UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Intent Based Networks: Desarrollo de un prototipo de networking con ayuda de IA y Machine Learning.

Carlos Andrés Montiel Dávila

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 15 de diciembre de 2023

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Intent Based Networks: Desarrollo de un prototipo de networking con ayuda de IA y Machine Learning.

Carlos Andrés Montiel Dávila

Nombre del profesor, Título académico **Ricardo Flores, Ph. D**

Quito, 15 de diciembre de 2023

© DERECHOS DE AUTOR

Nombres y apellidos: Carlos Andrés Montiel Dávila

Código: 00200407

Cédula de identidad: 1726083213

Lugar y fecha: Quito, 15 de diciembre de 2023

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Este trabajo realiza una exploración del concepto de Intent Based Networking (IBN), resaltando la necesidad de una revisión exhaustiva para evaluar el estado actual y anticipar direcciones futuras. IBN, un paradigma destinado a simplificar la gestión de redes, se enfoca en proporcionar intervenciones flexibles mediante la expresión de "intents," que son esencialmente requerimientos del usuario. Históricamente, IBN tiene conexiones con autonomic networks y computing, subrayando la importancia de políticas de red para la gestión automática de redes.

La identificación de "intents" implica expresiones a través de plantillas e interfaces gráficas, siendo esencial para la traducción efectiva. Esta traducción, fundamental para configurar nodos individualmente, emplea enfoques como plantillas, mapeo, refinamiento, y tecnologías semánticas. Sin embargo, el paradigma de IBN enfrenta desafíos críticos en la traducción y perfilamiento de intenciones, debido a la falta de definiciones claras y la variabilidad en la expresión de "intents," lo cual complica la implementación y requiere una solución efectiva.

La solución propuesta para abordar estos desafíos se basa en la implementación de un chatbot desarrollado con Rasa. Este chatbot utiliza Procesamiento del Lenguaje Natural (NLP) para reconocer intents, instrucciones y entidades de red expresadas por el usuario. Su función principal es la capacidad de crear y desplegar redes en un ambiente de producion, todo ello guiado por los requerimientos del usuario.

En esta solución, se han incorporado diversos ejemplos que abarcan la creación de redes con conexiones a switches, routers, y la implementación de VLANs. Además, el chatbot proporciona respuestas claras a conceptos básicos de networking y ofrece instrucciones

específicas para la configuración de nodos como routers. Estos ejemplos concretos facilitan la comprensión y utilización efectiva del sistema por parte del usuario.

El chatbot está diseñado para ser intuitivo y accesible para personas sin conocimientos previos en redes. Los intents de creación expresados por el usuario se traducen de manera inteligente a configuraciones de red de bajo nivel, permitiendo una interacción sencilla y efectiva. Este enfoque no solo simplifica la gestión de redes, sino que también democratiza el acceso al despliegue de infraestructuras de red, eliminando barreras de entrada técnica. La solución destaca por abordar de manera excepcional los pasos más desafiantes en IBN: el perfilamiento y la traducción de "intents". La habilidad del chatbot de interpretar expresiones de lenguaje natural a través de plantillas y su capacidad para traducir de manera inteligente los "intents" a configuraciones de red de bajo nivel representa un avance significativo.

Palabras clave: Intent Based Networking (IBN), Chatbot con Procesamiento del Lenguaje Natural (NLP), Gestión Automática de Redes, Automatización de Procesos.

ABSTRACT

This study explores the Intent-Based Networking (IBN) concept, emphasizing the need for a comprehensive review to assess the current state and anticipate future directions. IBN, a paradigm aimed at simplifying network management, focuses on providing flexible interventions through the expression of "intents," essentially user instructions. Historically, IBN has connections with autonomic networks and computing, underscoring the significance of network policies for automatic network management.

The profiling of "intents" involves expressions through templates and graphical interfaces, crucial for effective translation. This translation, essential for configuring nodes individually, employs approaches such as templates, mapping, refinement, and semantic technologies. However, the IBN paradigm faces critical challenges in the translation and profiling of intentions due to the lack of clear definitions and variability in the expression of "intents," complicating implementation and requiring an effective solution.

The proposed solution to address these challenges is based on the implementation of a chatbot developed with Rasa. This chatbot uses Natural Language Processing (NLP) to recognize intents, instructions, and requirements expressed by the user. Its main function is the ability to create and deploy networks in a simulation environment, all guided by user commands and intentions.

In this solution, various examples have been incorporated covering the creation of networks with connections to switches, routers, and the implementation of VLANs. Additionally, the chatbot provides clear responses to basic networking concepts and offers specific instructions for configuring nodes such as routers. These concrete examples facilitate the understanding and effective utilization of the system by the user.

The chatbot is designed to be intuitive and accessible to individuals without prior knowledge of networks. User-expressed creation intents are intelligently translated into low-level network configurations, enabling simple and effective interaction. This approach not only simplifies network management but also democratizes access to the deployment of network infrastructures, eliminating technical entry barriers. The solution stands out for exceptionally addressing the most challenging steps in IBN: profiling and translating "intents." The chatbot's ability to interpret natural language expressions through templates and its capability to intelligently translate "intents" into low-level network configurations represent a significant advancement.

Keywords: Intent-Based Networking (IBN), Chatbot with Natural Language Processing (NLP), Automatic Network Management, Process Automatization.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El paradigma de Intent-Based Networking (IBN) ha emergido como una revolucionaria aproximación para simplificar la gestión de redes, permitiendo a los administradores de red expresar sus requerimientos intenciones comerciales de manera abstracta, mientras la red interpreta y ejecuta estos requerimientos de manera automatizada. No obstante, a medida que esta tecnología ha evolucionado, se ha identificado una problemática significativa, especialmente en la traducción e identificación de configuraciones a nivel bajo. La transición de las intenciones del usuario a la configuración detallada de dispositivos de red puede ser un desafío, exigiendo soluciones que aborden esta complejidad de manera integral y amigable para el usuario.

Configurar dispositivos de red a nivel bajo, aunque es crucial, a menudo se convierte en un cuello de botella debido a la falta de herramientas intuitivas y soluciones eficaces, lo cual puede resultar en daños colaterales afectando aplicaciones: “Incorrect information, or misconfiguration, could interfere with the running of networked applications.” (Agarwal et al., 2009, p. 349). Aquí es donde entra en juego la necesidad de una solución integral que no solo aborde los problemas de configuración detallada, sino que también sea capaz de gestionar simultáneamente diversos aspectos de despliegue de configuraciones relacionados al IBN. Un enfoque que simplifique la configuración a nivel de dispositivo, al tiempo que se ocupa de la complejidad de la red, es esencial para hacer que IBN sea accesible y utilizable para una gama más amplia de profesionales de redes.

En este contexto, se busca una solución que, además de ser técnica y eficaz, también sea amigable con el usuario. La usabilidad es clave para garantizar que los profesionales de redes puedan aprovechar al máximo el potencial del IBN sin enfrentar obstáculos significativos en la implementación y gestión diaria de la red. Adicionalmente se busca que la solución no

sea únicamente aplicable para profesionales de red, se busca idealmente que esta sea útil para usuarios no expertos en redes.

2. REVISIÓN DE ESTADO DEL ARTE

Según Leivadeas: “Thus, we strongly believe, that the IBN concept is starting to get mature enough and a thorough review of the current developments is needed to evaluate the current state and reveal the future directions.” (Leivadeas, p.625,2023) se puede concluir que, al tener un estado avanzado y maduro de los avances actuales relacionados a IBN, se puede definir un marco teórico que logra definir el termino, proceso de creación, retos. Con el fin de poder implementar el chatbot, se debe tener una noción clara de los conceptos que se debe tratar y principalmente ver que problemáticas y soluciones existen al momento de plantear un sistema basado en “intents’

Debido a la necesidad de simplificar el manejo y configuración de una red, se usa el paradigma de Intent Based Networking (IBN), obteniendo intervenciones flexibles, agiles y simplificadas que ayudan al usuario a automatizar y simplificar el proceso. Al ofrecer una capa de abstracción y enfocarse en la funcionalidad final de una arquitectura de red, el usuario puede expresar como quiere que la red se comporte de forma general sin preocuparse específicamente en la configuración subyacente de cada nodo. Los requerimientos del usuario o programador de red se llevan a cabo en expresiones de lenguaje natural sin implicar conceptos técnicos especializados que implican conocimiento de configuraciones de bajo nivel. Esto es mejor conocido como “intents”.

Con el fin de traducir “intents” a configuraciones de red efectivas y automatizaciones de circuito cerrado, es decir un sistema que gestione diferentes redes y dispositivos conectados aislado de conexiones externas, se lleva a cabo un proceso que tiene 5 pasos que son esenciales para la creación y despliegue de redes a base de sus intents: identificación, traducción, resolución, activación y aseguramiento.

La identificación se basa en el paso inicial donde el usuario va a interactuar con el sistema para expresar su “intent”, el sistema donde se despliega el IBN debe tener la capacidad de interactuar con el usuario. En segundo lugar, el “intent” perfilado por medio de la expresión de lenguaje natural se traduce a configuraciones de red de bajo nivel que puedan ser implementados por los diferentes dispositivos de la red. La resolución se lleva a cabo cuando dos usuarios tratan de enviar sus “intents” al mismo tiempo, causando configuraciones de red que interfieren entre sí. Un sistema IBN debe tener un módulo que resuelva los “intents” que causan conflicto por medio de un módulo de resolución de políticas de red. La activación de “intents” consiste en que los requerimientos de configuración y de despliegue de la red por parte del usuario que creo el “intent” se ejecuten. Finalmente, el aseguramiento busca que el “intent” se cumpla incluso frente a condiciones dinámicas de red que puedan presentar dificultades, brindando medidas reactivas y de auto configuración de la red.

A pesar de ser un termino relativamente nuevo IBN tiene conexiones con la historia: “(..)it has strong ties with the past and the work that has been done around autonomic networks and computing.” (Leivadeas & Falkner, 2022, p. 630). Por ejemplo, SDN (Software Defined Networks) esta estrictamente relacionado con IBN, separando el plano de control con el plano de datos, y teniendo una arquitectura de tres capas en la cual el plano de control configura el comportamiento de los dispositivos y el plano de aplicación permite que los usuarios envíen sus pedidos que controlan el comportamiento y manejo de red. Las políticas de red

proporcionan reglas detalladas para el comportamiento de la red, complementando los requerimientos o" intents" de IBN al definir cómo deben ocurrir los cambios en la red. Las Políticas de Red desempeñan roles cruciales en la realización y automatización de la gestión de redes según los requisitos del usuario, cerrando la brecha entre los requerimientos del usuario y la configuración de la red.

