hace referencia a notebooks que, de acuerdo con un determinado proyecto, realizan una tarea específica de forma automática. Estos archivos tienen el formato *yml* que contienen definiciones clave-valor, cuya lógica de ejecución es progresiva, es decir, los comandos son ejecutados en orden, línea por línea. En este caso de uso, todos los archivos que fueron considerados pipelines fueron procesados por el servicio de Rasa NLU, que además fue consumido desde un endpoint específico:

  ```
  rasa:
    url: "http://localhost:5002/api"
  ```

  Esto quiere decir que el servicio con las funciones del asistente deberá consumir el framework de Rasa NLU como primer requisito, antes de que la interfaz web conversacional proceda a utilizar las funcionalidades del asistente como tal.

  El pipeline encargado de la configuración del entrenamiento se denomina config.yml, y su estructura se desglosa de la siguiente manera: **Idioma de la configuración,** donde se especifica según el estándar ISO 639-1 el lenguaje que se va a utilizar como intérprete de lenguaje natural

específico, en este caso, español “*es*”. **Identificación de modelos,** este apartado se ubica directamente después de definir el lenguaje, se definen textualmente los modelos o fases de entrenamiento a las que el asistente será sometido cada vez que se pretenda cumplir una tarea de entrenamiento, además; se pueden customizar parámetros específicos por cada modelo, y por último, **las políticas,** que funcionan como restricciones sobre las reglas y otros métodos de aprendizaje de lenguaje natural que regulan el entrenamiento del modelo. A continuación, se muestra la disposición del pipeline de configuración de Rasa NLU:

Figura 2: Estructura del pipeline de configuración.

```yaml
language: es

pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
    constrain_similarities: true
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
    constrain_similarities: true
  - name: FallbackClassifier
    threshold: 0.85
    ambiguity_threshold: 0.1
      You, now • Uncommitted changes
policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
    max_history: 5
    epochs: 100
  - name: RulePolicy
```

Fuente: Los Autores.

Como se puede observar en la *Figura 2*, existen tres niveles visibles de los cuales ya se habló previamente, inicialmente, se define el lenguaje de forma sencilla como una definición simple de tipo “clave-valor”, seguido por el *pipeline*; mediante la palabra clave “name” Rasa como framework es capaz de identificar textualmente la técnica o método de comprensión de lenguaje natural, a su vez, cada técnica se puede parametrizar de acuerdo con las necesidades de comprensión y aprendizaje del modelo. De forma general, se implementaron métodos para tres etapas de la comprensión de lenguaje natural del asistente:

- **Tokenizador:** Esta herramienta se encargó de dividir el texto en unidades más pequeñas llamadas tokens. Estos tokens pueden ser palabras individuales, signos de puntuación o incluso frases completas, dependiendo de cómo se configure el mismo. Esto ayuda a facilitar el procesamiento del texto por el modelo.
- **Vectorizador de características:** Consecutivamente se convirtieron los

tokens en números (vectores) legibles para el modelo, y posteriormente utilizar para hacer predicciones. Por ejemplo, cada palabra en un texto se puede convertir en un vector que representa su significado, y estos vectores se pueden combinar para representar todo el texto, esto para comprender y comparar diferentes textos utilizando los mismos criterios.

- **Clasificador:** Finalmente, utiliza los vectores generados por el *featurizer* para hacer predicciones sobre el texto. En este caso los clasificadores fueron utilizados para determinar la intención del y posteriormente activar una determinada acción o respuesta predeterminada. Un clasificador permite que el modelo tome decisiones basadas en el significado del texto, y no solo en su apariencia superficial.

Se trabajaron con distintos componentes en virtud de aumentar la precisión de la clasificación de intenciones y entidades, los mismos que se describen a continuación:

Tabla 3: Componentes utilizados en el entrenamiento del modelo.

Componente	**Descripción**
WhitespaceTokenizer	Divide el texto en tokens utilizando espacios en blanco como separadores, segmenta de forma rápida y sencilla el texto en unidades significativas.
RegexFeaturizer	Extrae características del texto utilizando expresiones regulares, identifica patrones específicos y genera características relevantes para clasficar entidades e intenciones.
LexicalSyntaticFeaturizer	Genera características léxicas y sintácticas a partir del texto, captura información relacionada con la estructura y el significado del texto.

CountVectorsFeaturizer	Convierte el texto en una representación numérica mediante el conteo de palabras y la creación de características basadas en el recuento de palabras.
EntitySynonymMapper	Garantiza que los diferentes términos utilizados para referirse a la misma entidad sean reconocidos y tratados de manera consistente.
ResponseSelector	Utiliza características lingüísticas y contextuales para clasificar la entrada del usuario y determinar la mejor respuesta a mostrar.
DIETClassifier	Identifica la intención detrás de la entrada del usuario y extrae las entidades relevantes presentes en el texto.

Fuente: Los Autores.

Con respecto a la parametrización de los componentes:

- Se utilizó `"char_wb"` como tokenizador del texto, ya que se necesitaba un análisis más detallado a nivel de caracteres y se deseaba capturar patrones de caracteres específicos tratándose de un idioma ortográficamente complejo como lo es el español, estas características podían influir directamente en la clasificación y extracción de entidades. “Wb” hace referencia a límite de palabras, esto indica que la división o tokenización fue realizada por límites de n gramas, que a su vez se refieren a una secuencia n caracteres consecutivos dentro de una palabra.
- Los parámetros `"min_ngram"` y `"max_ngram"` fueron planteados como 1 y 4 respectivamente, formando n gramas desde un carácter hasta máximo cuatro caracteres.

- "epochs" se refiere al número de veces que un modelo iteró sobre el conjunto de entrenamiento durante el propio entrenamiento del modelo. En este caso, se ubicó 100 veces porque es un valor recomendado para aprender patrones más complejos y ajustarse mejor a los datos, hay que tomar también, que en la práctica se obtuvieron resultados con un F1 Score superior a 0.9. durante la primera iteración de la validación del modelo.
- Con respecto a "constrain_similarities", se estableció en "True" para aplicar una restricción en la que las características de entrada y las características de las etiquetas (intenciones y entidades) fuesen más similares entre sí, lo que ayudó a mejorar la calidad de las predicciones del modelo.
- Finalmente se tuvo que agregar una fase que se hiciera cargo de las ocasiones en las que el asistente no pudiese clasificar el input en ninguna de las intenciones definidas en el bloque de comprensión de lenguaje natural, esto se lo pudo programar y clasficar como una intención por defecto llamada "FallbackClassifier", a su vez tiene dos parámetros definidos "threshold" y "ambiguity_threshold" que fueron utilizados para reaccionar ante la confianza de la clasificación de una intención, así es como se propusieron los valores de 0.85 y 0.1 respectivamente. El 0,85 del "threshold" significa que si la confianza de la intención más probable está por debajo de este umbral (es decir, menos del 85%), se activa el *FallbackClassifier,* por otro lado, el 0.1 del "ambiguity_threshold" hace referencia a que si la diferencia de confianza entre la intención más probable y la siguiente intención más probable es menor que el 10%, se considera que hay ambigüedad y se activa el *FallbackClassifier*.

- **Gestión de pipelines de liberación y acceso a los servicios del asistente:**

  Estos pipelines se encargaron de definir la comunicación entre el asistente virtual y sus funcionalidades internas, con acciones de servicios/aplicaciones externas.

En primer lugar, se encuentra credentials.yml este es un archivo de configuración de Rasa NLU que se utiliza para almacenar las credenciales de los servicios de terceros que se integran con el bot. Por ejemplo, si el bot utilizó una API de terceros para obtener información, el archivo credentials.yml es utilizado para almacenar las credenciales necesarias para acceder a esa API específica.
Aquí hay un ejemplo de cómo del contenido de un archivo credentials.yml:

```
rasa:
  url: "http://localhost:5002/api"
  token: "rasa_token"
```

En este ejemplo, las credenciales fueron definidas como una lista de claves y valores, donde el nombre de la clave indica el servicio que se está utilizando, y el valor es un objeto que contiene las credenciales necesarias.

Es importante tener en cuenta que el formato y los detalles específicos de las credenciales dependen del servicio que se esté utilizando, por lo que es necesario revisar la documentación del servicio para conocer los detalles específicos.

El proceso de conexión de servicios se completa con *endpoints.yml.* En este pipeline se hace referencia a la URL del servidor open source de Rasa dedicado para recibir mensajes del usuario, que se puede observar a continuación

```
rest:
  url: "http://localhost:500x/webhooks/rest/webhook"
```

Asimismo, el endpoint de las actions, que como ya se explicó previamente, están separadas del módulo principal de Rasa NLU.

```
action_endpoint:
  url: "http://localhost:50xx/webhook"
```

Se agregó también un script en el que se obtiene y se guarda el último modelo

entrenado en rasa para luego crear una instancia de la aplicación en *Flask* que administra las respuestas a las llamadas del servicio del asistente virtual.

- **Interacción directa con el modelo NLU**
  El comando *Rasa interactive* es una funcionalidad proporcionada por Rasa NLU que permite interactuar con un chatbot en tiempo real. Cuando se ejecutó el respectivo comando, se inició un servidor local donde se podía mantener conversaciones con el asistente virtual y realizar anotaciones en tiempo real para mejorar el modelo de lenguaje subyacente.

Cuando se inició el modo interactivo, el chatbot respondió a las entradas como lo haría normalmente, pero en esta ocasión, tuvo la capacidad de proporcionar retroalimentación al modelo anotando manualmente las intenciones y entidades en el texto de entrada como se puede observar en la siguiente Figura.

Figura 3: Rasa interactive (aprendizaje por refuerzo).

Fuente: Los Autores.

Por ejemplo, si se ingresa el input "Hola", el modelo predice que esto corresponde a la intención de "saludo" ("greet") con una métrica de predicción de 1.00. Por esta razón, el modelo responde utilizando uno de los saludos predefinidos.

El propósito principal de utilizar el modo interactivo es recopilar datos de entrenamiento para mejorar el modelo de lenguaje. Se pudo iterar en múltiples conversaciones, ajustar las anotaciones según sea necesario y guardar los datos anotados para utilizarlos en el entrenamiento posterior del modelo.

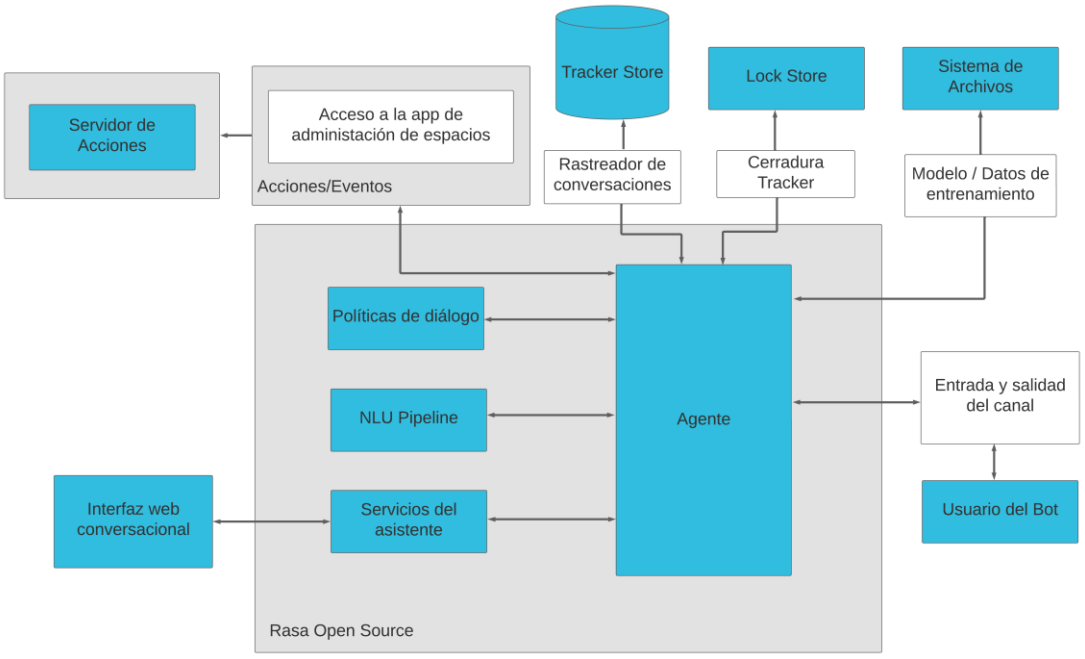
Una vez que se haya terminado de interactuar con el asistente virtual en el modo interactivo, se puede detener el servidor y utilizar los datos anotados recopilados

durante la sesión para entrenar o mejorar el modelo de Rasa NLU. Esto ayudará a que el asistente virtual sea más preciso y capaz de comprender mejor las intenciones y entidades en futuras interacciones.

Para permitir que el asistente se conectara con otras aplicaciones o interfaces conversacionales, fue necesario crear un endpoint que facilitara dicha interacción. En ese sentido, el endpoint /webhook desempeñó un papel crucial al establecer la comunicación entre el servidor de Rasa y otros sistemas externos. Esta ruta específica se utilizaba para recibir solicitudes y enviar respuestas relacionadas con las acciones personalizadas de Rasa Core.

En el proyecto, se realizó la configuración de la ruta /webhook utilizando el microframework Flask. Esta configuración se detalló en la sección de despliegue, donde se explicó cómo se implementó esta funcionalidad específica. El objetivo era asegurar que el asistente pudiera recibir las solicitudes y enviar las respuestas adecuadas a través del endpoint /webhook, permitiendo así una interacción fluida con otras aplicaciones o interfaces conversacionales.

Figura 4: Rasa NLU (Modelo de servicio).



Fuente: Los Autores.

Para permitir que el asistente respondiera a preguntas específicas relacionadas con las aplicaciones de la UDIV de infraestructura, se crearon acciones

personalizadas dentro del proyecto de Rasa. Estas acciones funcionaron como intenciones adicionales en el modelo. Cuando el usuario ingresaba un prompt o input, el modelo utilizaba los datos de entrenamiento para identificar si el input del usuario contenía alguna entidad que se relacionara con una de estas acciones personalizadas, si esto se cumplía, se ejecutaba una función correspondiente a la acción personalizada.

Dentro de estas acciones personalizadas, se realizaron llamadas a las APIs de la aplicación de “Administración de espacios” de la UDIV para iniciar las pruebas de integración. Dependiendo de la solicitud del usuario, se obtenían los filtros necesarios para mostrar los datos específicos requeridos. Esto permitía que el asistente proporcionara respuestas relevantes y precisas al usuario, utilizando la información obtenida a través de las APIs de la UDIV. De esta manera, se logró una interacción efectiva entre el asistente y las aplicaciones de la UDIV de infraestructura, brindando a los usuarios la información que solicitaban de manera específica.

Figura 5: Llamado a la API de "Gestión de espacio".

```
# Consultar la API para obtener la respuesta
r = requests.get('http://localhost:57032/api/reserva')
response = r.json()
```

Fuente: Los Autores.

Es importante realizar esta distinción inicial del proceso de interacción del usuario/desarrollador con el modelo NLU, porque es la forma más directa con la que se puede mantener una comunicación bidireccional e intervenir en directo, con el aprendizaje del modelo. Es decir, el modelo es capaz de aprender y discernir las intenciones de las entradas de los usuarios de forma automática a partir de los ejemplos definidos en su núcleo NLU, dominio, y reglas específicas, pero también, puede aprender por “refuerzo”, llamado así porque se lo consideró como un proceso ligeramente manual, debido a que se necesita de la consola, como requerimiento indispensable para entablar una conversación con algún modelo específico que se pretenda evaluar y reforzar (solo en caso de ser estrictamente necesario), de hecho, se recomienda mantener el enfoque tradicional de entrenamiento, que se basa en la definición de ejemplos.

Adicionalmente, como se puede observar en la *Figura 3*, se aprovechó el estado de interacción bidireccional, y se lo tomó como un proceso inicial de verificación del análisis de compresión de los inputs del usuario. Se puede observar también que se detallan las características que el modelo fue capaz de extraer a partir de la entrada. En base a un estándar propuesto por Rasa, se definen a continuación las fases por las que transcurre el proceso de comunicación usuario (en este caso el desarrollador) y un modelo de NLU específico, que principalmente ayudó a revisar métricas relativamente generales del desempeño del modelo, además de comprender el modus operandi del servicio de comunicación del framework de Rasa. Así es como el aprendizaje por refuerzo que implementa el framework obliga a que esta comunicación se divida en dos fases, una condicional, y otra de respuesta.

Fase condicional

Tiempo de espera de atención a la validación de la respuesta del modelo por parte del usuario compuesto por:

- **action_listen:** Antes de que el input fuese procesado por el modelo, el mismo inicia una actividad de escucha denominada “action_listen”, esta acción no debe ser programada en el script dedicado a las acciones porque ya viene definida por defecto, lo único que se tomó en cuenta es que esta acción funciona como una advertencia que indicaba dos cosas:
    1) El modelo había terminado de procesar un input previo.
    2) A su vez, el modelo estaba listo para “escuchar” una nueva solicitud entrante.

  En el caso de que se haga referencia al input origen, el modelo, al encontrarse en su estado inicial, claramente se mantuvo en modo “escucha”.
- **Input:** En esta ocasión, el autor del input, en calidad de usuario final o desarrollador, el valor de la entrada se especifica textualmente en la consola, en consecuencia, el modelo clasifica la intención de acuerdo a su entrenamiento. En el ejemplo, el input es “Hola”, y el modelo clasifica la intención como `intent: greet 1.00`, este valor hace referencia a la confianza con la que el modelo tomó la decisión de clasificación. Además de definir la intención, el modelo indicó el número de entidades que

encontró, y finalmente preguntó si lo que estaba definiendo era lo que se estaba esperando, o, por defecto, el mejor de los casos de respuesta.

Fase de respuesta:

Este apartado tiene dos versiones, la del usuario contestando a la verificación de la respuesta del modelo, y la del modelo preguntándole al usuario si puede volver a escuchar las peticiones de inputs, también definido previamente como "action_listen**"**. Para el primer caso, el modelo devuelve la respuesta predeterminada a la intención clasificada, en el ejemplo previo, la intención fue *greet* (saludo), y el modelo, devolvió una respuesta de saludo acorde a la conversación. El otro caso se activa únicamente después de cada respuesta del modelo.