Para la identificación de "intents" se destaca la forma en que se expresa el "intent", particularmente a través de plantillas e interfaces gráficas de usuario (GUI), siendo este mecanismo deseable debido a su retroalimentación con el usuario. Estas interfaces graficas permiten a los usuarios especificar un requerimiento utilizando menús desplegables y arrastrar y soltar elementos, lo que facilita la comunicación de lo que desean en la red. Las GUIs varían en complejidad según el contexto. El enfoque de GUI es una herramienta semiflexible que ayuda a los usuarios a expresar el requerimiento, pero a la vez puede restringir al usuario al no permitir especificar algo que no esté disponible como opción. Para abordar esto, IBN puede expresarse en lenguaje humano, ya sea escrito u oral, y debe estar equipado con una herramienta de Procesamiento del Lenguaje Natural (NLP), una subdisciplina de la Inteligencia Artificial (IA), que convierta el lenguaje humano en lenguaje de máquina. NLP se puede utilizar en diferentes contextos, como el reconocimiento de entidades nombradas, extracción de relaciones y extracción de palabras clave. En IBN, NLP se aplica a menudo a sistemas que ofrecen una interfaz conversacional similar a un chatbot, permitiendo a los usuarios expresar su requerimiento en lenguaje natural.

Se emplean diversos mecanismos de traducción según el tipo y alcance de la expresión del "intent". Un enfoque común es la traducción basada en plantillas y diseños, donde se asocian plantillas predefinidas con los requerimientos de los usuarios, lo que simplifica la conversión en políticas de red. Este método es especialmente adecuado para expresiones

basadas en interfaces gráficas y plantillas. Además, se utilizan técnicas de mapeo para manejar requerimientos abstractos descomponiéndolos en descriptores de red específicos, que luego se asocian con políticas predefinidas. Para escenarios más complejos, entran en juego procesos de refinamiento y Descriptores de Servicios de Red (NSD), lo que permite una traducción más detallada del requerimiento. Los enfoques inferenciales, combinados con NSD, pueden aprender y mejorar las traducciones con el tiempo, deduciendo partes faltantes y correlaciones entre elementos en el "intent". Los mecanismos basados en palabras clave y la clasificación mediante aprendizaje automático ayudan a mapear palabras clave a características o políticas de red específicas, ofreciendo un enfoque de procesamiento de lenguaje natural (NLP). Los gráficos semánticos RDF y los lenguajes semánticos como OWL se utilizan para estructurar y extraer relaciones semánticas dentro de los modelos de "intents", lo que facilita la traducción en políticas de servicio.

En el caso de detección y resolución de conflictos de "intents", se debe verificar que estos puedan ser implementados en la red y evitar que exista otros "intents" que puedan colisionar o causar conflicto. Los mecanismos que se hacen para detectar los conflictos se basan en diferentes enfoques. Por ejemplo, se puede detectar un conflicto antes de enviar un "intent" verificando que el "intent" pueda llevarse a cabo en las condiciones de red dadas y si las políticas de red establecidas permiten el despliegue del "intent". Otra técnica conocida es el mapeo de "intents" uno a uno determinando si están asociadas en cierto tiempo o espacio físico de la red. Igualmente se puede definir a los "intents" como grafos y solucionar las prioridades según diferentes métodos de búsqueda de grafos.

Al existir diferentes contextos sobre los cuales se puede desplegar un "intent" debido a la naturaleza de este y el tipo de usuario, el enfoque del proyecto y en general al tener un sistema de IBN debe priorizar la identificación y la traducción del "intent", con el fin de

configurar exitosamente los nodos individualmente. Uno de los mayores retos se da en estas dos fases: "(..) the bigger challenge here is how to make sure that the intent will be accurately captured and following how to automatically generate user configuration scripts (..)" (Leivadeas & Falkner, 2022, p. 628), al tener que identificar entidades como dispositivos y números de dispositivos y la configuración que se desea llevar a cabo en estos, se necesita un mecanismo de detección.

Existen diferentes retos al crear un sistema IBN en las diferentes fases clave y se los puede categorizar de la siguiente forma según la fase:

1. Identificación

- En cuanto a la definición del "intent" se pueden presentar problemas al tener diferentes usuarios con diferentes perspectivas y conocimiento de la red, expresándose de diferente forma. En este caso, un "intent" puede ser interpretado por un sistema IBN, pero no reconocido por otro.
- Se busca que el usuario y el "intent" tengan una alineación, es decir que basándose en el tipo de usuario y el conocimiento que este tenga, el "intent" puede tener un mayor o menor nivel de detalle y esto debe coincidir con el requerimiento de configuración. Esto significa que un "intent" pueda ser fácilmente reconocido y traducido a políticas de red, mientras que otros no.
- Al ser el networking un servicio que depende del administrador de red, se debe dar un punto medio en el cual se logre que este pueda expresar el "intent" y que las configuraciones deseadas puedan ser expresadas a alto nivel.

2. Cumplimiento

- Al momento de traducir un “intent” se puede generar más de una política de red que debe ser capturada por el sistema, una traducción errónea ocasionaría un rendimiento pobre de la red. Se busca que en cada capa se refine el “intent”, extrayendo políticas de red adecuadas y desplegando la configuración deseada.
- Un “intent” debe ser “Vendor agnostic” es decir que debe esconder y hacer caso omiso a detalles del equipo de red y tecnologías usadas por el fabricante, se busca que la configuración sirva con diferentes productos usando comandos genéricos.
- Como se mencionó anteriormente, el diseño de un módulo que resuelva conflictos al desplegar “intents” es vital para el funcionamiento de un sistema IBN. La creación y diseño de este módulo es complejo debido al número de “intents”, roles y prioridades de la red que son dinámicos.

3. Aseguramiento

- Un sistema IBN busca tener un comportamiento de red constante, lo cual no sucede en la realidad, por ende, se necesita un mecanismo de reintento de despliegue y de refinamiento según la condición de la red.
- Igualmente, un “intent” puede ser dinámico y puede cambiar según las condiciones de red dadas, creando retos adicionales ya que este evento dinámico puede necesitar recursos de red adicionales y afectar a otros “intents”.
- Finalmente, se necesita herramientas de monitoreo de red y telemetría para asegurar el cumplimiento de los “intents”. Estas herramientas pueden crear pequeñas

interrupciones en el servicio, siendo un problema para el funcionamiento general de la red.

Con el fin de plantear una solución que logre resolver los problemas al momento de perfilar y traducir "intents", se busca utilizar frameworks de inteligencia artificial para la creación de un chatbot que pueda proporcionar retroalimentación oportuna al usuario e interactuar para guiar en el proceso de creación y despliegue de su requerimiento. Se analiza diferentes opciones y se busca hacer una comparación entre las 3 herramientas más populares para la creación de chatbots: IBM Watson[1], Google Dialogflow [2]y Rasa[3]. Siendo estas opciones las más conocidas debido a su flexibilidad, precio, disponibilidad de comunicación y funcionalidad.

Tanto Dialogflow como Watson son opciones con una gama amplia de servicios de IA y de Machine Learning, reconocen y procesan diferentes prompts y queries. Al tener estas capacidades avanzadas, ambos productos son "closed source" y el precio de ambos escala según el uso y la magnitud del proyecto, siendo difícil poder hacer pruebas y desarrollar proyectos de magnitud pequeña sin tener que invertir una larga cantidad de dinero.

Por otro lado, Rasa se destaca debido a su naturaleza "open source", brindando a los desarrolladores flexibilidad y control sobre sus aplicaciones. Rasa permite la creación de chatbots que usen modelos de machine learning personalizados y que pueden ser entrenados para cada caso especial. Al tener una documentación amplia, la curva de aprendizaje de Rasa es baja comparada con otros frameworks mencionados previamente. Adicionalmente al

[1] Sitio oficial de IBM Watson: https://www.ibm.com/es-es/watson
[2] Sitio oficial de Google Dialogflow: https://cloud.google.com/dialogflow
[3] Sitio oficial de Rasa: https://rasa.com/

manejar lenguajes simples de programación como Python es ideal como herramienta inicial para crear un chatbot o prototipo preliminar.

Basándose en el artículo “Chat-IBN-RASA: Building an Intent Translator for Packet-Optical Networks based on RASA “(Cesila et al., 2023). Rasa se elige por su flexibilidad y capacidad de auto entrenamiento con datos mínimos, lo que lo hace adecuado para adaptarse a las diversas necesidades en “intent-based networking”. Además, Rasa permite la integración con múltiples fuentes de datos y la creación de acciones personalizadas, lo que facilita la interacción humana y la colaboración con otros componentes del sistema IBN. El artículo argumenta que los chatbots son útiles en IBN ya que proporcionan una interfaz amigable para usuarios no técnicos, permiten interacciones conversacionales, actúan como traductores de “intents” y facilitan la retroalimentación en tiempo real, mejorando la experiencia de usuario.

2.1 Trabajo relacionado

Al igual que en la solución mencionada previamente, el marco Rasa, se presenta como una opción flexible y de autoentrenamiento con datos mínimos. El trabajo relacionado resalta la aplicación de técnicas de NLP para la traducción de intenciones, abarcando desde sistemas de asistencia de voz hasta modelos de secuencia a secuencia. La arquitectura del Traductor de Intenciones Chat-IBN-RASA se sitúa en capas de Red, Intención y Negocios. Utilizando el chatbot basado en Rasa, interactúa con los usuarios para recopilar características de intenciones en un lenguaje natural. A través de la coordinación con el Elemento de Cómputo de Trayectorias y el Módulo de Gestión de Fallos de Red, se logra una implementación eficiente de los requerimientos en la red.

Asimismo, dentro del trabajo “Declarative Provisioning of Virtual Network Function Chains in Intent-based Networks” “(Massa et al., 2023), se logra plantear un enfoque de IBN pero por medio del “intents” realizados a bajo nivel en el lenguaje Prolog. El escenario

motivador presentado en el documento involucra a un interesado, como un desarrollador de juegos, expresando la intención de ofrecer una aplicación de juegos asistida por el chatbot. El escenario aborda problemas como el retraso, las demoras y los gráficos deficientes experimentados por los jugadores en sus dispositivos móviles al jugar. Para resolver estos problemas, el interesado planea aprovechar la computación en el borde para proporcionar tiempos de respuesta más rápidos, reducir la latencia y mejorar los niveles de gráficos. Prolog se utiliza en redes basadas en intenciones para manejar intenciones declarativas. Permite modelar y procesar intenciones de aprovisionamiento de servicios basadas en funciones de red virtual y su implementación. El lenguaje de programación lógica Prolog se utiliza para declarar intenciones expresadas por el usuario y razonar automáticamente sobre cómo cumplirlas. Este enfoque permite identificar soluciones que cumplen con las expectativas de la intención, dando lugar a varias ubicaciones posibles. Los programas Prolog pueden consultarse, y el intérprete de Prolog intenta responder a cada consulta aplicando una resolución lineal definida selectiva y devolviendo soluciones. Además, se utiliza la inferencia de Prolog para traducir intenciones en especificaciones de aprovisionamiento, y la naturaleza declarativa del lenguaje proporciona un nivel adecuado de abstracción que se ajusta de forma adecuada con los objetivos de las redes basadas en intenciones.

3. DESARROLLO DEL TEMA

3.1 Descripción de la propuesta:

Para solucionar los desafíos descritos previamente, la propuesta se basa en la implementación de un chatbot basado en el framework de inteligencia artificial Rasa. En este caso se plantea el objetivo de poder reconocer requerimientos e intents con un conjunto de datos limitados sobre el cual el framework se puede entrenar, así como la traducción a nivel bajo de los requerimientos planteados con el fin de poder desplegar los requerimientos por medio de un procesamiento que logre llevar a cabo este proceso y automatizar el proceso de traducción existente.