Este proceso de comunicación es muy simple, y comparte una relación muy estrecha con los elementos de la comunicación, tal y como se representa en la *Tabla 4,* se utilizó esta analogía recortada, clásica del lenguaje, para detallar el flujo de interacción establecido por Rasa NLU.

Tabla 4: Interpretación de la comunicación con el modelo NLU de Rasa.

Emisor	Usuario
Receptor	Modelo NLU
Mensaje	Input o mensaje de entrada del usuario
Canal	Texto, por medio de una consola virtual
Código	El input del usuario que ha sido pre procesado por un tokenizador, vectorizador y clasificador de textos

Fuente: Los Autores.

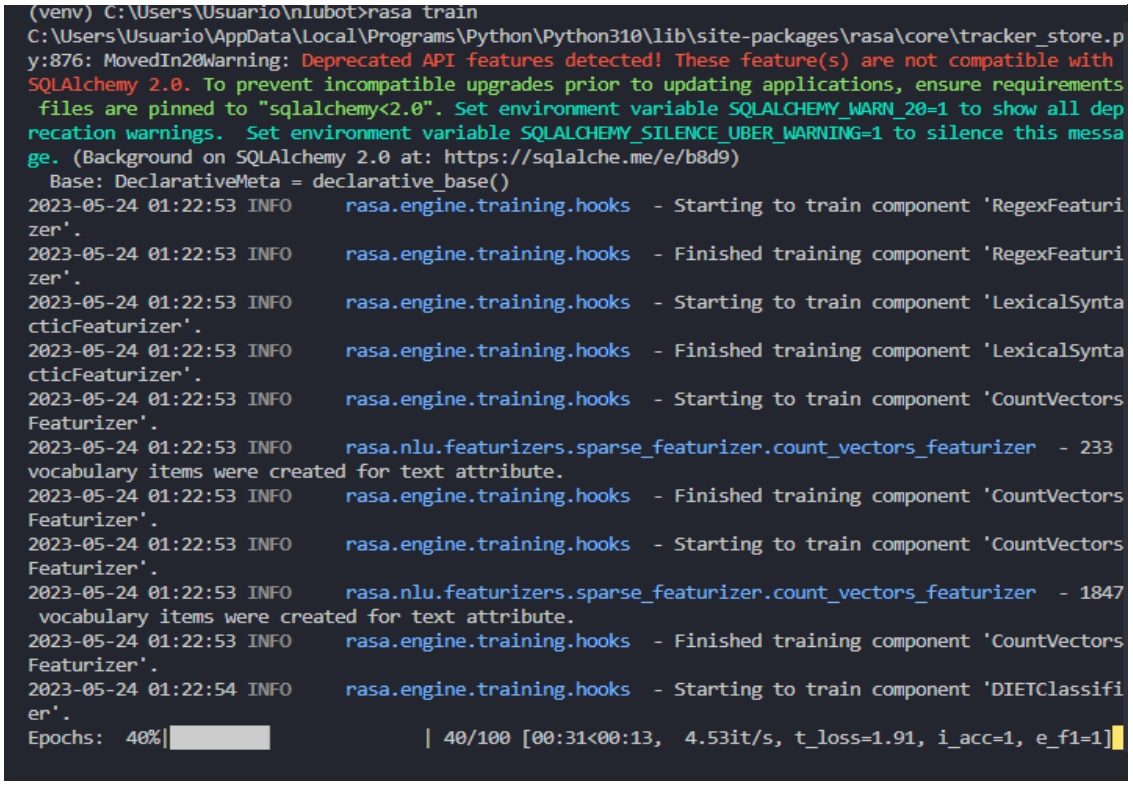
3.5. EVALUACIÓN

En el contexto de Rasa NLU, existen dos comandos principales relacionados con el entrenamiento de modelos: "rasa train" y "rasa train nlu". A continuación, se proporciona información sobre la diferencia y el propósito de cada uno de estos comandos.

"Rasa Train"

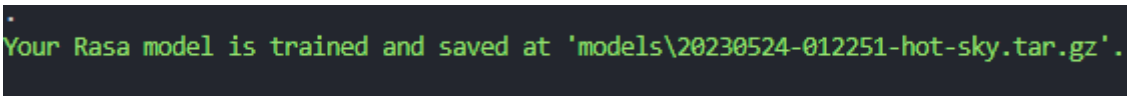
Se utiliza para entrenar tanto el modelo NLU como el modelo de diálogo. Durante el proceso de entrenamiento, se procesan los datos de entrenamiento que incluyen ejemplos etiquetados con intenciones y entidades. El sistema utiliza algoritmos de aprendizaje automático para ajustar los parámetros de ambos modelos y mejorar su capacidad para comprender y responder a las interacciones de los usuarios. A continuación, se muestra un ejemplo de cómo sería el proceso de entrenamiento con el comando "rasa train"

Figura 6: Entrenamiento del modelo por defecto.



Fuente: Los Autores.

Figura 7: Ruta del modelo entrenado por defecto.



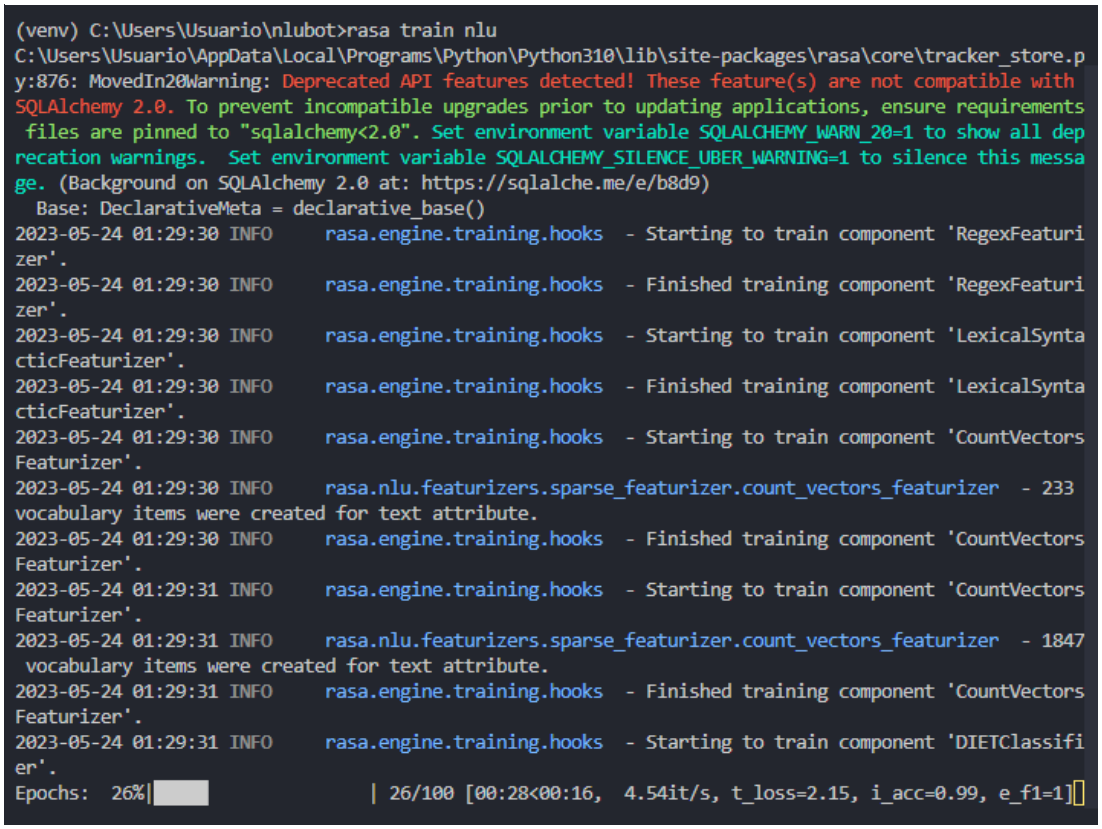
Fuente: Los Autores.

El entrenamiento del modelo NLU implica tomar los datos de entrenamiento proporcionados, como ejemplos de intenciones y entidades, y construir un modelo capaz de comprender y clasificar las entradas del usuario. El modelo de diálogo, por otro lado, se entrena utilizando ejemplos de diálogos etiquetados para aprender las políticas de conversación y generar respuestas adecuadas en función del contexto.

"Rasa Train Nlu"

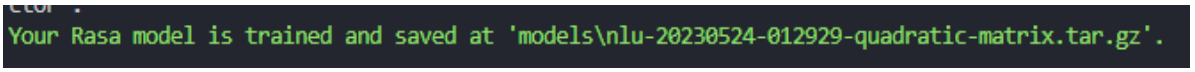
El comando "rasa train nlu" se utiliza específicamente para entrenar únicamente el modelo NLU en Rasa NLU. Este comando se centra en el procesamiento del lenguaje natural y se utiliza cuando se desea enfocar el entrenamiento exclusivamente en mejorar la comprensión de las intenciones y entidades en las interacciones con los usuarios. A continuación, se muestra un ejemplo de cómo sería el proceso de entrenamiento con el comando “rasa train nlu”

Figura 9: Entrenamiento de un modelo NLU (rasa train nlu).



Fuente: Los Autores.

Figura 8: Ruta del modelo NLU.



Fuente: Los Autores

Como se puede ver en la *Figura 9* Rasa guardará el nuevo modelo entrenado en una fila, y será de fácil acceso cada vez que se inicie una instancia de servicio del asistente virtual.

Cuando se ejecuta el comando "rasa train nlu", se procesan los datos de entrenamiento proporcionados. Estos datos contienen ejemplos de oraciones etiquetadas con intenciones y entidades. El sistema utiliza estos ejemplos para aprender patrones y construir un modelo de lenguaje natural capaz de comprender y clasificar las intenciones y entidades en las interacciones del usuario.

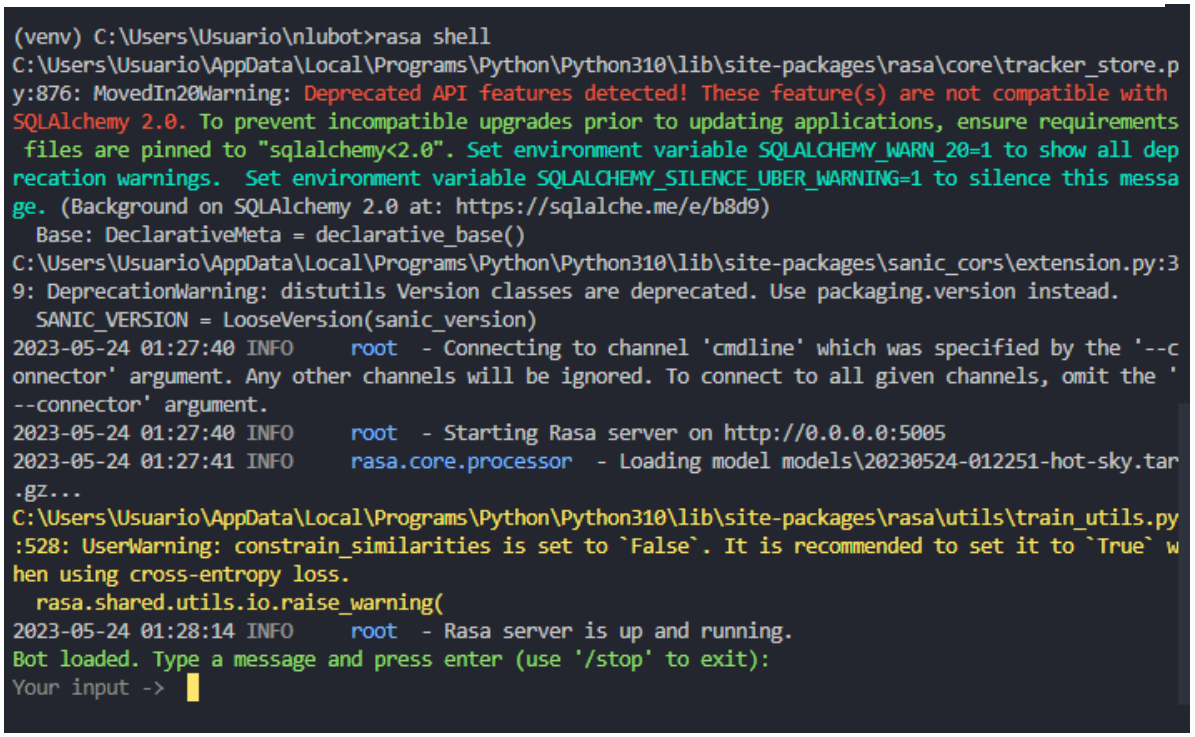
“Rasa Shell”:

El comando "rasa shell" en Rasa se utiliza para iniciar una sesión interactiva en la que se puede probar y depurar el asistente virtual. Al ejecutar el comando "rasa shell", se abre una ventana de consola o terminal donde puedes ingresar mensajes como si fueras un usuario interactuando con el sistema.

Durante la ejecución de un modelo entrenado con el comando “rasa train”, si se usa el comando "rasa shell", se emplean tanto el modelo NLU como el modelo de diálogo para generar respuestas de manera interactiva. El modelo NLU se encarga de analizar las intenciones y entidades presentes en los mensajes de entrada del usuario, mientras que el modelo de diálogo utiliza políticas de conversación definidas para producir respuestas coherentes y contextualmente relevantes.

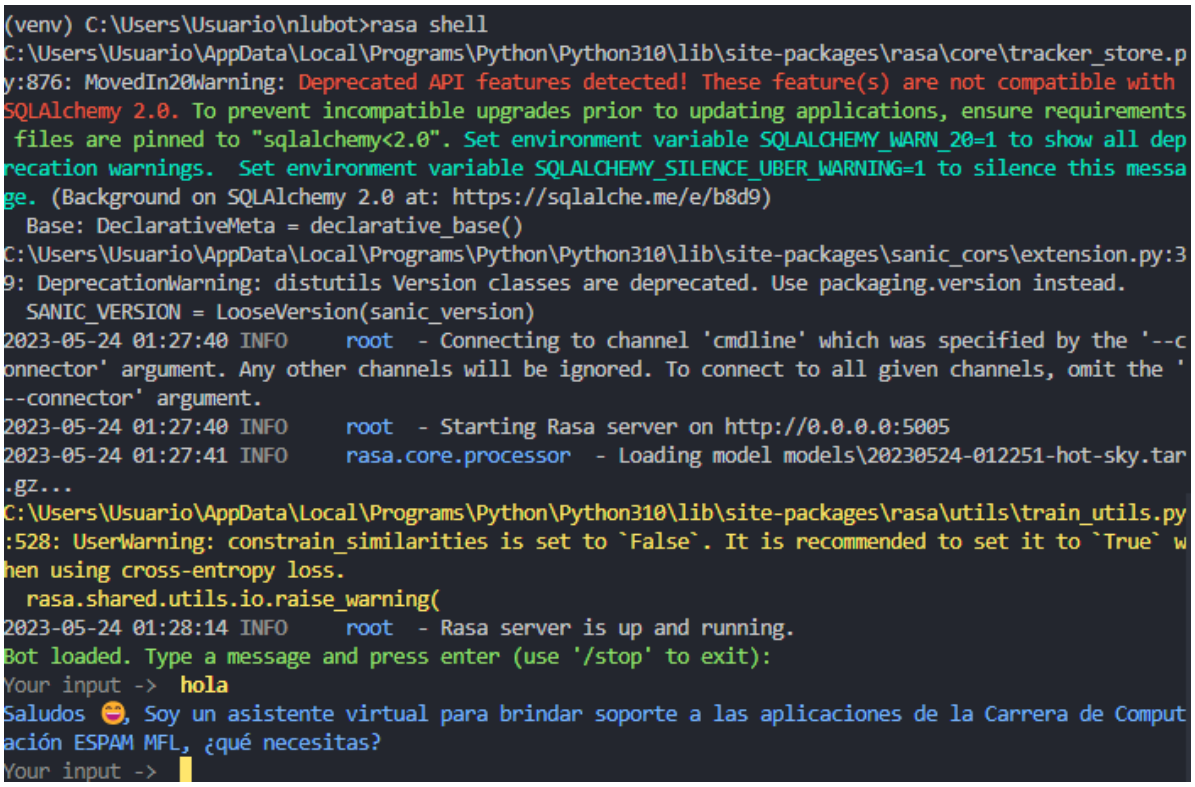
Durante la sesión de shell, el usuario puede ingresar mensajes de texto y el modelo NLU procesa dichos mensajes para identificar las intenciones subyacentes y las entidades relevantes. A partir de esta información, el modelo de diálogo determinará la respuesta más adecuada para enviar al usuario. De esta manera, se logra mantener una conversación fluida y congruente entre el usuario y el chatbot, Como lo podemos ver en las siguientes imágenes.

Figura 10: Interacción con el Modelo desde la terminal.



Fuente: Los Autores.

Figura 11: Respuesta del modelo desde la terminal.



Fuente: Los Autores

Por otro lado, al ejecutar un modelo entrenado con “rasa train nlu” específicamente mediante el comando "rasa shell", se pone el enfoque exclusivamente en la comprensión del lenguaje natural. En este caso, únicamente se emplea el modelo NLU para analizar los mensajes de entrada del usuario y generar las correspondientes intenciones y entidades.

La sesión de shell con un modelo NLU permite evaluar y probar la capacidad del modelo para comprender y clasificar las intenciones y entidades en tiempo real. A través del ingreso de mensajes de texto, es posible verificar cómo el modelo interpreta y etiqueta la información contenida en ellos, brindando una oportunidad para evaluar su desempeño, por ejemplo, si proporcionamos el siguiente input al modelo “hola”, el modelo evaluará entra entrada y decidirá si la intención es saludar u otra, en base a esto el modelo proporcionará con la interpretación que realizó. En el **(Anexo III)** se realizó un informe específico de

entrenamiento y evaluación de un modelo, utilizando el comando *“rasa train nlu”*

3.6. DESPLIEGUE

Se realizaron varios pasos iniciales para desarrollar las interfaces de programación de aplicaciones (API) que permitieran que el asistente virtual pudiera ser consumido por diversas aplicaciones de mensajería. En primer lugar, se utilizó el framework Flask para crear un servidor web que permitiera el consumo del asistente virtual. Para lograr esto, se definió una ruta de API en la aplicación Flask utilizando el decorador `@app.route('/webhook', methods=['POST'])`. Esta ruta se diseñó específicamente para procesar las solicitudes de los usuarios. Al interior de la función webhook(), se obtuvo el mensaje del usuario a través de la solicitud POST utilizando `request.json['message']`. De esta manera, se logró establecer una API funcional para el asistente virtual, que permite su integración con diversas aplicaciones.