La implementación del chatbot se plantea con el objetivo de interactuar efectivamente con el usuario, proporcionando una interfaz amigable y facilitando la expresión de requerimientos en lenguaje natural. Este enfoque, basado en la integración de tecnologías de procesamiento del lenguaje natural, se presenta como una solución práctica para superar las barreras asociadas con la complejidad técnica y facilitar la comunicación entre el usuario y el sistema IBN.

La propuesta aborda de manera integral los desafíos del paradigma de Intent-Based Networking (IBN), lidiando con problemas como traducción de “requerimientos”, aseguramiento y despliegue de requerimientos. Sin embargo, a medida que el IBN ha evolucionado, ha surgido una problemática considerable en la transición y reconocimiento de configuraciones detalladas. La transición de las intenciones del usuario a configuraciones específicas de dispositivos de red se convierte en un desafío operativo, siendo un cuello de botella debido a la falta de herramientas intuitivas. La propuesta busca abordar no solo la problemática de la configuración detallada, sino también proporcionar una gestión integral para hacer que el IBN sea más accesible y utilizable para una audiencia más amplia que los

profesionales de redes. Se plantea esta solución como respuesta igualmente a los diferentes problemas que pueden presentarse al momento de llevar a cabo la traducción, lo cual es sumamente importante en este tipo de sistemas: “One key piece to deliver the envisioned degrees of abstraction in IBN architectures is a suitable translation component to convert natural language into lower-level configurations and commands” (Cesila et al., 2023, p. 534), planteando a la traducción a niveles bajos como un mecanismo angular de los sistemas de IBN. El prototipo desarrollado tiene como objetivo resolver problemas básicos de IBN como la identificación y traducción, así como la finalidad de tener una plataforma que permita la visualización de la posible creación y despliegue de intents.

3.2 Descripción y Funcionamiento de Rasa

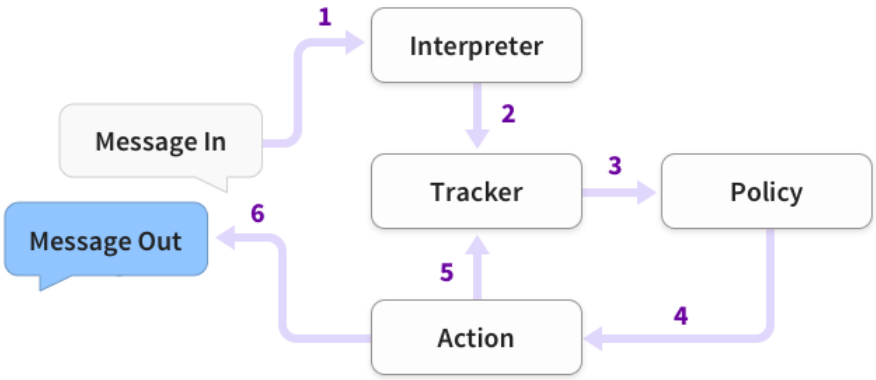

Ilustración 1: Flujo de Framework RASA.

Para entrenar un modelo de chatbot Rasa se utilizan diferentes archivos que van a definir el comportamiento y respuestas del chatbot. El caso de uso de interacción entre usuario y chatbot se basa en el flujo de la ilustración 1. El proceso de funcionamiento de un sistema de inteligencia artificial conversacional en Rasa consta de varios pasos. Comienza con la entrada del usuario, generalmente en forma de un mensaje o texto. Luego, el componente de interpretación (Interpreter) entra en juego, siendo responsable de procesar la entrada del usuario y extraer la información relevante, como intenciones y entidades. A continuación, el

Tracker registra el estado de la conversación, manteniendo un historial que incluye las interacciones pasadas. Las políticas (Policies) son cruciales en Rasa, ya que determinan las acciones que el chatbot debe realizar en función del contexto actual. Utilizando políticas basadas en aprendizaje automático, como Memoization[4] y Keras[5], se predice la próxima acción del bot. La ejecución de esta acción se lleva a cabo a través del mecanismo de acción, que puede incluir el envío de mensajes al usuario, llamadas a API u otras operaciones personalizadas. Finalmente, el bot responde al usuario basándose en la acción ejecutada, cerrando así el ciclo de interacción. Los archivos principales que se editan y donde se define el comportamiento del chatbot y los datos de entrenamiento son los siguientes:

1. domain.yml: Se registra las acciones, ejemplos de entrenamiento y posibles respuestas del chatbot.

2. config.yml: Configura componentes e hiper parametros de entrenamiento de framework.

3. data/nlu.yml: Contiene ejemplos de entrenamiento sobre el cual el framework aprendera a reconocer requerimientos.

4. data/stories.yml: Define diferentes flujos y caminos conversacionales, para que el chatbot logra tener un marco establecido con relación a que respuesta tomar a cabo según el “intent” o el pedido del usuario.

5. actions.py: Logra manejar procesamiento y funciones adicionales para manejar datos.

6. endpoints.yml: Conecta el chatbot a otros servicios, o puntos

[4] Memoization: Técnica de optimización que almacena resultados de funciones costosas, evitando recálculos al encontrarse los mismos inputs. Mejora la eficiencia, común en programación dinámica y algoritmos recursivos.
[5] Keras: Interfaz de alto nivel para redes neuronales en Python, diseñada para ser fácil de usar y flexible. Actúa como interfaz para TensorFlow, Theano y CNTK, facilitando la definición y entrenamiento de modelos de aprendizaje profundo.

Se aprovecha las ventajas ofrecidas por Rasa para personalizar el chatbot haciendo uso de los diferentes archivos y comandos que ofrecen flexibilidad y tecnología avanzada: "(..) features like slots, forms, supervised interactive learning, api integration, and database makes it a complete framework that can be used to perform highly complex tasks." (Sharma & Joshi, 2020, p. 1014). Se aprovecha diferentes características de Rasa con el fin de llegar a desplegar y atender los requerimientos planteados por el usuario o administrador de red.

En este caso, para empezar a entrenar al chatbot con conceptos de red se hizo cambios en el archivo de domain.yml, donde se registraron las acciones que se van a llevar a cabo según los prompts ingresados por el usuario, adicionalmente se registra el nombre de los "intents" especificados en el archivo nlu.yml y las entidades que se van a extraer en cada "intent" (tipos de dispositivos, numero de dispositivos, etc).

En el archivo nlu.yml se van a definir diferentes "intents" y formas en las cuales el usuario va a pedir la creación de redes y los conceptos básicos con diferentes expresiones naturales de lenguaje. En el archivo actions.py se definirán diferentes acciones personalizadas para definir el comportamiento del chatbot, el archivo en Rasa desempeña un papel crucial en la implementación de acciones personalizadas en los asistentes virtuales creados con esta plataforma. En esencia, este archivo contiene la lógica y el código necesario para ejecutar tareas específicas en respuesta a las intenciones detectadas y a la información extraída de los mensajes del usuario. Desde el manejo de solicitudes de API hasta la manipulación de bases de datos, las acciones definidas en actions.py permiten que el asistente realice acciones más allá de simplemente entender el lenguaje natural.

El manejo e interacción del framework se lo hace por medio de comandos de consola, que logran ejecutarse una vez que se instala este dentro del dispositivo deseado. Siempre para

probar el prototipo se levanta el servidor de actions.py por medio del comando rasa run actions y se procede a interactuar con el modelo por medio del comando rasa Shell, rasa train y test hacen entrenamiento y pruebas respectivamente. Se puede añadir argumentos adicionales en los diferentes casos. Se hace manejo de paquetes, libreras y dependencias por medio de un ambiente virtual Anaconda, evitando conflicto de dependencias y logrando instalar los diferentes requerimientos para la creación del framework (Tensorflow, Numpy, Pandas, etc.)

3.3 Descripción de Pipeline de funcionamiento

Con el fin de desarrollar el prototipo, se llevó a cabo un análisis a profundidad de los componentes del chatbot, clasificadores, formas de integrar “intents” y sus componentes, es decir la identificación y la traducción de “intents” a niveles bajos de clasificación. Con el objetivo de demostrar el despliegue efectivo de diferentes arquitecturas y configuraciones de red, se usa la herramienta VNX (Virtual Networks over LinuX). Esta herramienta permite la emulación de redes mediante tecnologías de virtualización.

Buscando obtener un proceso ordenado y que logre automatizar los diferentes componentes que se llevan a cabo en un proceso de reconocimiento de “intents” se logra construir un pipeline de los diferentes procesos llevados a cabo para tener un sistema IBN exitoso, como es visto en la ilustración 2.

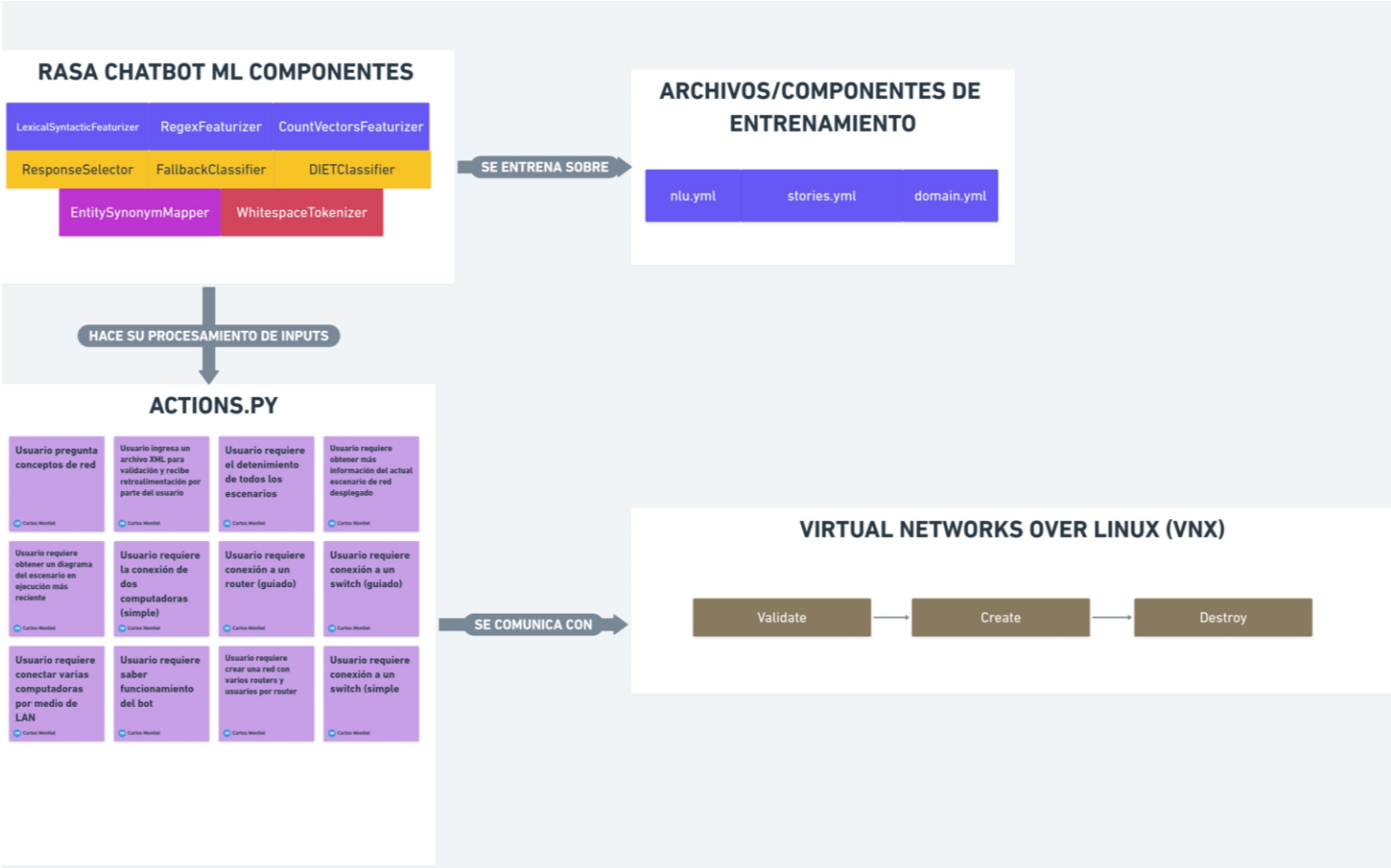

Ilustración 2: Representación Gráfica de Pipeline del prototipo y su funcionamiento.

Con el objetivo de obtener mayor exactitud, para poder abordar diferentes escenarios y casos de uso y lograr que con diferentes tipos de “prompts” el “intent del usuario sea

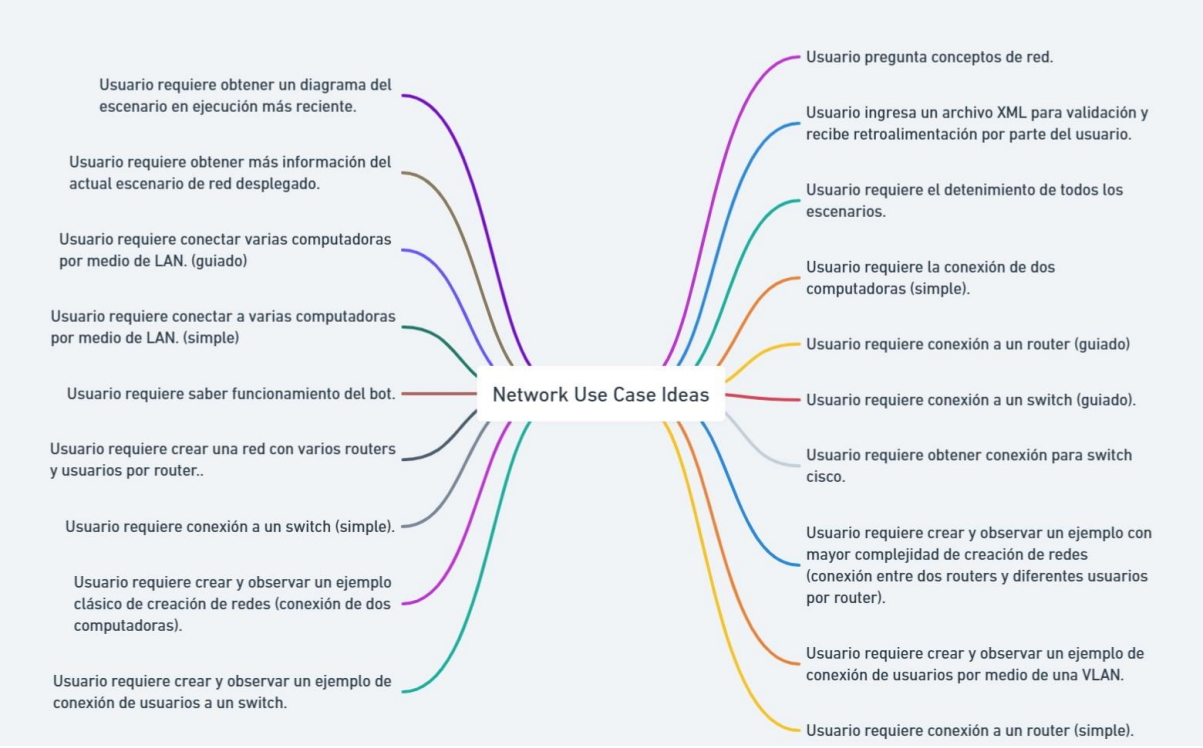

Ilustración 3: Mind-Map de casos de uso para el prototipo

reconocido, se configura dentro del archivo stories.yml, 19 tipos de flujos conversacionales diferentes o “stories”, los cuales definen los tipos de “intents” y respuestas por parte del chatbot que se llevaran a cabo según el caso detectado y guiaran y orientaran al usuario a la creación y despliegue de redes desde instrucciones básicas de alto nivel. Los flujos conversacionales creados se pueden observar en la ilustración 3.

Un flujo de conversación puede ser definido por medio de “intents” (pedidos propios del usuario con sus requerimientos) y “actions” (respuesta establecida para un “intent” o requerimiento detectado). Un flujo de conversación está compuesto de “intents” reconocidos por el framework y especificados dentro de este y respuestas o acciones de respuesta registradas dentro del framework y personalizadas según el desarrollador.

```
- intent: connect_two_computers_request
  examples: |
    - I want to connect two computers.
    - I want to connect two users in a network.
    - Connect two hosts.
    - I want to connect two hosts.
    - Simply make a connection between two computers.
    - I want to connect two computers in a network.
    - Make a connection between two computers
    - Hey, help me link up two computers!
    - I need to connect two users in a network, can you assist?
    - Any chance you can link two hosts together for me?
    - I'm looking to establish a connection between two hosts, can you guide me?
    - Could you assist in making a connection between two computers?
    - connect two computers.
    - connect two computers through a lan network.
```

Ilustración 4: Estructura de un "intent" dentro del archivo nlu.yml

El archivo nlu.yml define los ejemplos de entrenamiento para diferentes “intents”, es decir diferentes expresiones en lenguaje natural en las cuales se basa un mismo requerimiento o formas de expresar un requerimiento. Se logra especificar 30 formas de “intents” dentro de este archivo, cada uno con un mínimo de 8 ejemplos semánticos o diferentes formas de expresión de este mismo “intent”. Si un "intent" incluye información crucial para establecer una red, como el número de usuarios, dispositivos necesarios o tipo de dispositivo, se definen estas entidades de manera especial en los ejemplos de "intents". Esto permite que el clasificador identifique la entidad específica proporcionada en la solicitud del usuario. La ilustración 4 demuestra la estructura de un “intent” o requerimiento y diferentes formas de pedir este requerimiento según el formato del archivo.

Para establecer las acciones o respuestas que el chatbot realizara se debe definir dos tipos de acciones: acciones estándar y acciones personalizadas. Se refiere a acciones estándar, las acciones que están definidas dentro del archivo domain.yml, las cuales no requieren de

procesamiento adicional programático y que simplemente son conjuntos de textos que pueden ser utilizados en respuestas predeterminadas. Por otro lado, las acciones personalizadas son acciones específicas del chatbot que requieren lógica de procesamiento adicional y, por lo tanto, se implementan a través de código personalizado, estas están definidas dentro del archivo actions.py. Al combinar ambos tipos de acciones se busca ampliar la gama de formas de respuesta del chatbot, llevando a cabo procesamiento adicional que necesita código de Python.

Ilustración 5: Primer bloque de acciones de procesamiento.

Las acciones llevadas a cabo tienen un desencadenante y el procesamiento es diferente según la acción, entidad y requerimiento detectado. Detalles de las diferentes acciones se puede

observar en la ilustración 5 y 6, ejemplificando desencadenantes y funcionalidades de una serie de acciones tomadas como ejemplo que procesaran el ingreso de datos de usuario después del reconocimiento del “intent”.

Ilustración 6: Segundo bloque de acciones de procesamiento.

Con el fin de lograr que el chatbot logre identificar “intents”, separar entidades y responder acorde a el “intent” proporcionado se usa un pipeline con diferentes componentes de machine learning que ayudaran a preprocesamiento y procesamiento de datos para entrenar al chatbot con los ejemplos propuestos dentro del archivo nlu.yml. El modelo utilizado va a

hacer uso de componentes y políticas de entrenamiento planteados de forma secuencial en la ilustración 7.

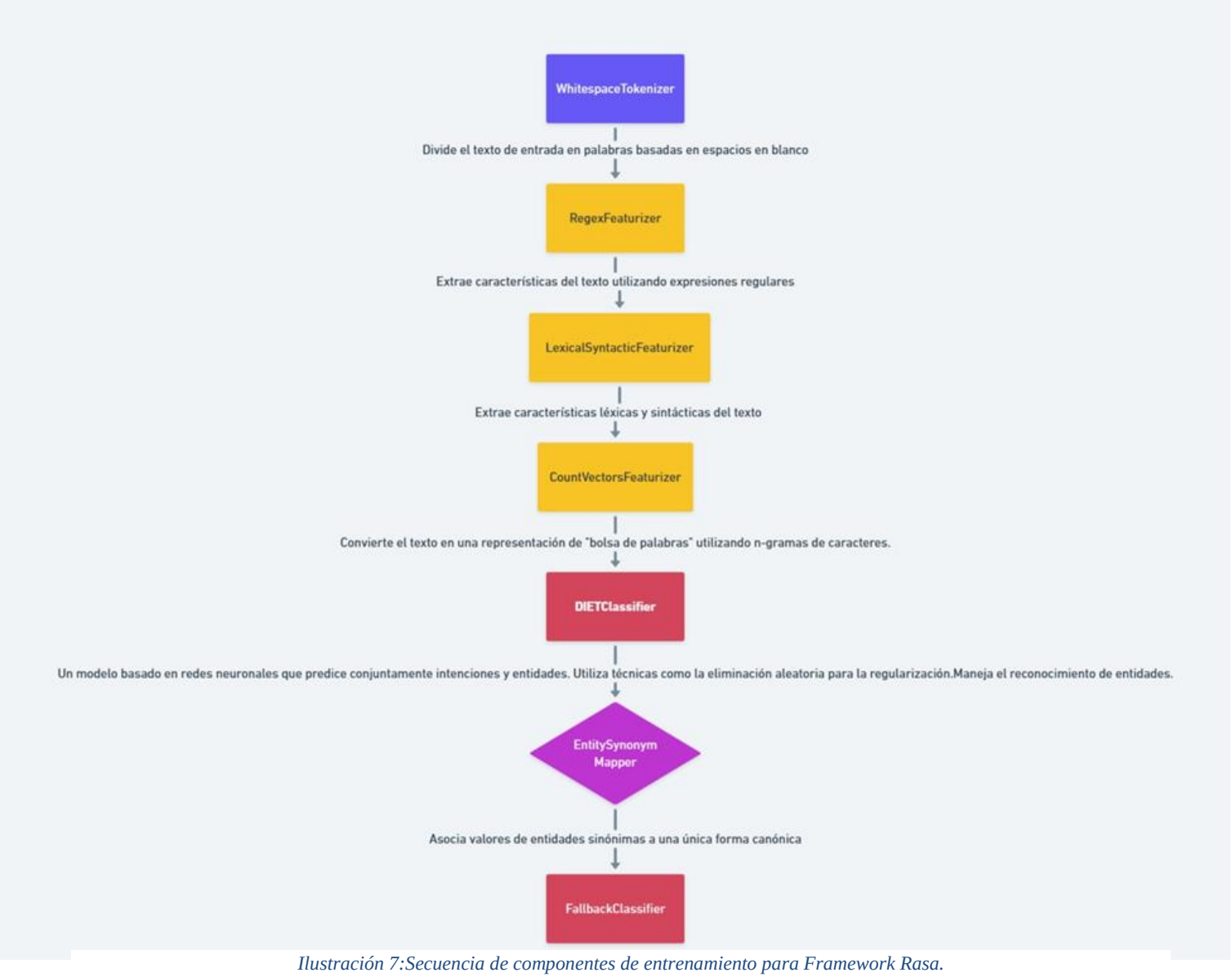

Ilustración 7:Secuencia de componentes de entrenamiento para Framework Rasa.