Luego, es necesario especificar en el archivo de endpoint el puerto del localhost en el cual se desea que el asistente virtual escuche. La URL que se debe especificar es "`http://localhost:5005/webhooks/rest/webhook`", lo cual indica que el servidor de Rasa Open Source se encuentra en escucha en la dirección local ("localhost") y en el puerto 5005.

```
rest:
  # The URL of your Rasa Open Source server for
receiving messages from the user
  url: "http://localhost:5005/webhooks/rest/webhook"
```

Una vez se hayan reunido todos estos elementos, será posible ejecutar el Asistente virtual y permitir su consumo por parte de cualquier aplicación de mensajería. Para llevar a cabo una prueba de funcionamiento, se utilizará NGROK, una herramienta gratuita que permite exponer un entorno local en Internet. Para realizar esto, se debe descargar NGROK desde su página oficial y proceder con la instalación. Una vez instalado, se accede a la terminal y se ejecuta el siguiente comando: "ngrok http 5005". Este comando permitirá

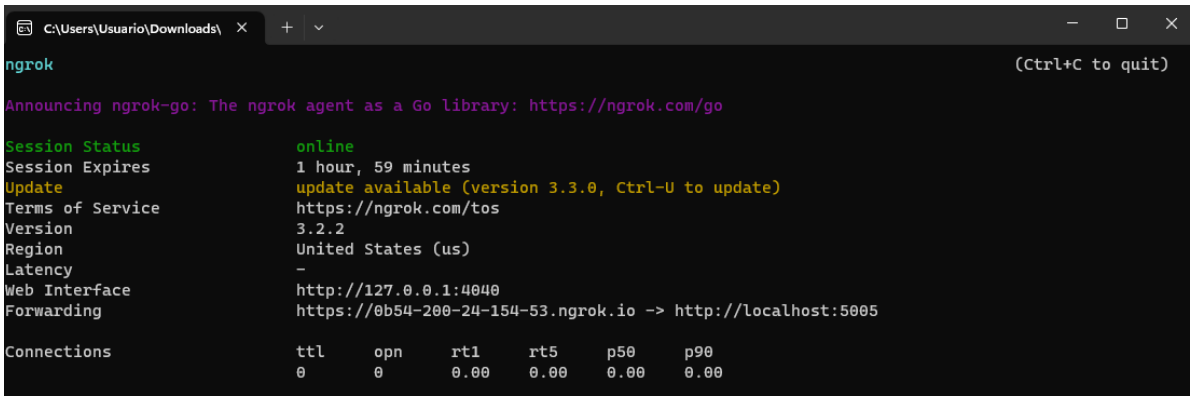
establecer una conexión y exponer el servidor local en el puerto 5005.

Figura 12: Terminal de ngrok con el comando "ngrok htttp 5005".

Fuente: Los Autores.

Con esta configuración, se indica que el puerto 5005 debe ejecutarse en la web. A continuación, en la terminal de Rasa, se deben ejecutar las acciones utilizando el comando "`rasa run actions`". Luego, se deben iniciar todos los servicios y APIs de Rasa utilizando el comando "`rasa run --enable-api --cors "*"`".

Figura 13: Servidor corriendo en ngrok.



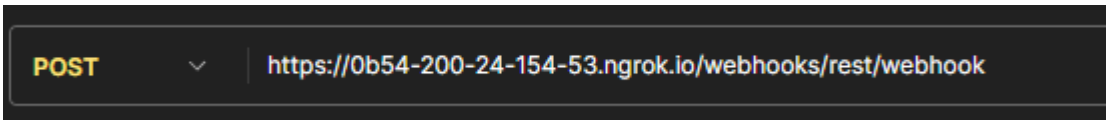
Fuente: Los Autores.

La dirección que se devuelve en "*Forwarding*" marca la integración de los servicios del asistente virtual con la interfaz web conversacional.

Con el propósito de validar el adecuado funcionamiento de la presente API, se utilizó el cliente Postman como herramienta para verificar la correcta respuesta del Asistente a las solicitudes. Se procedió a ejecutar el cliente Postman y, mediante el empleo del URL proporcionado por NGROK, se configuró en Postman utilizando el método "POST". En el URL se especificó la ruta designada para la escucha del asistente virtual, en este caso, se estableció como /webhooks/rest/webhook. Como resultado, la URL final adoptó la siguiente

estructura: https://0b54-200-24-15453.ngrok.io/webhooks/rest/webhook.

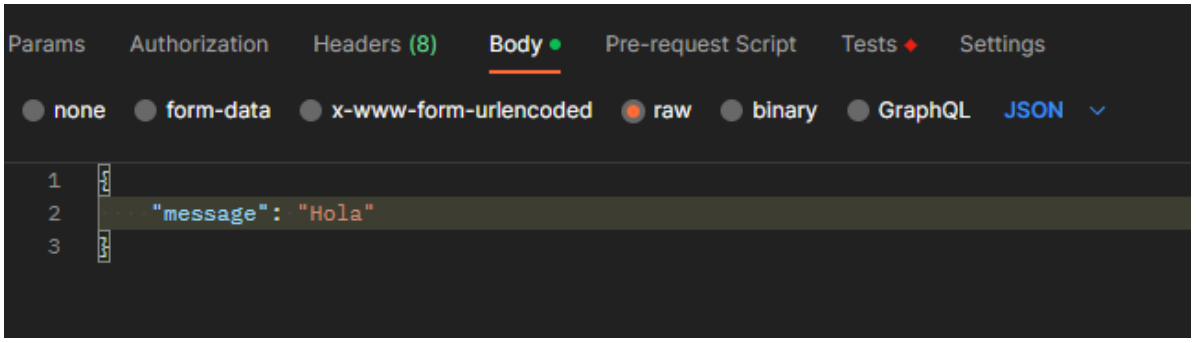
Figura 14: Postman usando método post para probar la APIs.



Fuente: Los Autores.

Sin embargo, es importante tener en cuenta que este método “POST” recibe un mensaje en formato JSON. Para enviar un mensaje al asistente virtual utilizando Postman, se debe configurar la solicitud de la siguiente forma:

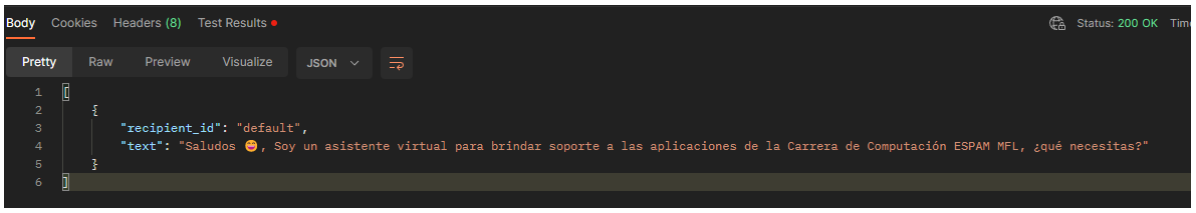
Figura 15: Formato JSON para enviarle input al modelo.



Fuente: Los Autores.

Por último, ya completados estos pasos, se envió una solicitud al asistente virtual en espera de recibir una respuesta. Esto se puede observar en la siguiente figura:

Figura 16: Respuesta del modelo en Postman.



Fuente: Los Autores.

Por otro lado, una vez que se haya completado el proceso de entrenamiento del asistente virtual utilizando e incorporando las intenciones y entidades relevantes, llega el momento de desplegar el asistente virtual lo podemos hacer en diferentes sistemas de mensajería. Rasa NLU ofrece la flexibilidad necesaria para adaptar

y utilizar el asistente virtual en una amplia gama de plataformas de comunicación, como Facebook Messenger, Slack, WhatsApp, Telegram y muchos más.

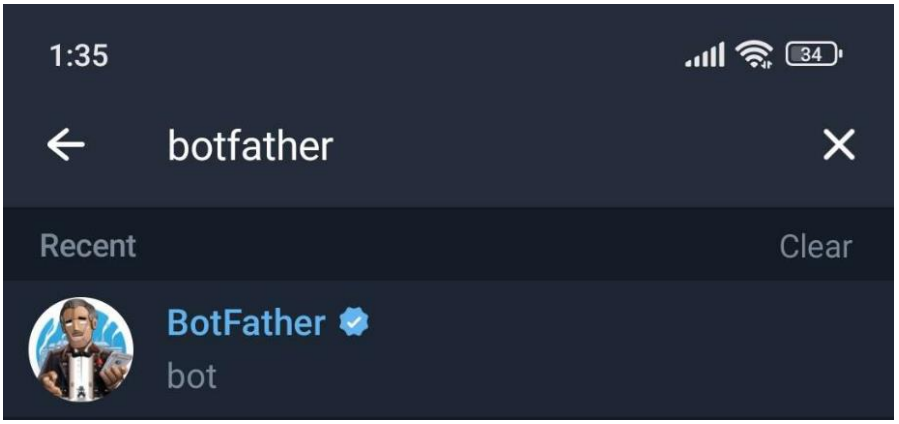
Además de aprovechar las plataformas de mensajería existentes, también es posible implementar el asistente virtual en un sistema de mensajería propio, diseñado específicamente para nuestras necesidades. Esto brinda un mayor control sobre la experiencia del usuario y permite personalizar la interfaz y las funcionalidades según nuestros requerimientos.

Al desplegar el asistente virtual en diferentes sistemas de mensajería o en uno propio, se aprovechó la naturaleza conversacional de Rasa NLU para ofrecer interacciones fluidas y naturales con los usuarios. Mediante la comprensión de las intenciones y entidades de los mensajes, el asistente pudo brindar respuestas relevantes y precisas, así como ejecutar acciones específicas según las necesidades del usuario.

Para el despliegue del asistente virtual, se utilizó la plataforma de mensajería Telegram debido a su facilidad de conexión con Rasa, lo que permitió llevar a cabo el despliegue de manera rápida y eficiente.

Con el propósito de establecer la conexión con Telegram, se requirió obtener ciertos tokens. La primera etapa consistió en acceder a la aplicación de Telegram y localizar al "BotFather". Dicho bot se especializa en la creación y configuración de bots dentro de la plataforma de Telegram.

Figura 17: Buscando al BotFather en Telegram.



Fuente: Los Autores.

Una vez se encontró a BotFather, se dio inicio al proceso de creación del bot mediante el empleo del comando /newbot. Durante esta fase, se requirió asignar un nombre y un nombre de usuario al bot, aspectos de vital importancia para su identificación y futura interacción. Tras finalizar la configuración del bot, BotFather otorgó un token de acceso único para el bot. Dicho token resulta esencial para establecer la comunicación entre el asistente virtual desarrollado en Rasa y la plataforma de Telegram.

Es importante destacar que mantener este token seguro es fundamental para proteger la integridad y privacidad del bot. Por lo tanto, es recomendable almacenar el token de forma segura y no compartirlo públicamente.

Figura 19: Creando un bot en Telegram.

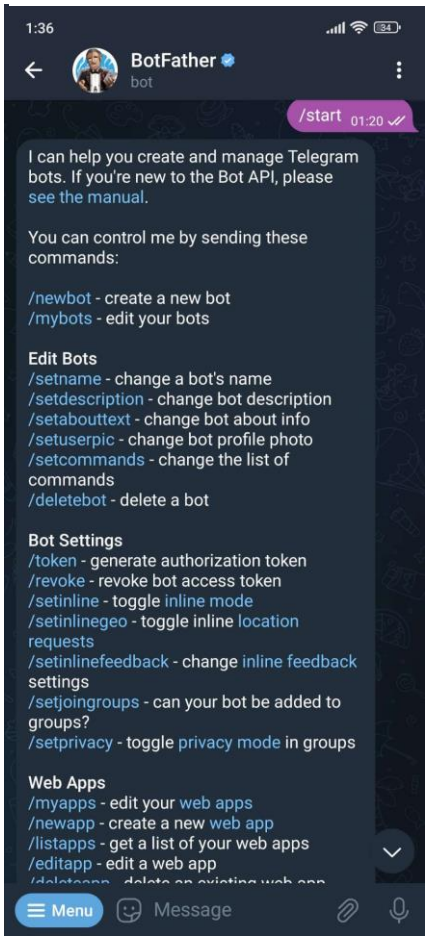
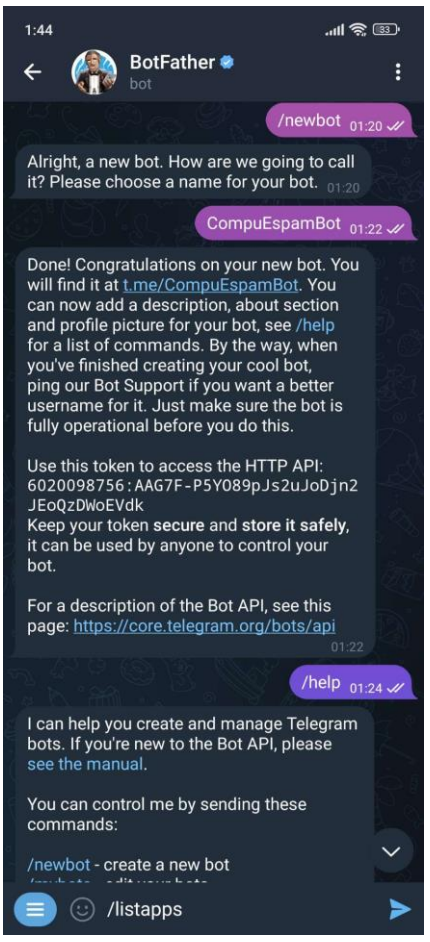


Figura 18: Colocándole username al bot.



Fuente: Los Autores.

Tras la finalización de este procedimiento, el sistema se encontraba preparado para establecer la conexión. En este punto, se procedió a trabajar con el archivo credentials.yml presente en el proyecto, con el objetivo de completar los siguientes datos: access_token, verify y webhook_url. A continuación, se proporcionará una explicación acerca del significado de cada uno de estos datos.

- access_token: Este campo se refiere al token de acceso necesario para autenticar y autorizar las solicitudes entre tu asistente virtual y Telegram. El token de acceso es proporcionado por Telegram al crear un bot a través de BotFather.
- verify: Se utiliza para establecer la autenticación y verificación de las solicitudes recibidas desde Telegram. En este caso no es más que el username que se le dio al bot que se creó en telegram
- webhook_url: Es la dirección URL en la que Telegram enviará las actualizaciones de mensajes a tu asistente virtual. Para poder proporcionarle el URL corremos el proyecto en NGROK.

Todos estos datos se ubicaron en el archivo de credential.yml de la siguiente forma:

```
telegram:
access_token: dummy_token_value
verify: CompuEspamBot
webhook_url:"https://b87a-200-24-154-
53.ngrok.io/webhooks/telegram/webhook"
```

Sin embargo, para garantizar el correcto funcionamiento del asistente virtual, fue necesario activar las acciones mediante el comando "rasa run actions". Además, el modelo debió ser ejecutado de manera distinta a la convencional, utilizando el comando "rasa run --enable-api --cors "*"". Este último comando permitió ejecutar un servidor de API de Rasa con la habilitación de CORS (Cross-Origin Resource Sharing) en todas las rutas.

Tras finalizar todas las etapas mencionadas, el asistente estaba listo para ser probado a través de Telegram. Para ello, bastaba con buscar el nombre del asistente en la plataforma, en este caso "CompuEspamBot", y posteriormente

iniciar una conversación con él. A continuación, se presenta un ejemplo de cómo realizar esta acción:

Figura 20: Conversación con el asistente virtual desde Telegram.

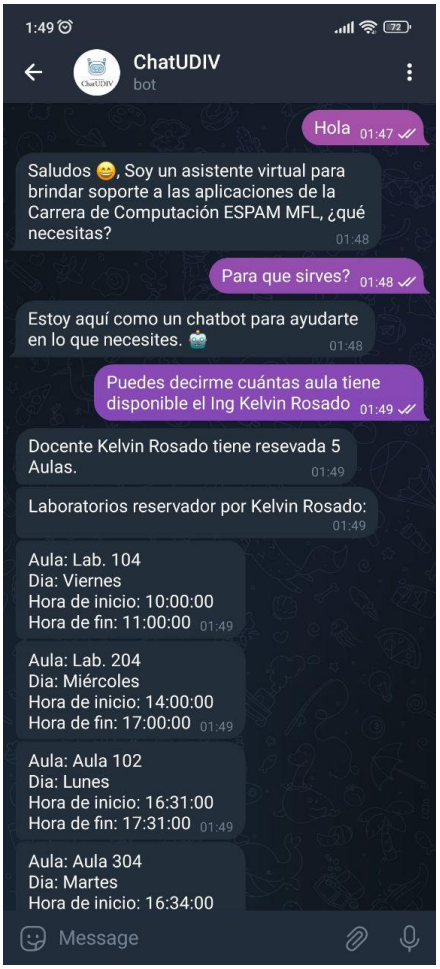
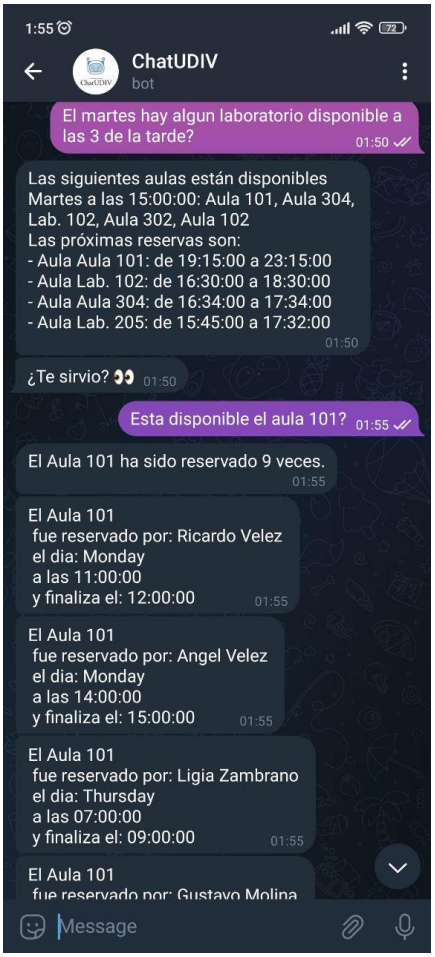


Figura 21: Conversación con el asistente virtual desde Telegram.

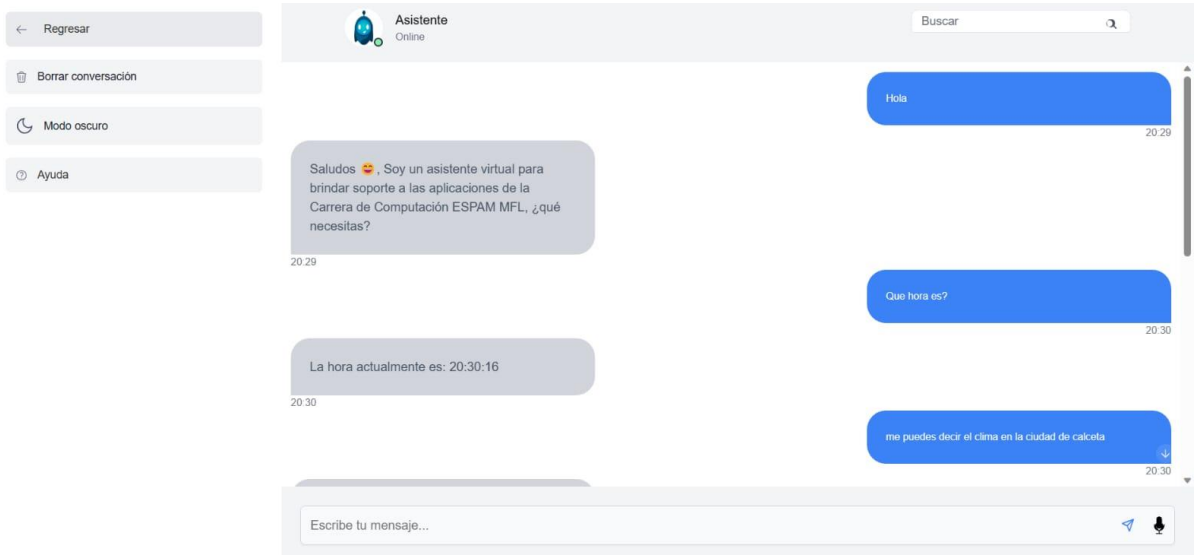


Fuente: Los Autores.

La implementación exitosa del asistente virtual en Telegram demuestra su capacidad de despliegue en diversas plataformas de mensajería. Gracias a este logro, el asistente está ahora disponible en la plataforma y listo para interactuar con los usuarios.

Con una configuración apropiada, una conexión establecida y los tokens de acceso correspondientes, se ha logrado conectar el asistente desarrollado en Rasa NLU con la plataforma de Telegram. Los usuarios tienen la posibilidad de acceder al asistente a través de la aplicación de Telegram y disfrutar de una experiencia conversacional fluida y eficiente.

Figura 22: *Despliegue del asistente virtual con la interfaz web conversacional*



Fuente: Los Autores.

Se puede observar que en la Figura 22, los servicios del asistente virtual fueron consumido por la interfaz web conversacional, la misma que forma parte de otro trabajo de integración curricular.

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Se destacó la importancia de combinar diferentes técnicas de procesamiento de lenguaje natural y llevar a cabo evaluaciones constantes de los inputs del usuario. Estas acciones resultaron clave para mejorar la precisión del modelo, que sufrió una mejora del F1 Score, que incrementó de 0.85 a 0.89 en iteraciones posteriores.
- Asimismo, el análisis exploratorio de datos proporcionó información valiosa sobre la naturaleza de las interacciones con el asistente. Se observó que las entradas de texto del usuario tienden a ser relativamente cortas debido a la arquitectura de preguntas y respuestas del chatbot. Esto permitió identificar áreas de mejora en la comprensión del modelo, lo que a su vez contribuyó a la optimización continua del mismo.
- En términos de la calidad del modelo NLU, se obtuvieron resultados satisfactorios en la predicción de intenciones y extracción de entidades. Con una confianza perfecta, se logró una clasificación correcta de 356 ejemplos de entidades en una simulación de 11 historias. Estos resultados demuestran que el asistente es capaz de interpretar el lenguaje natural de manera precisa y efectiva, cumpliendo con los objetivos establecidos.
- Por último, la implementación de una API con Flask y las pruebas realizadas en Postman, Telegram y la interfaz web conversacional mostraron la plena funcionalidad del servicio en un servidor local. Con tiempos de respuesta promedio por debajo de 1.5 segundos por petición, se validó la capacidad de comprensión y respuesta del asistente virtual, corroborando el éxito del proyecto en su conjunto.

4.2. RECOMENDACIONES

- En caso de aplicar una revisión bibliográfica como técnica de recopilación de información para proyectos que involucren inteligencia artificial, se sugiere dedicar la mayor parte de recursos y esfuerzos a delimitar el tema del trabajo de investigación, asimismo, se aconseja invertir más tiempo en la refinación de las palabras claves a utilizar en la búsqueda de la información bibliográfica para evitar desviarse en técnicas de comprensión de lenguaje natural que no corresponden a las necesidades del proyecto.
- Para mejorar la interacción del chatbot, se aconseja la eventual monitorización de los inputs del usuario mediante la aplicación de umbrales o límites para activar nuevamente el entrenamiento del modelo para identificar áreas de mejora y optimizar el modelo en consecuencia. Además, se propone trabajar en la diversificación de los datos de entrenamiento para mejorar la capacidad del modelo en comprender más contextos. Los autores proponen utilizar un modelo pre entrenado que comprenda el lenguaje natural para después definir inputs de entrenamiento de un caso de uso específico.
- Es importante tener en cuenta que, si bien la predicción de entidades obtuvo un F1-Score de 0.89, existe margen de sobra para mejorar aún más este aspecto. En este sentido, se recomienda dedicar recursos y esfuerzos adicionales para perfeccionar la precisión de la predicción de entidades. Los autores sugieren mantener una atención continua en el desarrollo y refinamiento de la extracción de entidades, con el objetivo de alcanzar niveles aún más altos en todas las métricas.

- En caso de que los servicios de la aplicación presenten adiciones, modificaciones o actualizaciones, se recomienda re documentar las APIs, realizar nuevamente pruebas exhaustivas, explorar nuevas plataformas de demostración y mantener un tiempo de respuesta óptimo. Esto contribuirá a fortalecer la calidad y usabilidad del asistente virtual, permitiendo su despliegue exitoso en diferentes escenarios y entornos de uso.

BIBLIOGRAFÍA

Ait-Mlouk, Addi, y Lili Jiang. 2020. “KBot: A Knowledge Graph Based ChatBot for Natural Language Understanding over Linked Data”. IEEE Access 8:149220–30. doi: 10.1109/ACCESS.2020.3016142.

Carlos, U., de Madrid, I., Galán, V., Tutora, C., & Castro Galán, E. (2015). Aplicación de la Metodología CRISP-DM a un Proyecto de Minería de Datos en el Entorno Universitario.

Cortés, Wilton, y Omar Valverde. 2020. “Analítica de datos no estructurados para dar soporte a la toma de decisiones en el área de comercialización de la empresa Representaciones BATERICAR S.A.C. utilizando la metodología ICAV y plataforma de Microsoft”.

Duncan, Bob, y Aspen Olmsted. 2018. CLOUD COMPUTING 2018 Proceedings of the Ninth International Conference on Cloud Computing, GRIDs, and Virtualization.

ESPAM MFL. (2016). Modelo Educativo Escuela Superior Politécnica Agropecuaria de Manabí “Manuel Félix López”. Editorial Humus (ed.).

ESPAM MFL. (2019). Obtenido de: http://www.espam.edu.ec/recursos/sitio/espam/EstatutoESPAMMFL.pdf

ESPAM MFL. (2022). ESPAM MFL. http://www.espam.edu.ec/web/oferta/grado/computacion.aspx

Higinio, F., Espinoza, R., & Cervantes, R. E. (2021). Revisión Bibliográfica: La Metodología del Aprendizaje basado en la Investigación. Ciencia Latina Revista Científica Multidisciplinar, 5(1), 1079–1093. https://doi.org/10.37811/cl_rcm.v5i1.312

Jiao, Anran (2020). An Intelligent Chatbot System Based on Entity Extraction Using RASA NLU and Neural Network. *Journal of Physics: Conference Series*, *1487*(1). https://doi.org/10.1088/1742-6596/1487/1/012014

Jiménez, Adrián. 2019. “Aplicación para crear chatbots y asistentes virtuales inteligentes”.

Jose, K., & Lakshmi, K. (2018). Joint Slot Filling and Intent Prediction for Natural Language Understanding in Frames Dataset. Research in Computing Applications.

Manjarrés-Betancur, R. A., & Echeverri-Torres, M. M. (2020). Asistente virtual académico utilizando tecnologías cognitivas de procesamiento de lenguaje natural. Revista Politécnica, 16(31), 85–95. https://doi.org/10.33571/rpolitec.v16n31a7.

Namazifar, Mahdi, Alexandros Papangelis, Gokhan Tur, y Dilek Hakkani-Tur. 2021. “LANGUAGE MODEL IS ALL YOU NEED: NATURAL LANGUAGE UNDERSTANDING AS QUESTION ANSWERING”. 7788–92.

Ramesh, Kiran. 2017. “A Survey of Design Techniques for Conversational Agents”. 336–50. doi: 10.1007/978-981-10-6544-6.

Ramesh, Vignav, y Anton Kolonin. 2020. “Interpretable Natural Language Segmentation Based on Link Grammar”. Proceedings - 2020 Science and Artificial Intelligence Conference, S.A.I.ence 2020 25–32. doi: 10.1109/S.A.I.ence50533.2020.9303220.

Rychalska, Barbara, Helena Glabska, y Anna Wroblewska. 2018. “Multi-Intent Hierarchical Natural Language Understanding for Chatbots”. 2018 5th International Conference on Social Networks Analysis, Management and Security, SNAMS 2018 256–59. doi: 10.1109/SNAMS.2018.8554770.

UDIV de Infraestructura. (2022). Documento de aprobación.

Valderrama-Chauca, Enrique, Apaza-Huanca, Jorge, Cari-Mogrovejo, Lenin, Arizaca-Machaca, & Erick. (2022). Predictive Model Implemented in KNIME Based on Learning Analytics for Timely Decision-Making in Virtual Learning Environments. In IJETEL) International Journal of Emerging Technologies for E-Learning (Vol. 1, Issue 1).

Virkar, Mayuresh, Vikas Honmane, y S. Upendra Rao. 2019. “Humanizing the chatbot with semantics based natural language generation”. 2019 International Conference on Intelligent Computing and Control Systems, ICCS 2019 (Iciccs):891–94. doi: 10.1109/ICCS45141.2019.9065723.

ANEXOS

ANEXO I. ESTUDIO DE CASO RASA: NLU.

En la actualidad, el procesamiento del lenguaje natural (NLP) se ha vuelto cada vez más importante en la interacción humano-máquina. Para lograr una experiencia de conversación fluida y natural, es necesario comprender el significado intencional detrás de las entradas del usuario.

El objetivo de este estudio de caso es implementar un bloque NLU usando Rasa NLU, una plataforma de código abierto para el desarrollo de chatbots y sistemas de conversación. A través de la implementación de Rasa NLU, se busca mejorar la comprensión de las intenciones del usuario y proporcionar una respuesta más precisa y eficiente.

Este estudio de caso incluye los pasos necesarios para instalar Rasa NLU, definir intenciones y entidades relevantes, entrenar el modelo y evaluar su precisión. Además, se describen los resultados esperados y los benéficos cubrimientos esperados a medida que se implementa el bloque NLU con Rasa NLU.

Implementación de un bloque NLU con Rasa NLU

Objetivo: Crear un sistema capaz de comprender el significado intencional detrás de las entradas de texto y voz del usuario.

Alcance: El sistema será capaz de identificar intenciones, extraer entidades relevantes y proporcionar una respuesta apropiada basada en la intención del usuario.

Pasos de implementación:

- **Instalar Rasa NLU**
- **Crear un entorno de desarrollo**
- **Definir las intenciones y entidades relevantes**
- **Entrenar el modelo con datos de entrenamiento**
- **Evaluar la precisión del modelo**
- **Integrar el modelo en un sistema de conversación, como un chatbot o un asistente virtual.**

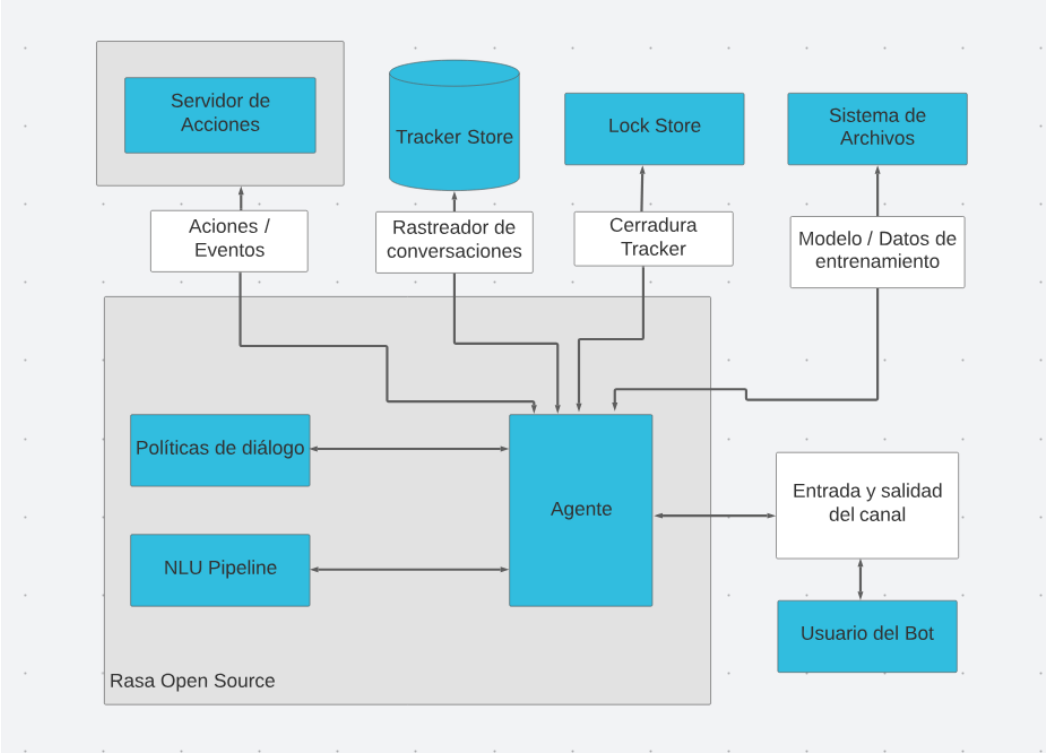
RESULTADOS ESPERADOS:

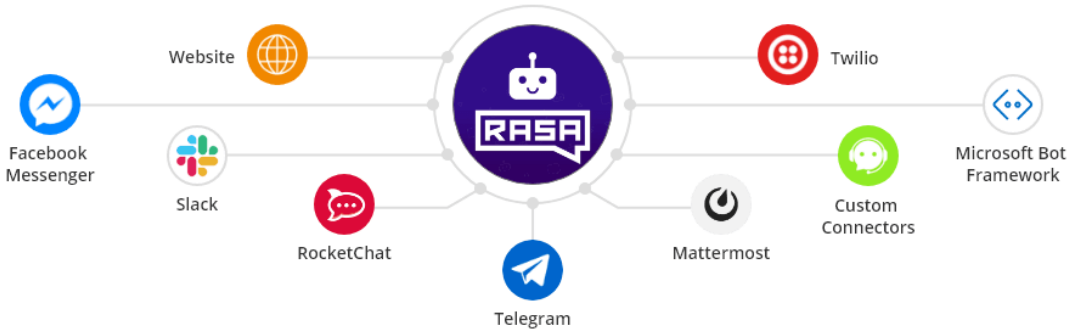
Un sistema capaz de comprender las intenciones del usuario y responder de manera apropiada, mejorando la interacción humano-máquina y la eficiencia en tareas repetitivas.

BENEFICIOS:

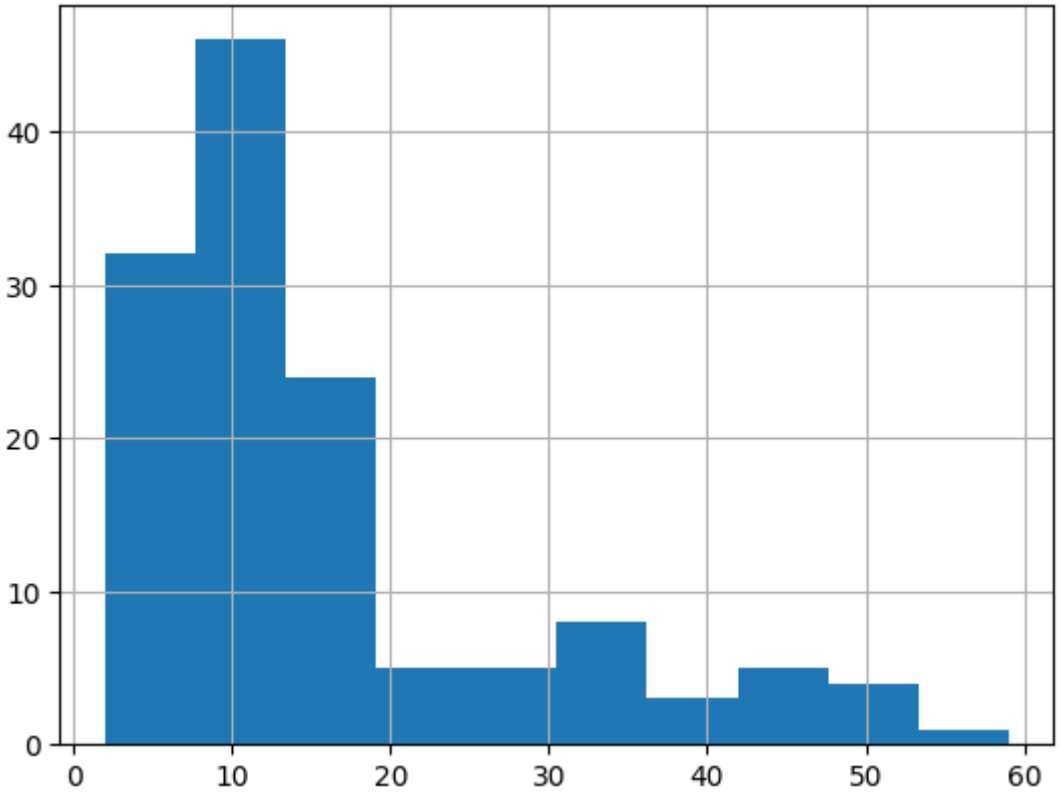
La implementación de un bloque NLU permitirá una mejor comprensión de las

entradas del usuario y una respuesta más precisa y eficiente, mejorando la satisfacción del usuario y la eficiencia en tareas repetitivas.