El componente inicial, WhitespaceTokenizer propio de Rasa, divide el texto de entrada en tokens individuales según los espacios en blanco, creando los elementos fundamentales para un análisis posterior. A continuación, RegexFeaturizer aprovecha expresiones regulares para identificar y extraer patrones específicos o características dentro del texto, ayudando en el reconocimiento de información estructurada. Posteriormente, LexicalSyntacticFeaturizer contribuye capturando tanto las estructuras léxicas como sintácticas de la entrada,

proporcionando un contexto más rico para el procesamiento subsiguiente. Las dos instancias de CountVectorsFeaturizer desempeñan un papel clave al convertir el texto en representaciones numéricas, y la segunda instancia se centra en n-gramas de caracteres, ofreciendo un análisis más detallado de las características a nivel de caracteres.

DIETClassifier es un componente basado en redes neuronales que realiza tareas duales de clasificación de intenciones y reconocimiento de entidades. Utiliza una arquitectura basada en transformers e incorpora mecanismos de eliminación aleatoria (drop_rate) para mejorar la generalización y prevenir el sobreajuste. La restricción de similitudes entre ejemplos durante el entrenamiento perfecciona aún más su capacidad para distinguir entre diferentes intenciones y entidades. (Bunk, et al., 2020). En la fase de featurization, se realiza la tokenización de las oraciones de entrada, aplicando características dispersas y densas como codificaciones one-hot e incrustaciones preentrenadas, esta última sometida a dropout para evitar sobreajustes. La capa completamente conectada iguala las dimensiones de estas características. Un transformador de dos capas con atención de posición relativa codifica el contexto a lo largo de la oración. Para el reconocimiento de entidades nombradas, se implementa una capa de etiquetado Conditional Random Field (CRF), la cual ayuda a identificar dependencias en una secuencia de caracteres considerando el contexto, sobre la secuencia de salida del transformador, evaluando la capacidad de la arquitectura para predecir etiquetas de entidades mediante la log-verosimilitud negativa del CRF. En lo que respecta a la clasificación de intenciones, la arquitectura utiliza la salida del transformador para el token de clasificación y las etiquetas de intenciones, generando un espacio vectorial semántico con una pérdida de producto punto. Durante la inferencia, la similitud de producto punto clasifica las posibles etiquetas de intenciones. Se incorpora un

objetivo de entrenamiento adicional inspirado en el modelado de lenguaje enmascarado como regularizador, permitiendo que el modelo aprenda características más generales del texto.

EntitySynonymMapper maneja expresiones sinónimas al mapearlas a una representación común, asegurando consistencia en la comprensión de las entradas del usuario.

ResponseSelector desempeña un papel crucial en la recuperación de respuestas, empleando técnicas de aprendizaje automático para seleccionar una respuesta apropiada según el contexto y la entrada del usuario. Finalmente, FallbackClassifier es un mecanismo de seguridad que predice acciones de respaldo cuando la confianza del modelo está por debajo de cierto umbral, permitiendo que el sistema maneje de manera elegante situaciones en las que el modelo está inseguro o se enfrenta a consultas fuera de su alcance.

La política TED (Transformers Embedding for Dialogue) en el archivo config.yml de Rasa desempeña un papel fundamental en la gestión del diálogo. Basada en arquitecturas de transformers, la política TED, con sus parámetros ajustables como max_history y epochs, se centra en prever las acciones futuras del usuario considerando un historial específico de la conversación. Al emplear embeddings de transformers, este componente permite una comprensión contextual avanzada de las interacciones pasadas. La opción constrain_similarities refuerza la capacidad del modelo para discernir entre diversas acciones durante el entrenamiento. La política TED se propone como un bloque de construcción candidato para desarrollar arquitecturas de vanguardia en diversas tareas de diálogo, demostrando su eficacia en comparación con LSTM, una arquitectura de redes neuronales que captura dependencias a largo plazo, en un conjunto de datos de diálogo orientado a tareas (Vlasov et al., 2020).

La arquitectura de transformer, utilizada en la política TED, se basa en un mecanismo de autoatención que atiende dinámicamente a diferentes partes de la historia del diálogo en cada turno. A diferencia de las redes neuronales recurrentes (RNN) tradicionales, los transformers son capaces de desentrañar segmentos de discurso entrelazados y responder de manera apropiada a conversaciones complejas y de varios turnos (Vlasov et al., 2020). La capacidad del mecanismo de autoatención de centrarse selectivamente en elementos informativos en una secuencia hace que los transformers sean adecuados para procesar historias de diálogo con múltiples temas superpuestos (Vlasov et al., 2020).

Los hiperparametros de cada clasificador, tokenizer o política son ajustados de forma manual incremental, probando diferentes combinaciones que busquen obtener el mejor rendimiento del modelo y tratando de optimizar dos métricas importantes como la exactitud (accuracy) y el f1 score. El modelo se entrena modificando el archivo config.yml, el cual tiene la estructura y diferentes componentes de Machine Learning con los hiperparametros correspondientes para cada paso tales como: épocas, tasa de regularización,

Se logra obtener las métricas de rendimiento del modelo separando el dataset de nlu.yml (el cual está estructurado como un archivo con varios intents como el que fue mostrado en una imagen previamente) en un set de entrenamiento y set de prueba (0.8 % y 0.2% correspondiente). Se lleva a cabo el entrenamiento del modelo y las métricas para el DIETClassifier para “intents” y entidades se puede observar por medio de un histograma que clasifica entre predicciones correctas e incorrectas y el nivel de confianza del modelo para determinar y reconocer los “intents” y entidades, como se puede evidenciar en las ilustraciones 8 y 9.

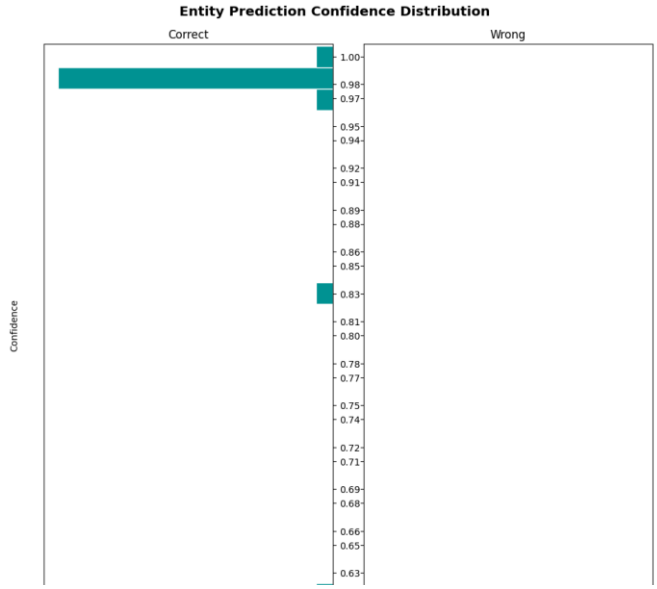

Ilustración 8: Histograma resultante de predicción de Entidades junto con confianza

.

.

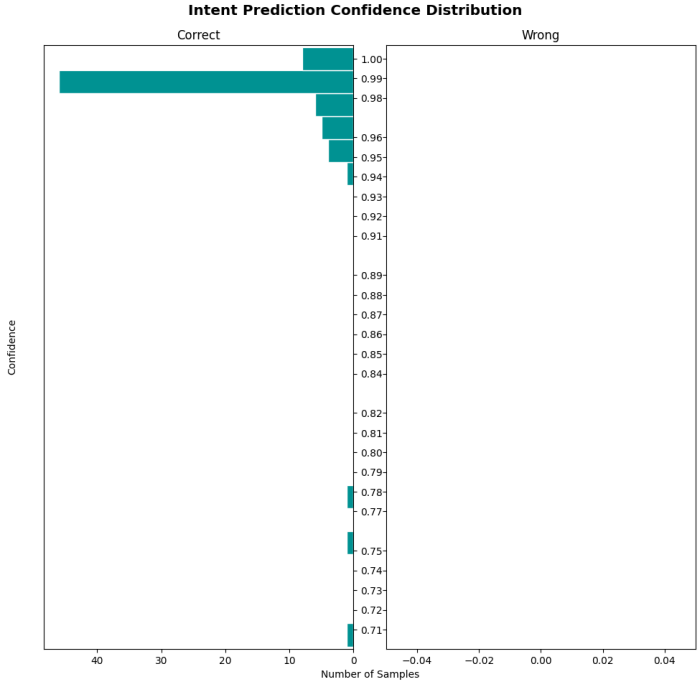

Ilustración 9: Histograma resultante de predicción de "intents" junto con confianza.

Se puede ver que el clasificador logra reconocer los “intents” de manera exitosa, podemos observar que la mayoría de las predicciones correctas tienen una alta confianza, lo que indica que el modelo es bastante seguro en sus predicciones correctas. “Intents” que tienen

pocas muestras de entrenamiento son predichos con una confianza mayor a 0.5, esto es debido a tener pocas muestras de entrenamiento y de prueba del “intent” en cuestión.

En cuanto a la clasificación de entidades llevada a cabo igualmente por DIET classifier se obtiene el siguiente rendimiento en métricas: Accuracy de 0.967 y F1_Score de 0.986. El entrenamiento del modelo es llevado a cabo de manera periódica, considerando que se agregan nuevos ejemplos e intents al dataset.

2.3 Fase Experimental

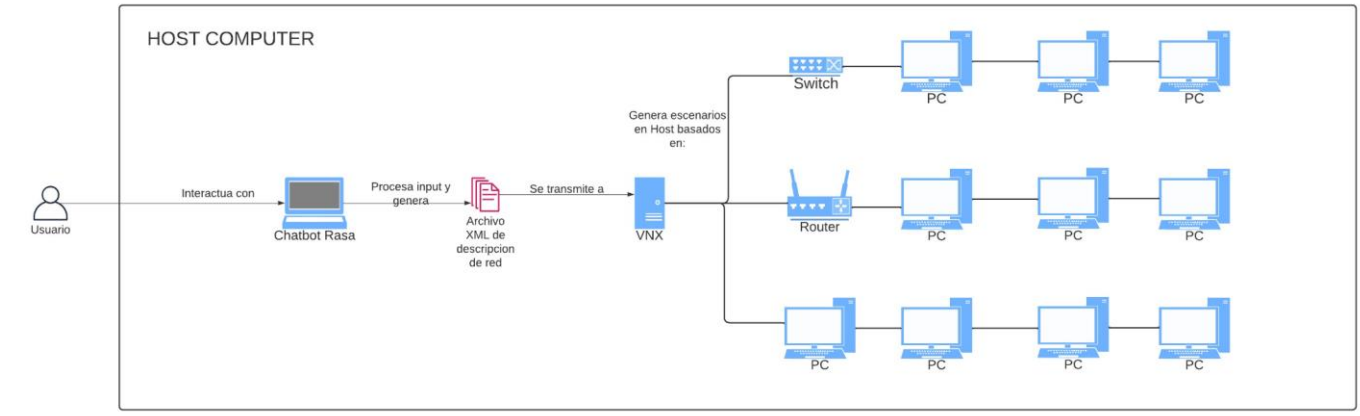

Ilustración 10: Flujo de funcionamiento de prototipo de chatbot

Como se comentó previamente, se usa VNX para llevar a cabo las pruebas de creación y despliegue de “intents”. VNX es una herramienta desarrollada por el Departamento de Ingeniería Telemática (DIT) de la Universidad Politécnica de Madrid (UPM) que permite a los usuarios crear escenarios de red virtuales. Proporciona una forma de administrar bancos de pruebas sin la complejidad de inversión y gestión necesaria para crearlos utilizando equipos reales. Se busca simplificar los aspectos de resolución y aseguramiento de “intents”, por este medio se logra tener un solo administrador de red que logre crear y desplegar las diferentes redes, evitando tener conflictos de envío de “intents” por múltiples usuarios lo cual es un problema clásico en la resolución al momento de que se envía diferentes requerimientos. Adicionalmente VNX provee retroalimentación al usuario al momento de tener

configuraciones de red erróneas o al momento en el cual el despliegue de una red sufre algún tipo de cambio o fallo, siendo útil en el campo del aseguramiento de los “intents”. En este caso en la mayoría de los experimentos usan comandos propios de la herramienta, los cuales ayudan a la creación de escenarios, la pausa de ejecución de escenarios y la validación de la sintaxis de escenarios creados, validando sintaxis, existencia de directorios y virtualización de equipos.

En la primera fase de experimentos, los esfuerzos se centraron exclusivamente en la verificación de archivos XML, específicamente la validación de la correcta sintaxis de la descripción de escenarios de red. Con el fin de que VNX logre desplegar redes virtuales, se necesita un archivo de descripción, el cual va a contener diferentes elementos dentro de diferentes tags XML como maquinas virtuales, dispositivos de red, configuraciones de virtualización, configuraciones de red como IPV4, IPV6, etc. Este proceso de revisión se realiza a través de la plataforma VNX, utilizando el comando -validate-xml, con el propósito de garantizar que cada archivo generado no presente errores. La validación se realiza manualmente, es decir se usa el comando –validate-xml proporcionado por VNX que valida si el archivo XML proporcionado cumple los requisitos establecidos por un archivo base XSD, donde cada archivo producido por el chatbot se somete a este proceso y el comando mencionado no emite mensajes en caso de ser exitoso. Este procedimiento se repite para diversos escenarios, y en el caso de encontrar algún error en la generación de XML, se lleva a cabo una modificación dentro de la acción encargada de generar el archivo correspondiente. Esta fase inicial se enfoca en asegurar la integridad sintáctica de los escenarios antes de avanzar a la siguiente etapa del experimento, proporcionando una base sólida para el desarrollo y la implementación de redes en fases posteriores del proceso. La generación de archivos XML depende del dispositivo, un ejemplo con un router que da conectividad a tres usuarios se puede

observar en la ilustración 11. En este caso se define condiciones generales y de ayuda dentro del tag <global>, las redes se definen dentro del tag <net> escogiendo el tipo deseado de red, finalmente los dispositivos de red y hosts virtuales se definen dentro del tag <vm>, dentro de este se puede configurar las interfaces, rutas y direcciones IP con el fin de establecer la conexión.

```
<?xml version="1.0" ?>
<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
  <global>
    <version>2.0</version>
    <scenario_name>vnx_custom_network_router_3_users</scenario_name>
    <automac/>
    <vm_mgmt type="none"/>
    <vm_defaults>
      <console display="no" id="0"/>
      <console display="yes" id="1"/>
    </vm_defaults>
    <cmd-seq seq="ls12">ls1,ls2</cmd-seq>
    <cmd-seq seq="ls123">ls12,ls3</cmd-seq>
    <cmd-seq seq="ls1234">ls123,ls4</cmd-seq>
    <help>
      <seq_help seq="start-www">Start apache2 web server</seq_help>
      <seq_help seq="stop-www">Stop apache2 web server</seq_help>
    </help>
  </global>
  <net mode="virtual_bridge" name="Net0"/>
  <net mode="virtual_bridge" name="Net1"/>
  <vm name="h1" type="lxc" arch="x86_64">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu64</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.1.0.2/24</ipv4>
    </if>
    <route gw="10.1.0.1" type="ipv4">default</route>
  </vm>
  <vm name="h2" type="lxc" arch="x86_64">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu64</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.1.0.3/24</ipv4>
    </if>
    <route gw="10.1.0.1" type="ipv4">default</route>
  </vm>
  <vm name="h3" type="lxc" arch="x86_64">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu64</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.1.0.4/24</ipv4>
    </if>
    <route gw="10.1.0.1" type="ipv4">default</route>
    <filetree root="/var/www/" seq="start-www"/>
    <exec ostype="system" seq="start-www" type="verbatim">chmod 644 /var/www/*</exec>
    <exec ostype="system" seq="start-www" type="verbatim">service apache2 start</exec>
    <exec ostype="system" seq="stop-www" type="verbatim">service apache2 stop</exec>
  </vm>
  <vm name="r1" type="lxc" arch="x86_64">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu64</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.1.0.1/24</ipv4>
    </if>
    <if id="2" net="Net1">
      <ipv4>10.1.3.1/24</ipv4>
    </if>
  </vm>
  <host>
    <hostif net="Net1">
      <ipv4>10.1.3.2/24</ipv4>
    </hostif>
```

Ilustración 11: Archivo vnx_custom_network_router_3_users. xml

En la segunda fase de experimentación, se procede a la creación de escenarios mediante el comando --create. Este proceso interactivo varía según las especificaciones de cada escenario, desplegando consolas detalladas o emitiendo confirmaciones en la consola principal. El objetivo central es verificar la correcta creación y despliegue de las redes definidas, ajustando parámetros de generación de funciones o virtualización en caso de errores. La interactividad del proceso permite intervenir manualmente para tomar decisiones específicas.

Se realiza una exhaustiva verificación de la conectividad entre nodos y la funcionalidad de los componentes. Durante este proceso, se registran mensajes de error o confirmaciones exitosas para facilitar la resolución de problemas en etapas posteriores. La documentación detallada de cada escenario, incluyendo parámetros y ajustes realizados, es esencial para la replicación y análisis de resultados en futuras iteraciones del experimento. En resumen, esta fase se centra en garantizar la funcionalidad del entorno de prueba, adaptando parámetros según sea necesario para lograr un despliegue exitoso.

La tercera fase de experimentación lleva a cabo la comparación entre el LLM de OpenAI ChatGPT y el prototipo desarrollado. Este proceso nuevamente es interactivo y busca demostrar que la especialización del prototipo de chatbot genera los resultados deseados y despliega redes al tener comandos de automatización con VNX, en comparación a ChatGPT. Esto implica que el prototipo no solo debe ser capaz de comprender y responder preguntas, sino también de desplegar acciones automatizadas utilizando los servicios de VNX. La capacidad de ejecutar comandos específicos y llevar a cabo tareas automatizadas es esencial para evaluar la utilidad práctica del chatbot en un entorno operativo. En comparación con ChatGPT, que es un modelo de propósito general, se espera que el prototipo especializado no solo supere en conocimiento y relevancia en su dominio específico, sino que también demuestre una mayor eficiencia al utilizar los comandos de despliegue con VNX. Esta comparación no solo se centra en la calidad de las respuestas proporcionadas, sino también en la capacidad del chatbot para interactuar de manera efectiva con los usuarios. La diferencia principal entre el prototipo y ChatGPT se basa en el hecho de que la identificación y traducción de intents a bajos niveles de configuración solo se da en el prototipo, ChatGPT logra únicamente identificar el “intent” pero no tiene capacidad de desplegar, resolver y asegurar que un “intent” se esté implementando de forma satisfactoria, al ser un modelo multipropósito. Con el fin de llevar a cabo esta comparación se ingresó “prompts” o requerimientos iguales dentro

de ChatGPT y del prototipo de Rasa, denotando diferentes rangos de respuestas. Dentro de ChatGPT al ser un modelo de NLP, las respuestas del chat en cuestión tienen un rango mayor de detalle en sus respuestas, a pesar de esto ChatGPT no tiene acceso a ejecución o despliegue de configuraciones de red, y la mayor desventaja de este como ya se dijo se basa en que el modelo necesita una mayor cantidad de información y retroalimentación por parte del usuario para dar instrucciones de como desplegar una red de forma física, nunca existe ningún tipo de prueba practica como la que se lleva a cabo en el chatbot desarrollado con Rasa. El chatbot propuesto como solucion puede tener un rango limitado de respuestas y no existe tanto detalle o variabilidad dentro de estas, pero su mayor ventaja es la capacidad de interactuar con las configuraciones y redes desplegadas y no necesitar una mayor cantidad de información o retroalimentación por parte del usuario para identificar el "intent" y desplegarlo en el ambiente de simulación.

Finalmente, la cuarta fase de experimentación se basa en la automatización de la fase 1 y fase 2, es decir que no se necesite explícitamente llamar a los comandos –validate y –create dentro de la consola Linux, si no que automáticamente al reconocer y validar el intent por medio de Rasa, se logre desplegar la configuración deseada en el ambiente de simulación. En este caso dentro del archivo actions.py, se procedió a crear una acción personalizada que ejecuta subprocesos dentro del sistema operativo que siguen el siguiente orden: Se verifica que ningún escenario o configuración se esté ejecutando, en caso de que se encuentre una configuración en estado de ejecución proceda a ser eliminada, en caso de que no se encuentre ninguna ejecución, se procede a empezar el escenario detectado en el intent y sobre el cual se generó la configuración VNX en formato de archivo XML. Esta secuencia de pasos es hecha por medio de los subprocesos previamente mencionados, los cuales hacen llamadas a los comandos –create y –destroy, los cuales gestionan la creación y la eliminación de escenarios respectivamente. Se detecta los escenarios pasados ejecutados por medio de un archivo

histórico en formato txt, donde la penúltima línea representa el escenario o configuración ejecutado previamente y la última línea representa el escenario o configuración a ser ejecutado o recientemente identificado como “intent”. De esta forma se logra la automatización y adicionalmente se logra tener trazabilidad de los diferentes escenarios ejecutados, sin mencionar que esto logra solucionar de forma estricta los problemas que pueden surgir al momento de la resolución y activación, ya que se está asegurando de que un solo usuario tenga la capacidad de activar un solo intent a la vez, evitando colisiones de “intents” y adicionalmente asegurando el hecho de que un solo administrador de red pueda modificar los “intents” según lo requerido. Las pruebas llevadas a cabo en esta fase consisten en verificar el despliegue de máquinas virtuales en los diferentes escenarios planteados con un numero variado de switches, routers, etc., comprobando el funcionamiento de las diferentes máquinas y de la conectividad establecida por medio de PING.