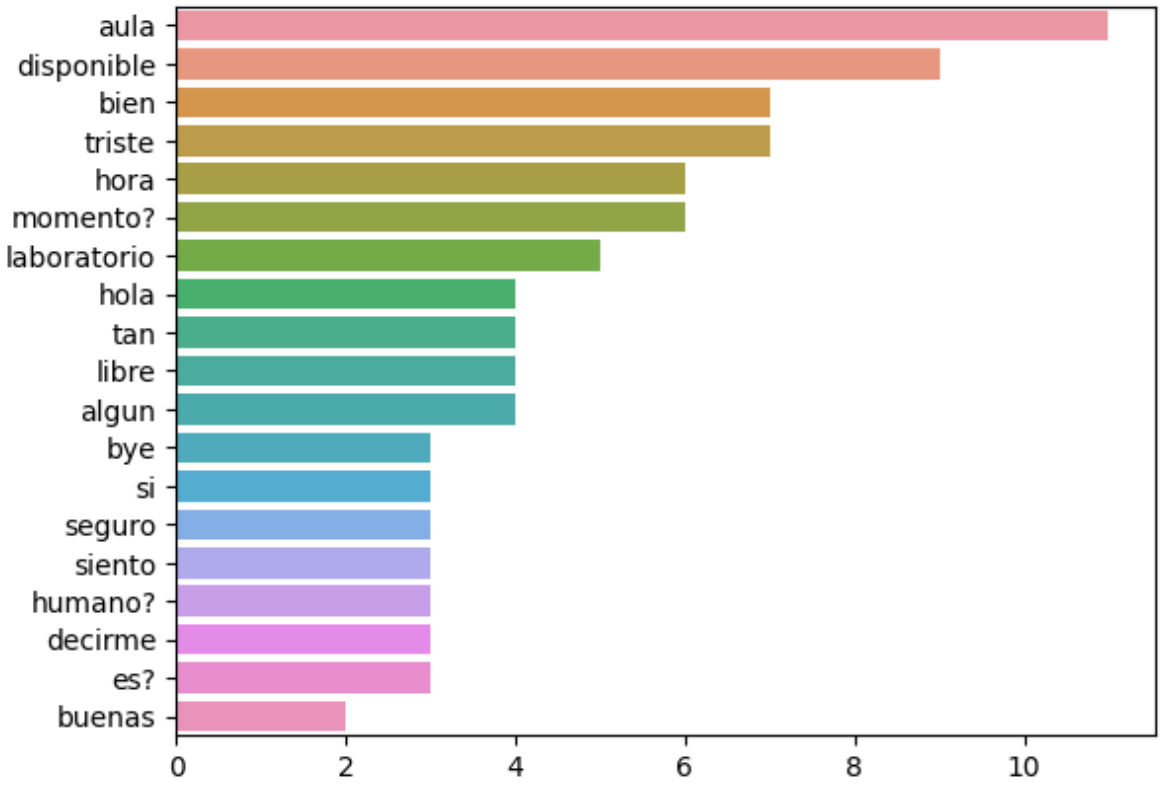


ANEXO II. ANÁLISIS EXPLORATORIO DEL CORPUS DE DATOS SELECCIONADO PARA LA PRIMERA ITERACIÓN DEL ENTRENAMIENTO DEL MODELO.



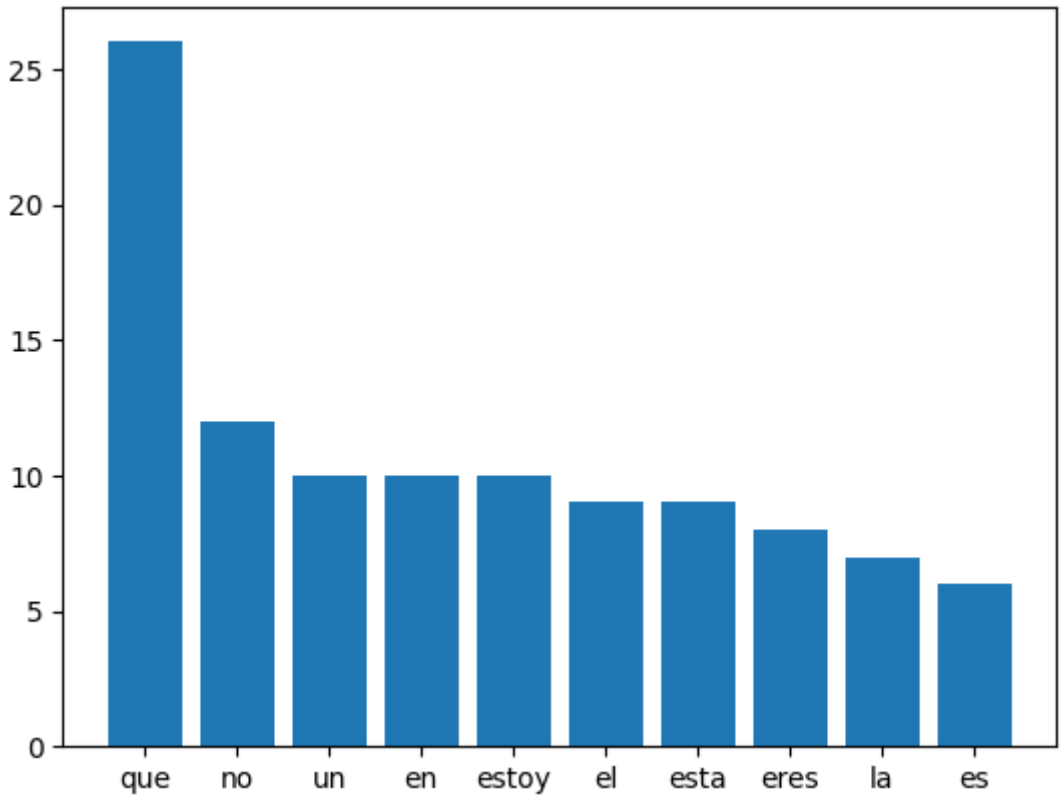
Anexo 2.1. Distribución de la Longitud de Entrada de Texto por Usuario

En el anexo 2.1 se puede observar en el eje de las X la longitud total de caracteres de la entrada de texto proporcionada por el usuario, mientras que en el eje de las Y las veces en que los textos de esa longitud se repiten. Se puede ver una gran frecuencia de textos entre 10-12 caracteres, de hecho más de 40 de los ejemplos de entrada del usuario tienen esa longitud de caracteres, a diferencia de los que poseen más de 50 caracteres que aparecen mínimamente.



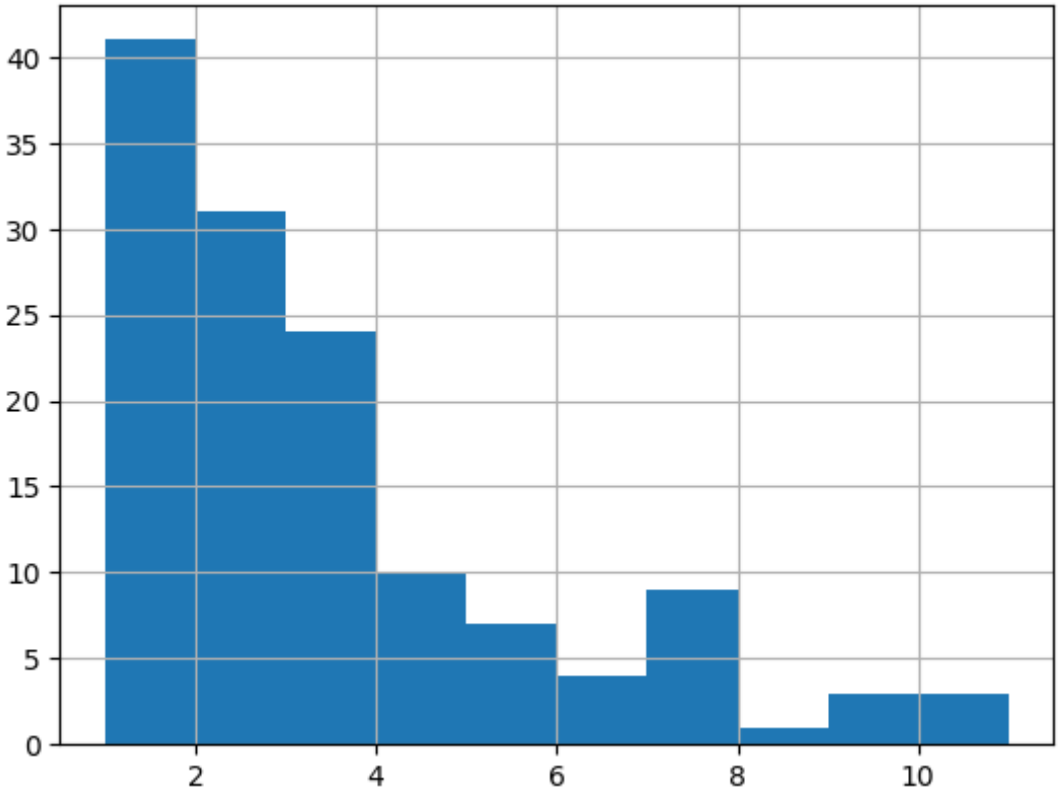
Anexo 2.1. Análisis de Frecuencia de Palabras Relevantes en Entrada de Texto por Usuario

En el anexo 2.2 se muestra la frecuencia con la que las palabras relevantes se repiten en los inputs del usuario, para esto se eliminaron todos los stopwords o palabras no contribuyen a identificar el tema o la idea principal del texto. En el eje de las X se representa la frecuencia, por otro lado, en eje de las Y se definen las palabras, como se observa “aula” es la palabra que más se repite, acorde a la categoría de experticia del modelo (ayudar en la gestión de espacios), que básicamente es el propósito de este, además de entender el lenguaje natural.



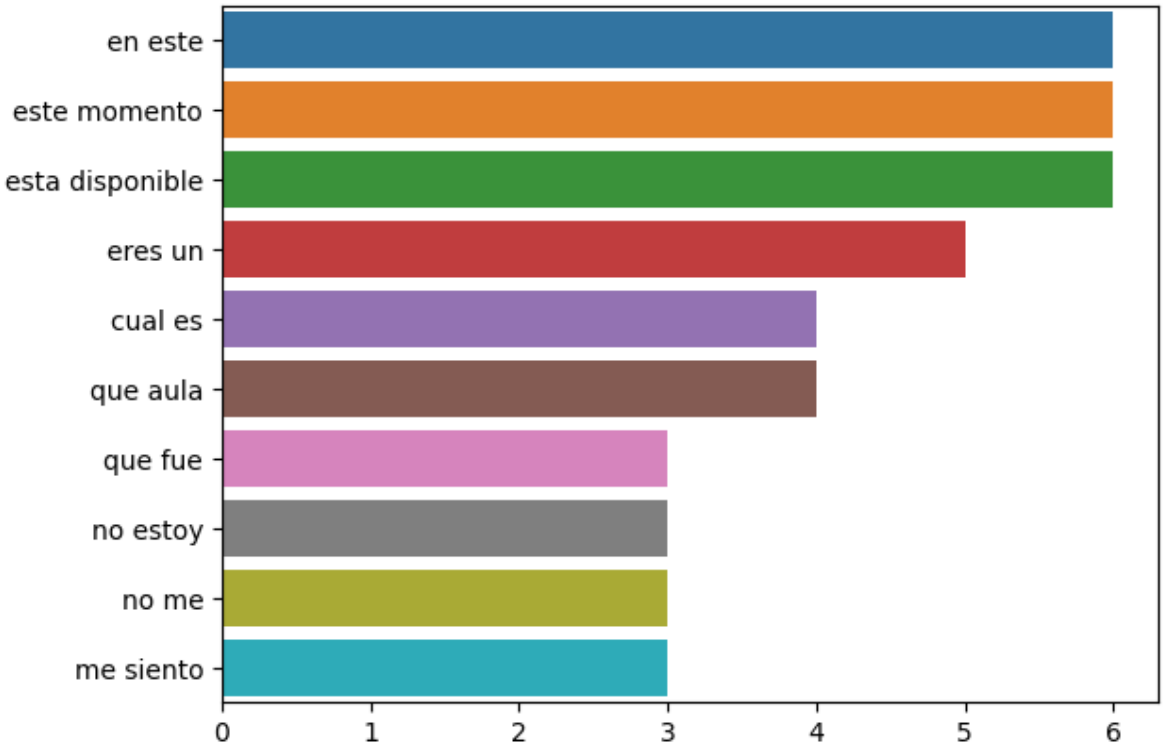
Anexo 2.2. Distribución de Frecuencia de Stopwords en Entrada de Texto por Usuario

En el anexo 2.3 se determina la frecuencia de los stopwords, es decir, se toma todo lo que fue descartado en el anexo 2.2, para verificar la frecuencia con la que aparece información no relevante en los inputs de los usuarios. En el eje de las X se definen las palabras no relevantes, y en el eje de las Y, su respectiva frecuencia. Se verifica que “que” es la palabra que más se repite, pero es evidente ya que es el monosílabo más utilizado como conector de ideas.



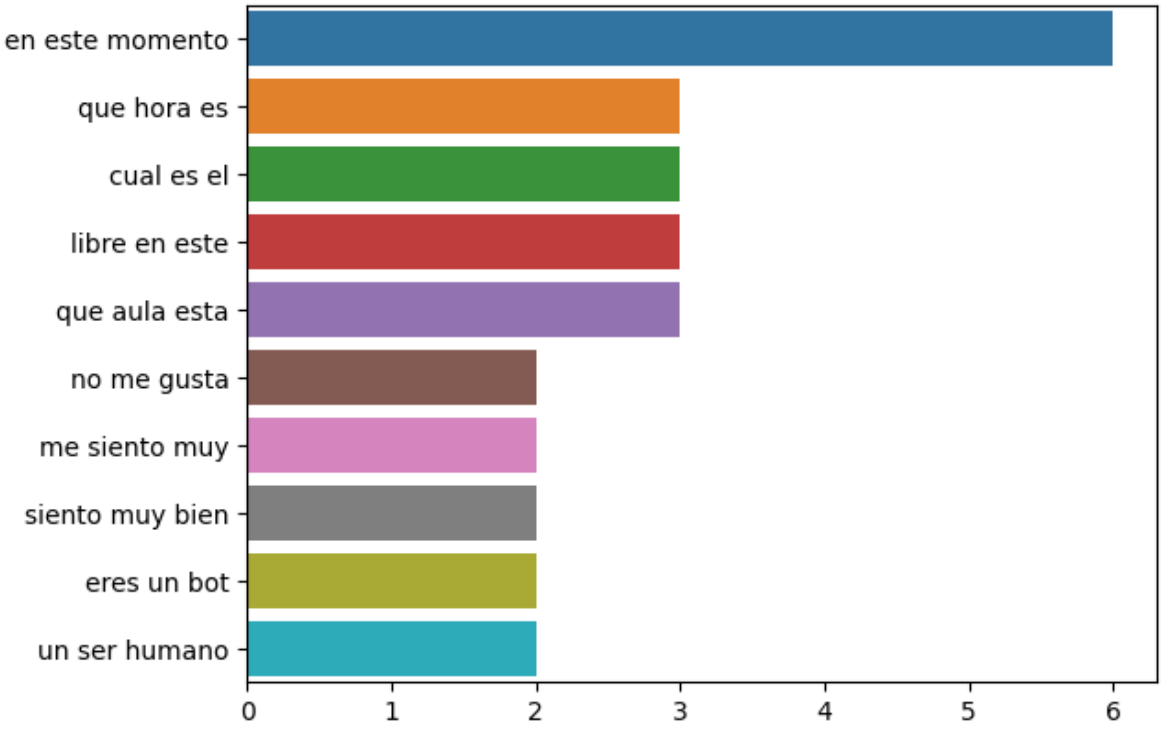
Anexo 2.3 - Distribución de la Cantidad de Palabras en Entrada de Texto por Usuario

El anexo 2.4 representa la cantidad de palabras que tiene un input del usuario (eje de las X) y el número de veces que ese input con la misma cantidad de palabras se repite (eje de las Y). Los inputs del 1-2 palabras son los más aparecen en el corpus, y esto se ve reflejado por la gran cantidad de variaciones de los saludos y despedidas.