En resumen, las diferentes fases de experimentación son pasos progresivos que permiten que exista un proceso de verificación secuencial y que dan la posibilidad al desarrollador de verificar que no va a existir un error posible en el pipeline que permite la automatización del proceso de validación y ejecución de un “intent”. Se verifica la integridad sintáctica, funcionalidad y eficiencia del sistema, asegurando la correcta implementación de "intents" en el entorno de simulación. Este enfoque progresivo facilita la identificación y corrección de posibles errores en el pipeline de automatización, brindando confianza en la ejecución exitosa de "intents" para desplegar la red virtual. Los resultados de los diferentes prompts con diferentes topologías y generaciones automáticas de consola y del prototipo final se pueden evidenciar en las ilustraciones

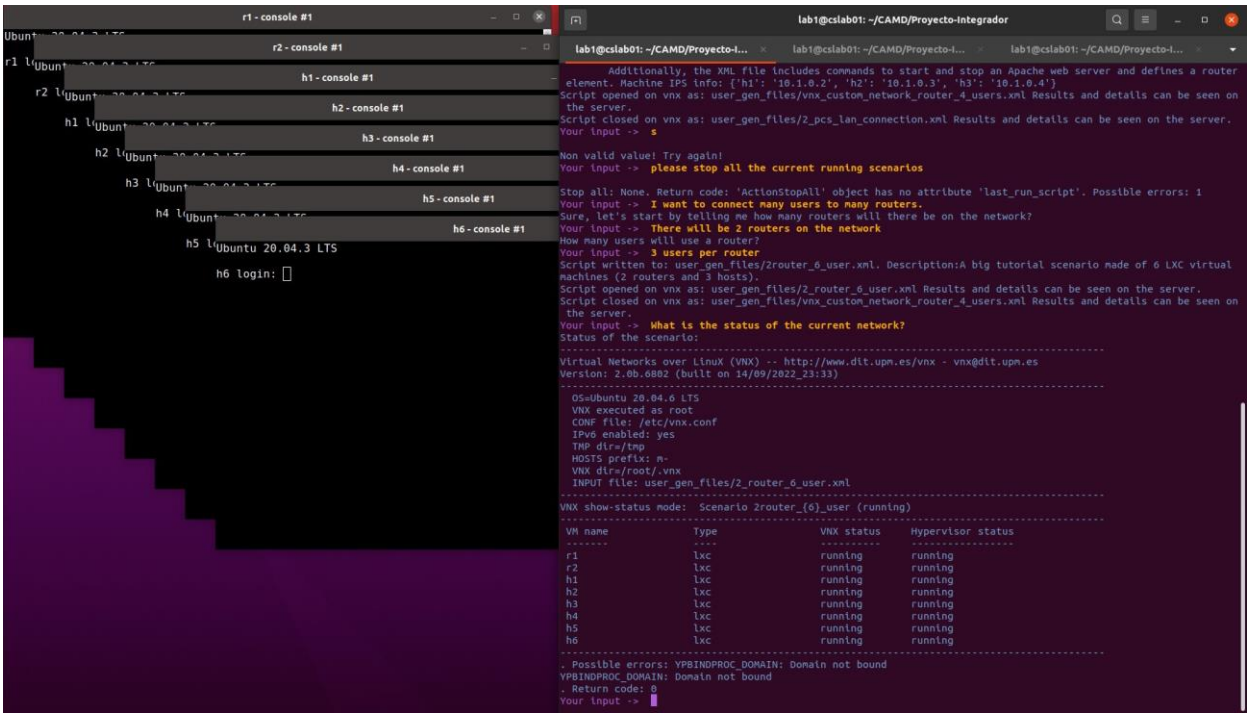

Ilustración 13:Prompts y generación de escenario con 2 routers y 3 usuarios por router.

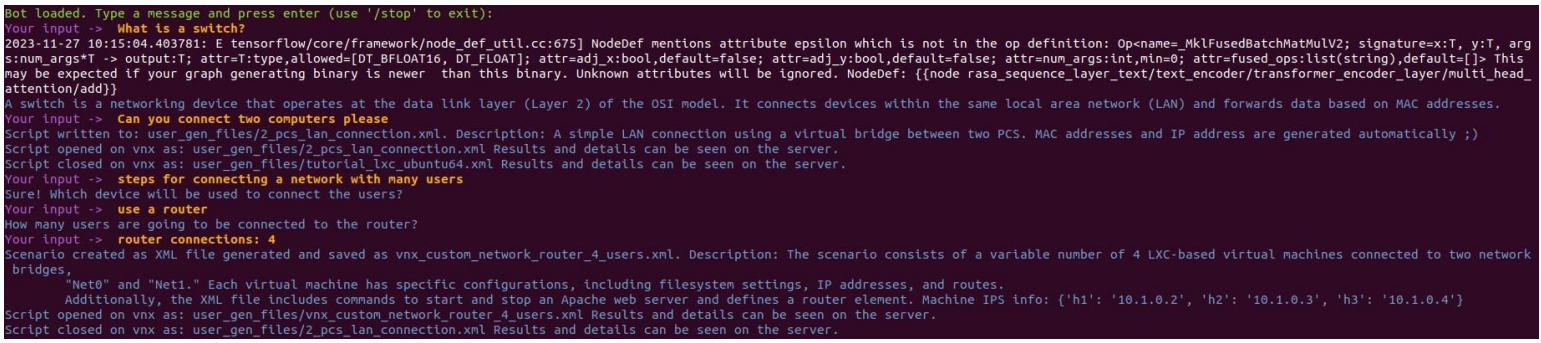

Ilustración 12: Ejemplo de prompts y respuesta de chatbot.

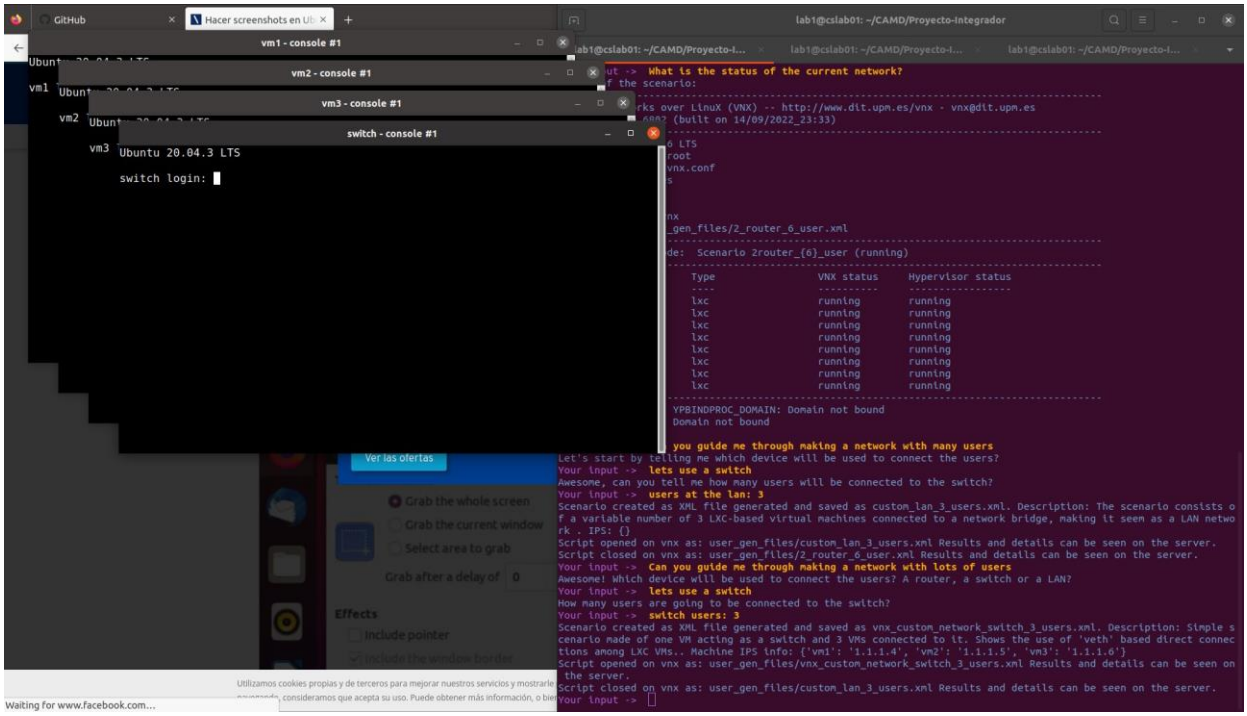

Ilustración 14: Prompts y generación de conexión de 3 usuarios con un switch.

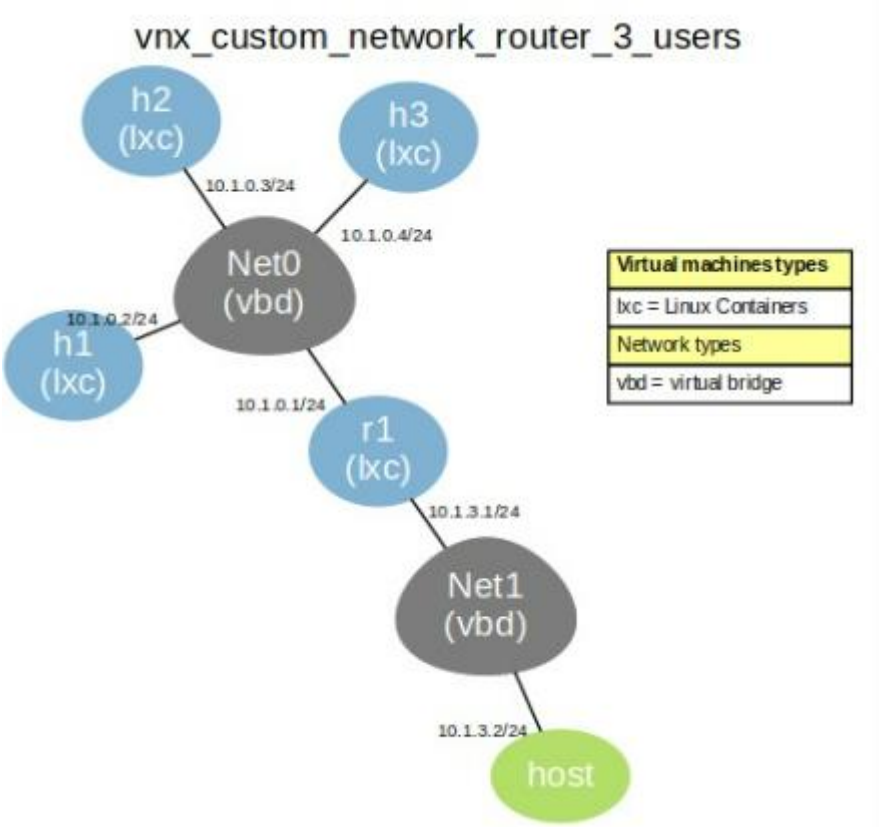

Ilustración 15: Topología generada de conexión entre un router, 3 hosts..