Anexo 2.4. Frecuencia de Secuencias de Pares de Palabras en Entrada de Texto por Usuario

En el anexo 2.5 se hicieron secuencias de pares de palabras para evidenciar la frecuencia con la que aparecen en los inputs de los usuarios

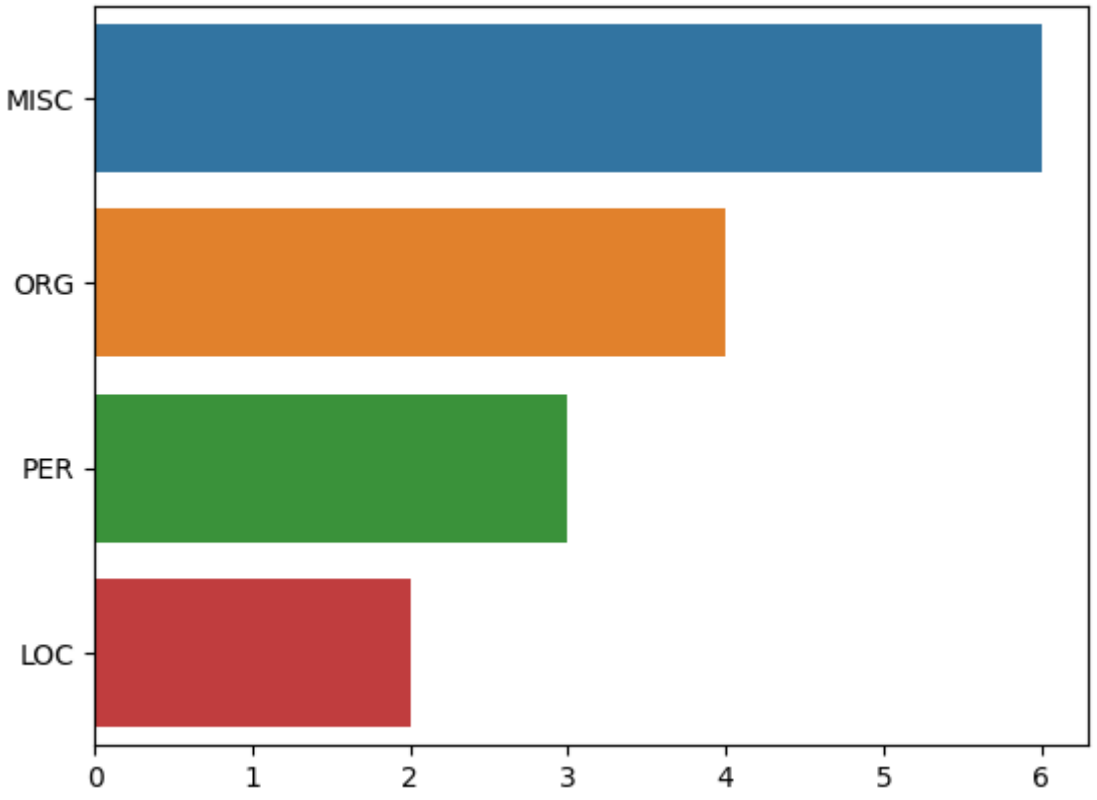


Anexo 2.5 - Frecuencia de Secuencias de Tríos de Palabras en Entrada de Texto por Usuario

En el anexo 2.6 se hicieron secuencias de tríos de palabras para evidenciar la frecuencia con la que aparecen en los inputs de los usuarios

Anexo 2.6. Nube de Palabras en Entrada de Texto por Usuario

En el anexo 2.7 se observa una nube de palabras, el tamaño de la representación indica la frecuencia con la que aparece en los textos, esto quiere decir que las veces que aparece la palabra es directamente proporcional a su tamaño.



Anexo 2.7. Extracción de Entidades en Entrada de Texto por Usuario

En el anexo 2.8 se realiza una extracción de entidades, MSC: Miscelánea, ORG: Organización, PER: Persona, LOC: Lugar. Aparece una mayor cantidad de entidades misceláneas, ya que los textos pertenecen a un campo mucho más general y no aportan muchas entidades evidentes.

ANEXO III. DESCRIPCIÓN Y VALIDACIÓN DEL ENTRENAMIENTO DEL MODELO.

Existen dos métodos de entrenamiento de un modelo de comprensión de lenguaje natural en el framework de Rasa NLU, para fines de evaluación, se utilizó el comando "*rasa train nlu*" que permitió identificar el proceso de selección de intenciones, además se demostraron las métricas específicas de confianza con la que el modelo tomaba las decisiones, de acuerdo con esto:

Se adecuaron los datos, historias y reglas de entrenamiento

rasa train nlu

```
(venv) C:\Users\Usuario\nlubot>rasa shell
C:\Users\Usuario\AppData\Local\Programs\Python\Python310\lib\site-packages\rasa\core\tracker_store.p
y:876: MovedIn20Warning: Deprecated API features detected! These feature(s) are not compatible with
SQLAlchemy 2.0. To prevent incompatible upgrades prior to updating applications, ensure requirements
 files are pinned to "sqlalchemy<2.0". Set environment variable SQLALCHEMY_WARN_20=1 to show all dep
recation warnings.  Set environment variable SQLALCHEMY_SILENCE_UBER_WARNING=1 to silence this messa
ge. (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base: DeclarativeMeta = declarative_base()
2023-05-24 01:30:55 INFO     rasa.core.processor  - Loading model models\nlu-20230524-012929-quadrat
ic-matrix.tar.gz...
NLU model loaded. Type a message and press enter to parse it.
Next message:
```

***Anexo 3.1**. Entrenamiento del modelo NLU*

Se puede observar en el anexo x que la ruta en la que se guardó el modelo entrenado fue en la carpeta models de la raíz del proyecto. Se puede diferenciar entre un modelo entrenado para interactuar con los servicios del asistente, de otro modelo dedicado a la validación del entrenamiento del mismo, y si, es posible validar el modelo con un proceso de comunicación usuario-asistente, pero resultaría en una comprobación a ciegas porque no se presentan métricas que aporten a la evaluación argumentativa del mismo.

Ahora bien, la diferencia más visible entre estos modelos, se encuentra en el nombre con el que se guardan, por ejemplo un modelo para:

Utilizar los servicios de interacción: ***"****models\283237823-4903402-kirby"*

Evaluar los servicios: ***"****models****nlu****-283237823-4903402-kirby"*

Una vez que el modelo culminó con su entrenamiento, se procedió a cargar respectivamente, y se indicó que se tenía que escribir un mensaje (input) para que el modelo lo pueda analizar, en el ejemplo de evaluación de la capacidad de comprensión con respecto al lenguaje natural, se obtuvieron los siguientes resultados:

```
NLU model loaded. Type a message and press enter to parse it.
Next message:
hola
{
  "text": "hola",
  "intent": {
    "name": "greet",
    "confidence": 0.9999626874923706
  },
  "entities": [],
  "text_tokens": [
    [
      0,
      4
    ]
  ],
  "intent_ranking": [
    {
      "name": "greet",
      "confidence": 0.9999626874923706
    },
    {
      "name": "bot_challenge",
      "confidence": 9.318023330706637e-06
    },
    {
      "name": "bot_challenge_father",
      "confidence": 4.266634732630337e-06
      "name": "mood_unhappy",
      "confidence": 1.9808512661256827e-06
    },
    {
      "name": "give_time",
      "confidence": 1.9537728803697973e-06
    },
    {
      "name": "goodbye",
      "confidence": 1.8263673382534762e-06
    },
    {
      "name": "mood_thanks",
      "confidence": 1.6541921468160 58e-06
    }
  ],
  "response_selector": {
    "all_retrieval_intents": [],
    "default": {
      "response": {
        "responses": null,
        "confidence": 0.0,
        "intent_response_key": null,
        "utter_action": "utter_None"
      },
      "ranking": []
```

Anexo 3.2. *Metricas de acuerdo a una entrada del Usuario*

En relación al anexo x, se interpretó que a partir del input “Hola”, el modelo consideró que se trataba de una intención de saludo con una confianza de 0.99 con respecto a las demás intenciones disponibles que tenía el modelo para escoger, luego, el modelo indicó el número de entidades que pudo haber encontrado en el texto de la entrada. A partir de esto el modelo determinó el número de tokens en los que se dividió el texto.

En este caso, considerando que solo es una palabra, "hola" se divide en un solo token, y los índices [0, 4] indican que el token abarcó desde la primera letra, es decir, en el índice 0 ("h") hasta el carácter en el índice 4, que es exclusivo. En otras palabras, el token abarca todos los caracteres de la palabra "hola".

Además, se incluyó un detalle con un ranking de intenciones para revisar específicamente cuáles fueron los valores de la confianza de cada una de las opciones que tenía el modelo para elegir, y tal como se puede observar, el valor de la confianza de las demás intenciones está muy por debajo de lo que representa el 10% de la confianza de la intención de saludo (0.99), en realidad, los otros valores de confianza corresponden a valores ínfimos que no se consideran representativos para influir en la toma de decisiones del modelo.

Con respecto a la selección de respuestas, que es el apartado final, se encuentra vacío, lo que significa que no se utilizó una ponderación en las respuestas ya que, en principio, no fue necesario determinar la selección de una respuesta específica, por lo que este campo es completamente aleatorio.

Reporte de métricas

Este reporte se asemeja al estilo del análisis exploratorio de datos que se demostró previamente en el **Anexo II**. Se analiza el modelo clasificador de intenciones, y para ello se utilizan todos los datos de entrenamiento del modelo ubicados en la carpeta “data”. Se tomó una muestra aleatoria del conjunto de instancias correspondientes a la definición de intenciones para validar la predicción de las mismas con respecto al último modelo entrenado, abarcando un total de 211 ejemplos.

Es necesario recordar que estos ejemplos no corresponden a la intención per se, sino que hacen referencia a las instancias que el modelo es capaz de reconocer para clasificar a la entrada con una intención específica, por ejemplo:

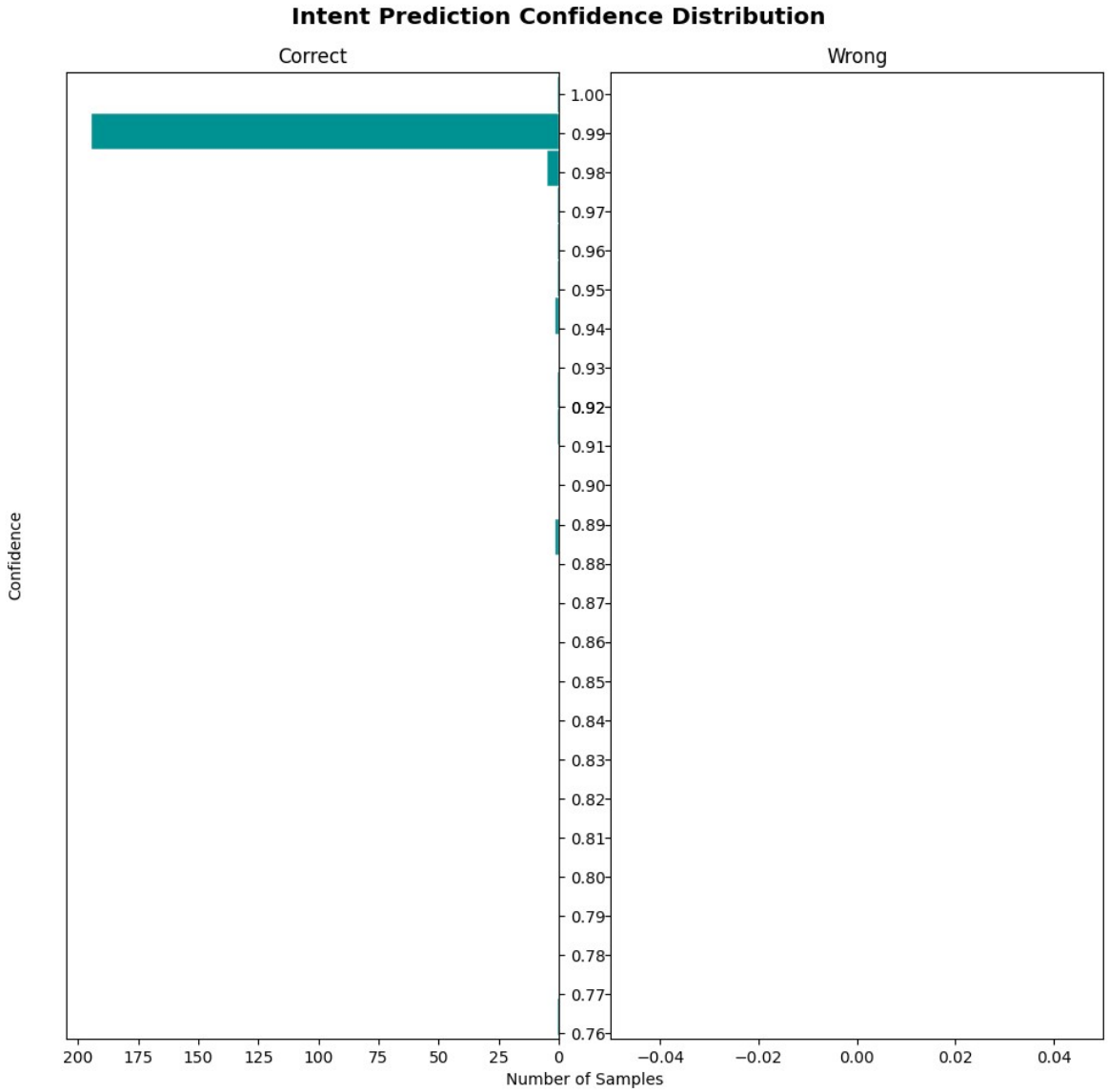
- intent: afirmación
  examples: |

- si

- yes

- afirmativo

Los ejemplos resaltados corresponden a lo que se refiere la muestra de validación de la predicción del modelo en base a determinadas intenciones, en este caso, “afirmación”.



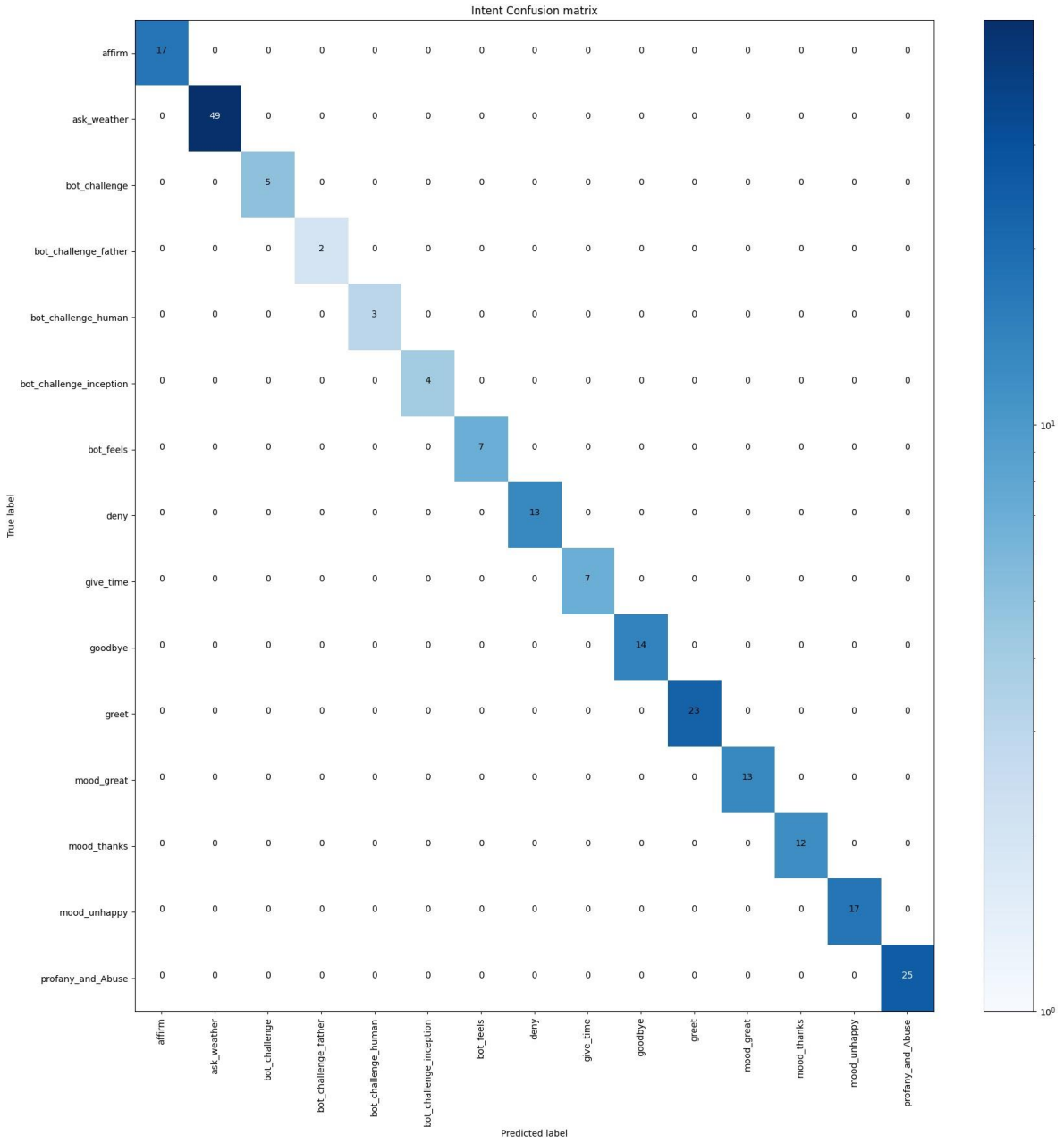
***Anexo 8**. Distribución de la predicción de las intenciones*

De acuerdo con el anexo 8, se pudo inferir que:

1. Todas las intenciones fueron clasificadas correctamente.
2. El valor mínimo de confianza en la predicción del modelo en referencia a las intenciones es del 0.89.

3. La mayor parte de los ejemplos se clasificaron de forma correcta con una confianza igual o mayor a 0.98.

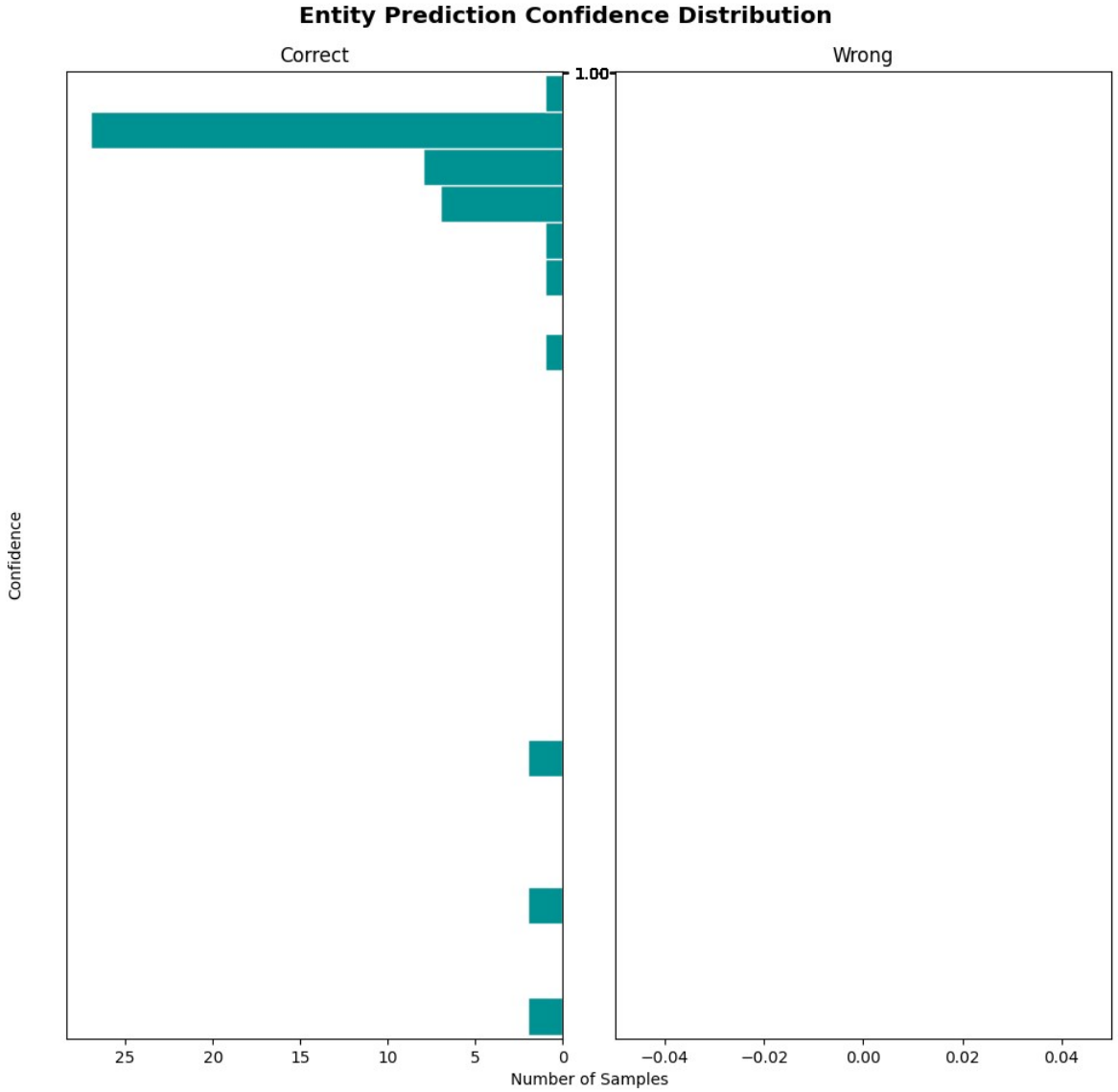
También se realizó un análisis más profundo, detallando la frecuencia de las intenciones (en términos de valores de predicción) dentro de una matriz de confusión



Anexo 9. *Matriz de confucion /clasificacion de intenciones)*

Según el anexo 9, se puede observar claramente que la intención con más representatividad de predicciones fue la acción de preguntar por el clima.

Ahora bien, con respecto a la calidad de la extracción de entidades en las acciones externas, el modelo predijo de forma correcta 28 de las 35 veces en que estas acciones fueron llamadas, comportándose de la siguiente manera:



Anexo 10. *distribucion de la prediccion de entidades*

El anexo 10 detalla la distribución de las de las muestras con respecto a la confianza de la predicción de las entidades, arrojada por el modelo más reciente que haya sido entrenado. A partir de la Figura se puede concluir que:

1. En todos los casos, el modelo predijo las entidades de forma correcta, ninguno cruzó el umbral del error.
2. Descartando las instancias aisladas, los datos siguen una distribución normal.
3. Más de 25 entidades se clasificaron de forma correcta con al menos más del 0.90 de la confianza posible.

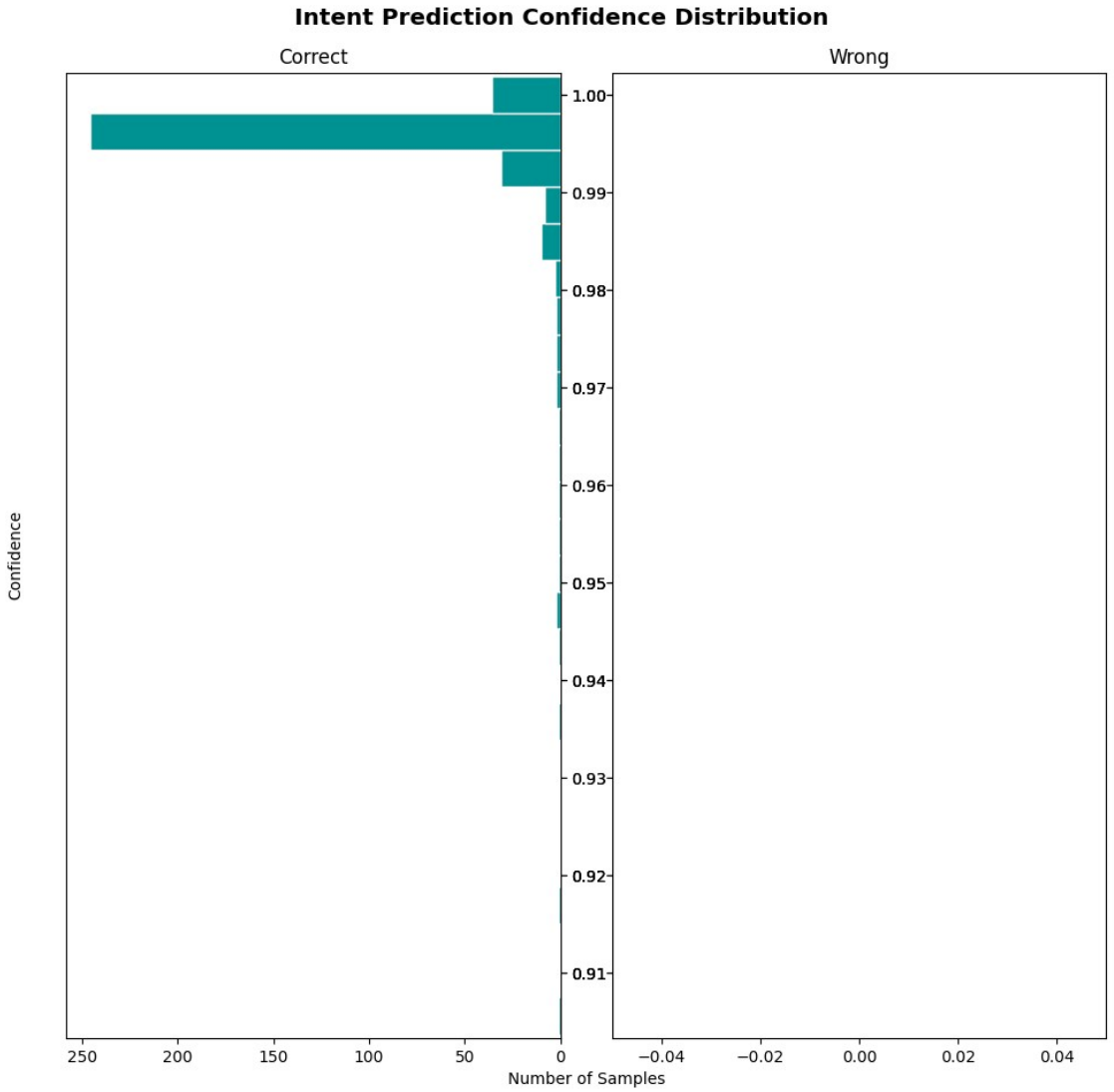
A continuación, se muestran los resultados de otras métricas clásicas de validación referentes a la predicción de las entidades en referencia a la muestra de ejemplos:

```
2023-05-26 00:46:14 INFO     rasa.core.test  - Evaluation Results on ACTION level:
2023-05-26 00:46:14 INFO     rasa.core.test  -  Correct:          28 / 35
2023-05-26 00:46:14 INFO     rasa.core.test  -  F1-Score:         0.846
2023-05-26 00:46:14 INFO     rasa.core.test  -  Precision:        0.915
2023-05-26 00:46:14 INFO     rasa.core.test  -  Accuracy:         0.824
```

Anexo 11. *Metricas de validacion de la extracion de entidades*

ANEXO IV. RE-VALIDACIÓN DEL ENTRENAMIENTO DEL MODELO.

En esta ocasión se realizó una simulación de la interacción con un total de 11 historian que contenían conversaciones entre el usuario y el asistente, y se obtuvieron los siguientes resultados



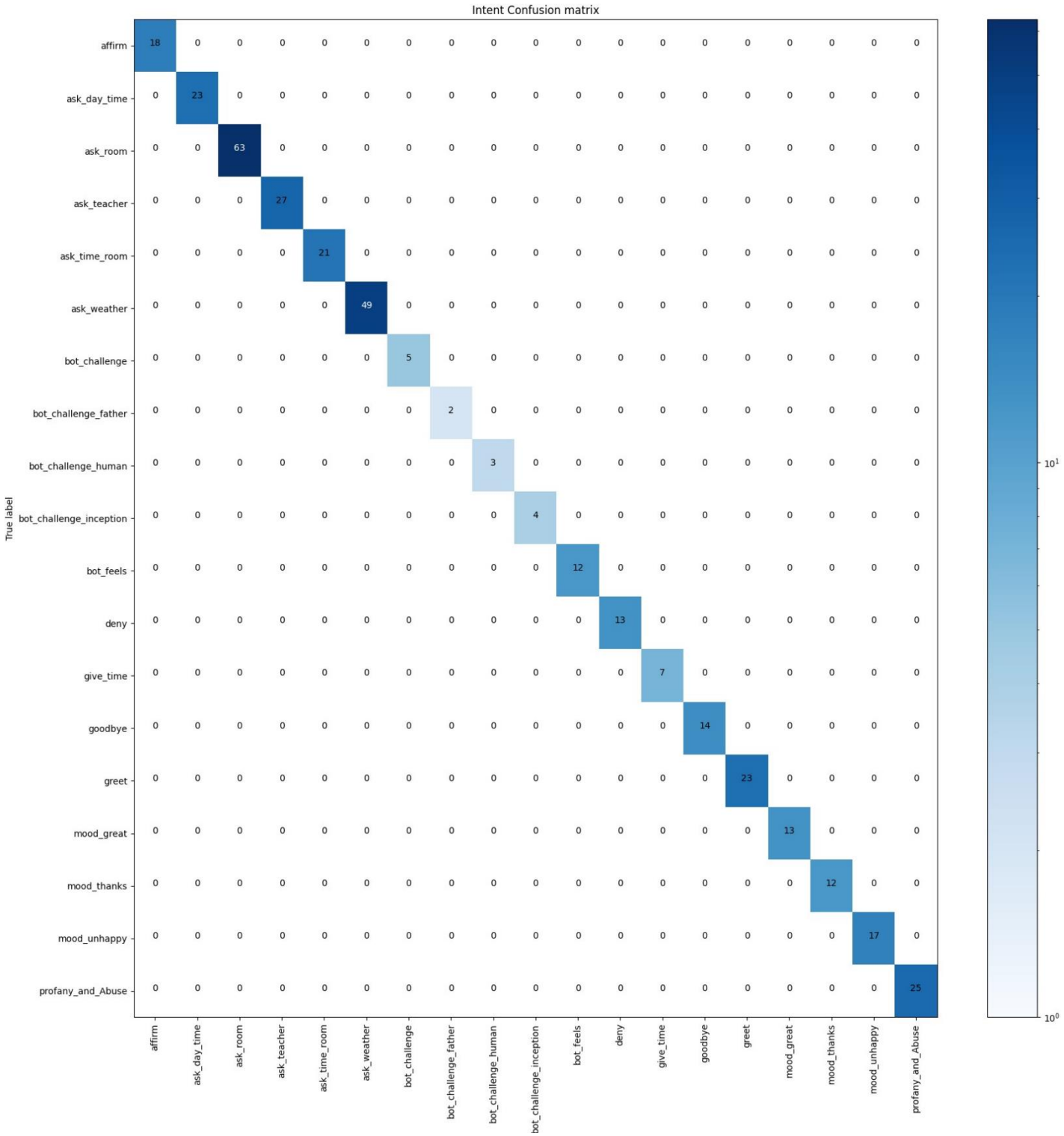
***Anexo 12**. Distribución de la predicción de las intenciones*

De acuerdo con el anexo 12, se pudo inferir que:

1. Todas las intenciones fueron clasificadas correctamente.
2. El valor mínimo de confianza en la predicción del modelo en referencia a las intenciones roza el 0.91.
3. La mayor parte de los ejemplos se clasificaron de forma correcta con una confianza igual o mayor a 0.99.

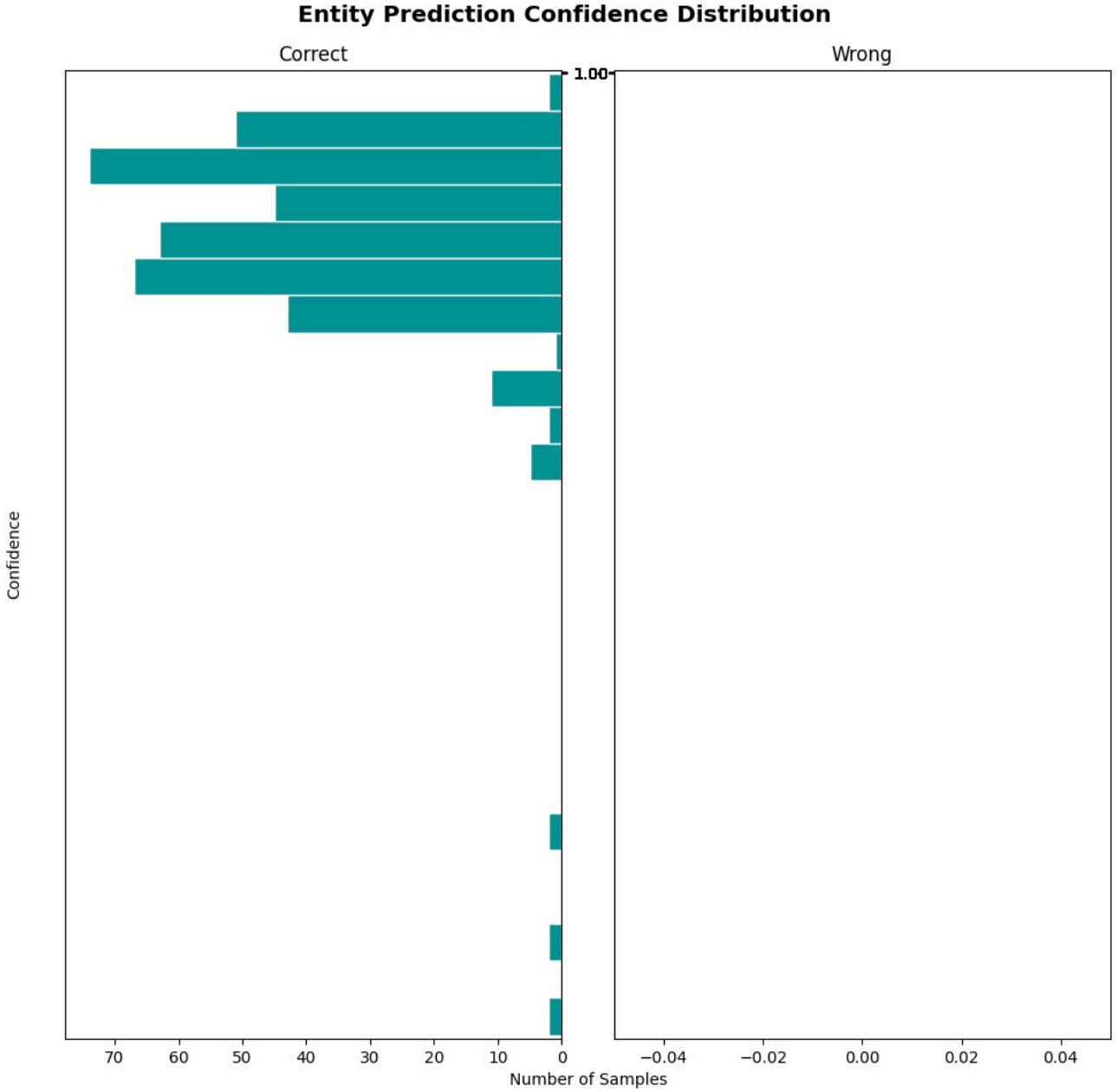
También se realizó un análisis más profundo, detallando la frecuencia de las intenciones (en términos de valores de predicción) dentro de una matriz de

confusión



Anexo 13. *Matriz de confucion /clasificacion de intenciones*

Según el anexo 13, se puede observar claramente que la intención con más representatividad de predicciones fue la acción de preguntar por el aula o laboratorio. Ahora bien, con respecto a la calidad de la extracción de entidades en las acciones externas, el modelo predijo de forma correcta 35 de las 41 veces en que estas acciones fueron llamadas, comportándose de la siguiente manera:



Anexo 14. *distribucion de la prediccion de entidades*

El anexo 14 detalla la distribución de las de las muestras con respecto a la confianza de la predicción de las entidades, arrojada por el modelo más reciente que haya sido entrenado. A partir de la Figura se puede concluir que:

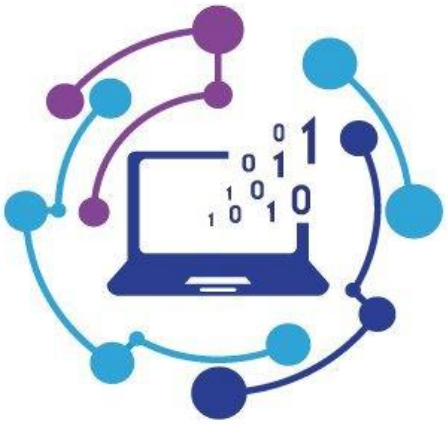
1. En todos los casos, el modelo predijo las entidades de forma correcta, ninguno cruzó el umbral del error.
2. La mayoría de las entidades tienden a clasificarse con una confianza de 1.00.

A continuación, se muestran los resultados de otras métricas clásicas de validación referentes a la predicción de las entidades en referencia a la muestra de ejemplos:

```
2023-06-07 00:41:14 INFO     rasa.core.test  -  Correct:          35 / 41
2023-06-07 00:41:14 INFO     rasa.core.test  -  F1-Score:        0.885
2023-06-07 00:41:14 INFO     rasa.core.test  -  Precision:       0.907
2023-06-07 00:41:14 INFO     rasa.core.test  -  Accuracy:        0.873
```

Anexo 15. *Metricas de validacion de la extracion de entidades.*

ANEXO V. MANUAL DE PROGRAMADOR.



UNIDAD DE TECNOLOGÍA

MANUAL DE PROGRAMADOR

OBJETIVO

Registrar la metodología, estructura lógica y características determinantes con las que el programador o el equipo de programación desarrolla y entrena el modelo NLU para el asistente virtual, de tal forma que estas especificaciones sirvan de referencia técnica a los nuevos integrantes del equipo, al soporte técnico del sistema, a la documentación y la evaluación del producto.

ChatUDIV

ASSISTENTE VIRTUAL EMPLEANDO TÉCNICAS DE COMPRESION DE LENGUAJE NATURAL

Manual de Programador

Autores:

Jesús Stefano Cajape Bravo

Sandro Alberto Palau Delgado

Versión 1.0.0 - 2023/06/06

TABLA DE CONTENIDO

1. INTRODUCCIÓN

El propósito de este manual es brindar a los desarrolladores o equipos de desarrollo una comprensión completa del patrón de entrenamientos, métodos, procedimientos y funciones empleados en la codificación del proyecto "DESARROLLO DE UN ASISTENTE VIRTUAL EMPLEANDO TÉCNICAS DE COMPRESIÓN DE LENGUAJE NATURAL".

2. IDENTIFICACIÓN DEL SISTEMA

PARAMETROS	DETALLE
Nombre del Sistema	ChatUDIV
Versión	V1.0
Logotipo	
Área de desarrollo	Esta plataforma fue desarrollada para la UDIV de Infraestructura de la Carrera de Computación de la ESAPM MFL
Desarrollador / Equipo	Jesus Stefano Cajape Bravo Sandro Antonio Palau Delgado
Modelo de desarrollo	CRISP-DM
Paradigma de programación	Orientada a objetos, declarativa, funcional, basado en eventos y asincrónica.

2.1.ALCANCE

Este documento describe información sobre los requerimientos y requisitos que debe cumplir el equipo de desarrollo para realizar correcciones o futuras actualizaciones al código fuente entregado. Adicionalmente incluye información general de la aplicación, como es el caso de la arquitectura, funcionamiento lógico, arquitectura utilizada y estándares de codificación.

2.2. REQUERIMIENTOS

2.2.1. Requerimientos de hardware

Las características que debe poseer el ordenador mediante el cual se interactuará con la solución software son las siguientes:

- Sistema operativo Windows, Linux o Mac
- 8 GB RAM mínimo, recomendado 16 GB RAM
- Procesador Intel Core I5 o Ryzen 5 mínimo recomendado
- Disco solido de 100 GB
- Acceso a Internet
- Un dispositivo inteligente

2.2.2. Herramientas de software

Para la ejecución un proyecto de Rasa, el desarrollador o equipo de trabajo deberá instalar en su equipo de trabajo las siguientes herramientas:

- Dependencias de Rasa NLU
- Sitema de gestión de paquetes (pip)
- Editor de código VsCode (recomendado)
- Python 3.10.5 (recomendado)

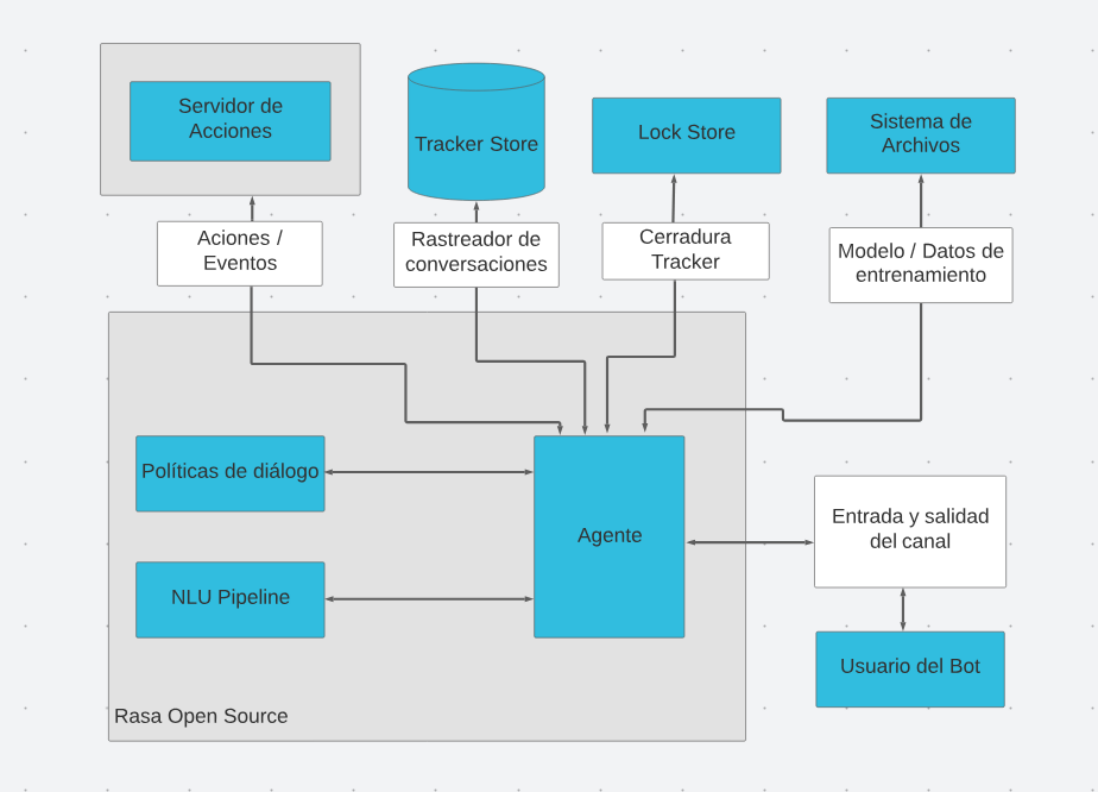
2.2.3. Conocimientos previos

Para que un equipo de trabajo realice cambios en la plataforma se deben tener conocimientos acerca de:

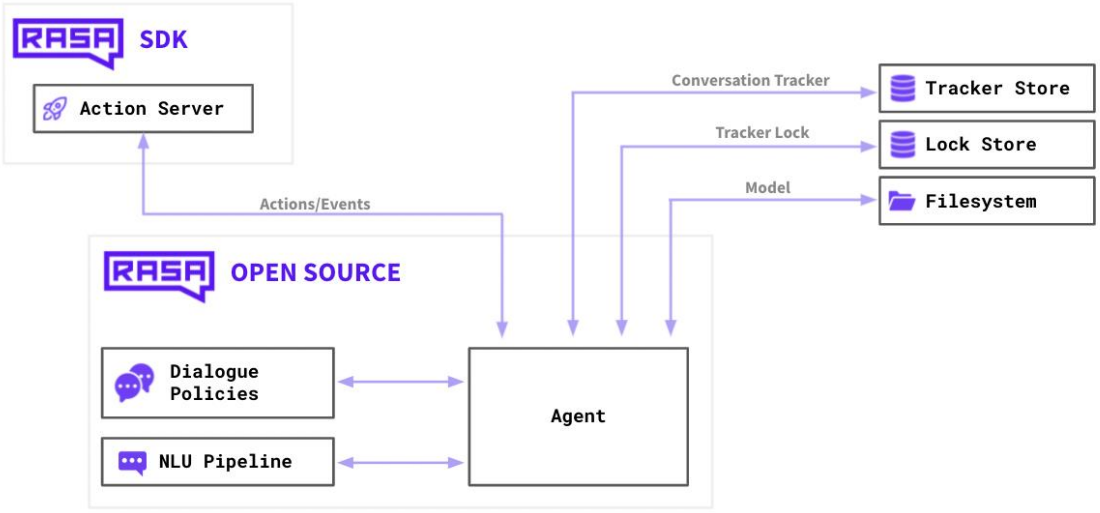
- Python
- Flask
- Sintaxis yml
- Postman
- Comandos de Rasa NLU
- Git y GitHub

3. ANÁLISIS Y DISEÑO DE LA APLICACIÓN

3.1.Vista Funcional



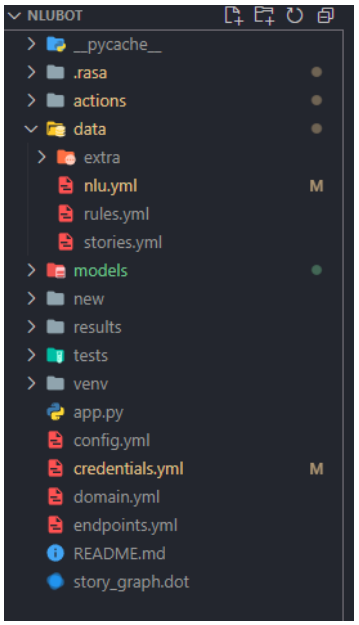
3.2.Vista Lógica (Arquitectura Cliente-Servidor)



3.3. Descripción del funcionamiento de Rasa

3.3.1. Ingresar datos de entrenamiento

Para entrenar nuevas funciones para el asistente virtual "ChatUDIV", es necesario abrir el proyecto en un editor de código. Recomendamos utilizar Visual Studio Code. La estructura del proyecto es la siguiente:



Carpeta "Data": Esta carpeta contiene el archivo "nlu.yml", que contiene todos los datos de entrenamiento con los que el modelo se entrenará. Es el archivo más importante en este caso.

Por ejemplo, para realiza un entrenemianto para que el modelo entienda los saludos, hay que seguir la siguiente estructura:

```
- intent: greet
  examples: |
    - buenas noches
    - buenas tardes
    - buenos dias
    - hey
    - hola
    - hola amigo
    - hola bro
```

Donde "- intent: greet" es el nombre que se le dará a la intención, es este caso colocamos saludo en inglés, luego se debe proporcionar ejemplos de estos saludos, como ven en el código proporcionado se coloca ejemplo tales como "buenas noches", "buenas tardes", "buenos días". Con esta estructura el modelo tiene conocimiento que todos estos datos que le estamos proporcionando son datos que debe de entender como saludo.

Una vez realizado esto, el siguiente paso es proporcionar las posibles respuestas que el asistente virtual puede dar en función de una o varias intenciones. Para ello, debemos dirigirnos al archivo "domain.yml". En este archivo, debemos crear las respuestas posibles que el asistente dará al usuario. Por ejemplo, si el usuario saluda, el modelo deberá responder con un saludo. Para lograr esto, debemos crear la siguiente estructura:

```
responses:
  utter_greet:
  - text: ¡Hola! 😀 Soy un asistente virtual para brindar soporte a las aplicaciones de la Carrera de Computación ESPAM MFL, ¿En qué puedo ayudarte hoy?
  - text: Buenos días 😁, Soy un asistente virtual para brindar soporte a las aplicaciones de la Carrera de Computación ESPAM MFL, ¿cómo puedo ser de ayuda?
  - text: Bienvenido 😊, Soy un asistente virtual para brindar soporte a las aplicaciones de la Carrera de Computación ESPAM MFL, ¿en qué puedo ayudarte?
  - text: Saludos 😄, Soy un asistente virtual para brindar soporte a las aplicaciones de la Carrera de Computación ESPAM MFL, ¿qué necesitas?
```

Dentro de la sección "responses" se encuentran todos los posibles saludos que el asistente dará en función de una intención. Debes crear una etiqueta para cada ejemplo en este caso, la etiqueta se llama "utter_greet" y se debe de escribir el texto que se quiere que el asistente virtual diga cuando lo saluden (en este ejemplo específico).

En el archivo "domain.yml", también debes especificar todas las intenciones que se han creado en el archivo "nlu.yml". Esto se realiza de la siguiente manera:

```
intents:
- affirm
- ask_weather
- bot_challenge
- bot_challenge_human
- deny
- give_time
- goodbye
- greet
- mood_great
- mood_unhappy
- bot_feels
- profany_and_Abuse
- nonsense_words
- mood_thanks
- bot_challenge_inception
- bot_challenge_father
- ask_teacher
- ask_room
- ask_time_room
- ask_day_time
```

Bien, ahora debemos generar son “story,yml” es un componente esencial para entrenar y mejorar la capacidad de conversación de un asistente virtual. Una story representa un ejemplo de interacción entre el usuario y el asistente, que incluye una serie de pasos o acciones. Y la estructura para generar una story es la siguien, poniendo de ejemplo el “greet” saludo.

```
- story: greeting
  steps:
  - intent: greet
  - action: utter_greet
```

Para especificar una story, es necesario asignarle un nombre, como en este caso, "greeting". Además, se deben detallar los pasos que se seguirán en la interacción. Por ejemplo, cuando la intención del usuario sea "saludar", la intención del modelo debería ser también "saludar", pero con uno de los saludos predeterminados que se especificaron en el archivo "domain.yml". Esto permite al asistente virtual responder adecuadamente al saludo del usuario, utilizando una de las respuestas predefinidas.

En rasa tambien permite creara reglas, son una parte fundamental en el flujo de conversación de un asistente virtual desarrollado con Rasa. Estas reglas permiten definir patrones de interacción específicos y establecer acciones basadas en ciertas condiciones. Para crear una

regla debemos irnos al archivo rule.yml y seguir la siguiente estructura.

```
rules:
- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye
```

Para crear una regla, es necesario asignarle un nombre en este caso el nombre es "Decir adiós cada que el usuario diga adiós". Esto le indica al asistente virtual que, sin importar en qué punto de la conversación se encuentre, cada vez que el usuario diga "Adiós", el asistente virtual debe responder con una despedida adecuada. De esta manera, se asegura una interacción coherente y amigable con el usuario.

3.3.2. Entrenamiento del Modelo

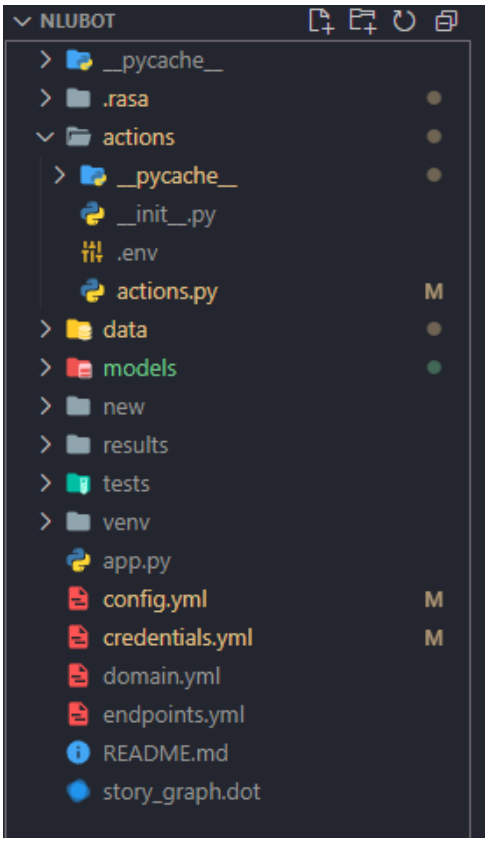
Una vez que todos los datos de entrenamiento estén registrados en sus respectivos archivos, es necesario proceder con el entrenamiento del modelo. Para ello, puedes acceder a Visual Studio Code y abrir una terminal de línea de comandos (CMD). Asegúrate de ubicarte en la ruta del proyecto, donde se encuentran los archivos necesarios para el entrenamiento. Desde allí, podremos ejecutar los comandos necesarios para iniciar el entrenamiento del modelo en Rasa:

- Para acceder al entorno virtual del proyecto, es necesario ejecutar el siguiente comando en la terminal: ".\venv\Scripts\activate". Esto activará el entorno virtual y te permitirá trabajar en el proyecto de forma aislada.
- Una vez dentro del entorno virtual, podremos entrenar el modelo utilizando el comando específico de Rasa: "rasa train". Se Ejecuta este comando en la terminal y el modelo comenzará a entrenarse, utilizando los datos de entrenamiento y la configuración especificada.
- Cuando el modelo esté entrenado, podremos probarlo utilizando el comando "rasa shell" en la terminal. Esto abrirá una sesión interactiva donde podrás simular una conversación con el modelo. Puedes ingresar mensajes como si fueras un usuario y el modelo responderá en consecuencia, mostrándote la salida en la terminal.

4. ACCIONES

4.1.Acciones Personalizada

Las acciones personalizadas en Rasa NLU son componentes esenciales para extender la funcionalidad y la interacción de un asistente virtual. Estas acciones permiten realizar tareas más complejas y específicas durante una conversación con el usuario. Una acción personalizada puede ser cualquier cosa, desde llamar a una API externa hasta interactuar con una base de datos o realizar cálculos personalizados.Para crear un accion personalizada en el proyecto de "ChatUDIV", debemos ir a la carpeta de action y dentro de esa carpeta vamos al archivo action.py



- En el archivo "action.py" se deben de importar las siguiente clases y modelos

```
from typing import Any, Text, Dict, List
from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher
```

necesarios para que la acción funcione correctamente

- Define una clase para tu acción personalizada que herede de la clase base "Action" proporcionada por Rasa:

```
class CustomAction(Action):
```

- Dentro de la clase, implementa el método "name()" para especificar el nombre de tu acción personalizada:

```
def name(self) -> Text:
    return "nombre_accion"
```

- Implementa el método "run()" donde definirás la lógica específica de tu acción personalizada

```
def run(self, dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
    # Lógica de tu acción personalizada
    # Puedes utilizar el objeto 'dispatcher' para enviar mensajes al usuario
    # y acceder a la información del 'tracker' para obtener datos del usuario
    return []       You, 1 second ago • Uncommitted changes
```

Una vez que hayas seguido estos pasos y definido tu acción personalizada, podrás utilizarla en el flujo de conversación de Rasa NLU. Recuerda especificar el nombre de tu acción en el archivo "domain.yml" y utilizarlo en las historias o reglas según corresponda.

```
actions:
- action_ask_weather
- action_show_time
- action_default_fallback
- action_ask_teacher
- action_ask_room
- action_ask_time_room
- action_day_time
```

Además, es fundamental crear el entrenamiento necesario para esta acción en el archivo "nlu.yml". Debes proporcionar ejemplos de frases de usuario que activen específicamente esta acción. Asimismo, es importante crear una historia que involucre el uso de estas acción en el archivo “story.yml”