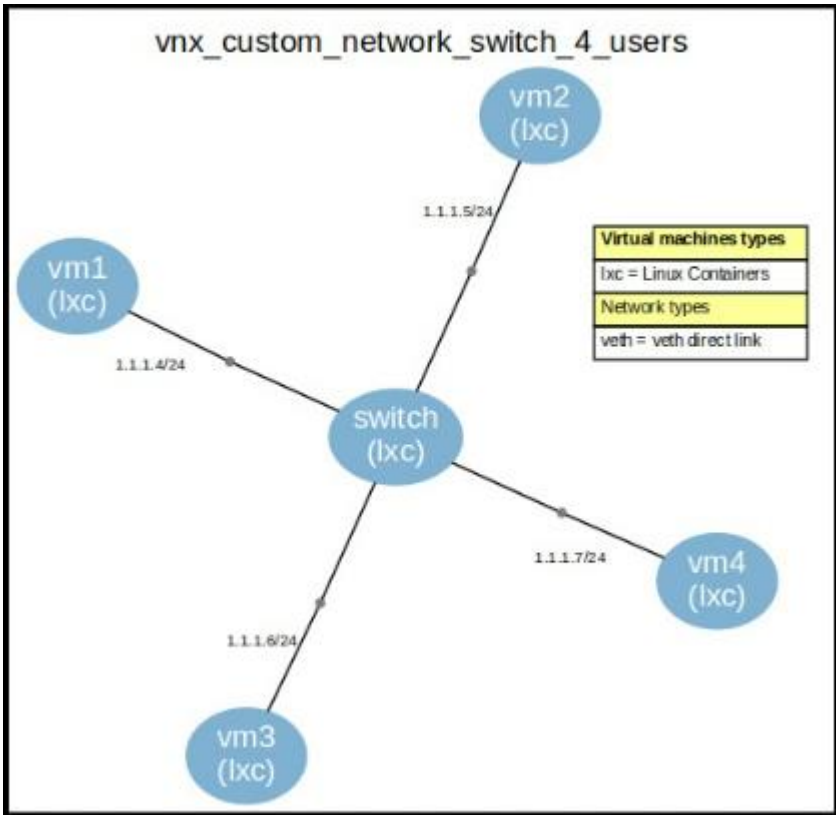

Ilustración 16: Topología generada de conexión entre un switch y 4 hosts.

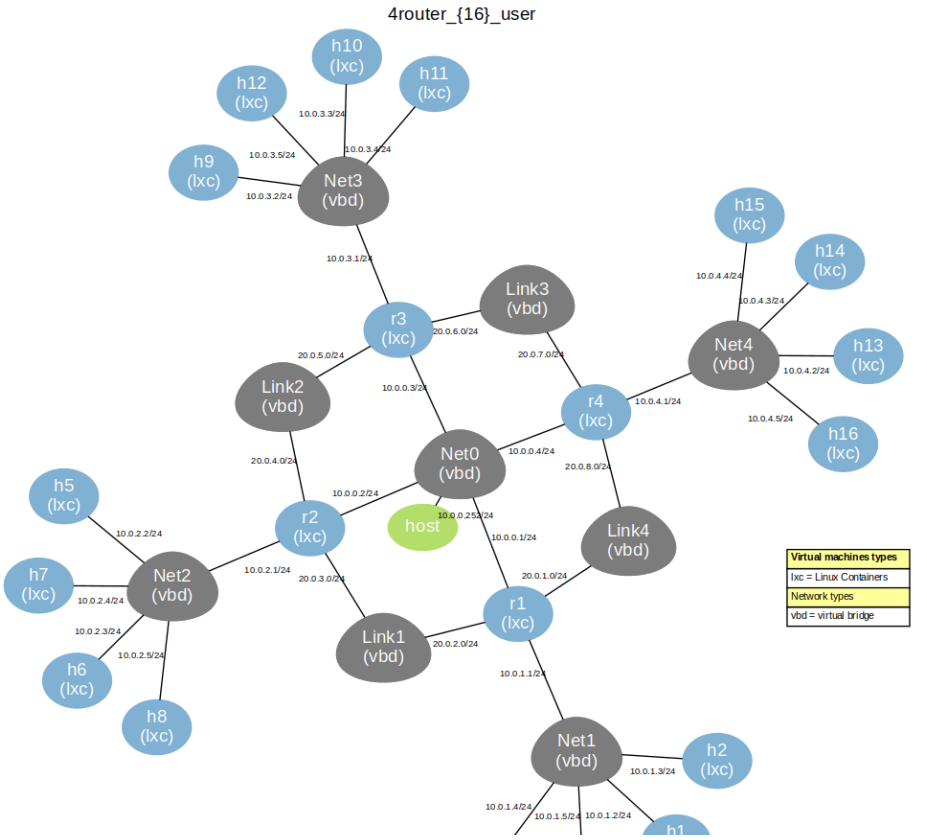

Ilustración 17:Opción de topología redundante, creada en caso de 4 routers y 4 usarios por router

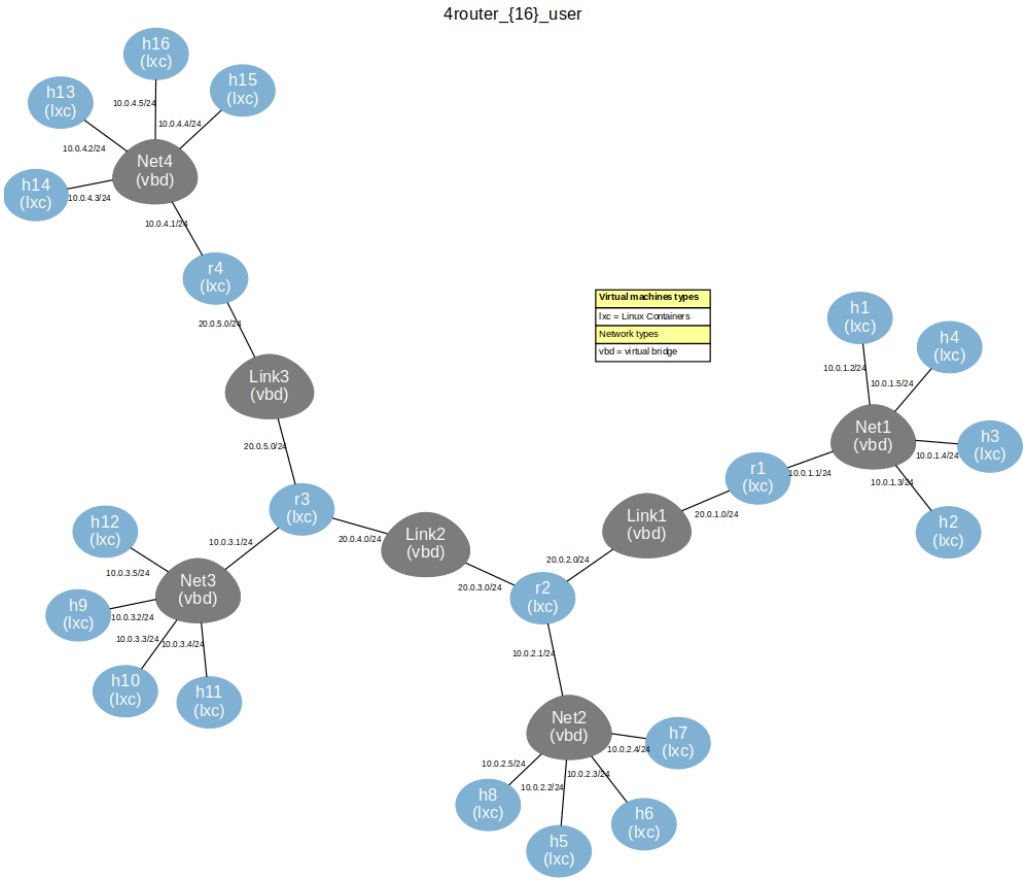

Ilustración 18: Opción de topología secuencial, creada en caso de 4 routers y 4 usuarios por router

4. CONCLUSIONES

4.1 Objetivos Logrados

El prototipo desarrollado cumple su objetivo principal de identificar, traducir, asegurar y desplegar "intents". El prototipo hace uso de frameworks y herramientas actuales, especialmente el uso de Rasa permite que los experimentos relacionados a la validación, ejecución y despliegue de un "intent" sean exitosos. Se cumple el objetivo de reconocer "intents", las entidades relevantes dentro del "intent" y logra desplegar una vez reconocido el "intent", la configuración deseada por el usuario administrador de red. Se logra plantear soluciones dentro de la plataforma que logran hacer uso de técnicas de Machine Learning con el fin de identificar simples configuraciones de usuario sin la necesidad de que este tenga un

conocimiento ya previo de aspectos de redes detallados como IP, subnetting, direcciones MAC, tipos de conexiones y tipos de dispositivos de conexión. Adicionalmente, la interfaz conversacional logra ser una herramienta útil que provee retroalimentación al usuario oportuna y que logra detallar diferentes configuraciones de red en términos que puedan ser entendibles para un usuario con pocos conocimientos sobre redes de comunicaciones.

El uso de herramientas y componentes de Machine Learning, facilitan el aprendizaje de diferentes casos y de diferentes formas de expresar un requerimiento o "prompt", con el fin de que el chatbot logre predecir e identificar los requerimientos y tomar la acción acorde según el dataset sobre el cual fue entrenado. Este tipo de componentes son diseñados específicamente para ser entrenados sobre ejemplos específicos, al ser el set de entrenamiento ejemplos de red, el prototipo va a ser eficiente en el despliegue y reconocimiento de "intents" relacionados con redes. Al crear diferentes marcos de flujos conversacionales, se da la oportunidad al usuario de plantear requerimientos de diferentes formas y obteniendo los resultados deseados.

El ambiente de despliegue escogido (VNX) es apto para dar una perspectiva de diferentes comportamientos de red, variando el número de dispositivos y el tipo de dispositivo, resultando en una red virtualizada que se comporta igual que una red física. La ejecución de diferentes configuraciones y escenarios de red por medio de consolas virtuales es un proceso interactivo y didáctico que demuestra visualmente al usuario como funcionaria el requerimiento deseado al momento de ser desplegado, el proceso de la comunicación entre dispositivos y la diferencia de rendimiento y de configuraciones a nivel bajo según el número de dispositivos y dispositivo de red deseado.

La interfaz conversacional es una solución versátil y oportuna, acoplada junto con los componentes y modelos de Machine Learning, logra ser la mejor forma de comunicar y tratar de plantear "intents" o requerimientos. Al tener diferentes modelos de LLM en la actualidad, que demuestran la falta de capacidad de ejecutar código, el prototipo propuesto es una

alternativa novedosa y que logra distinguirse de la amplia gama de chatbots o de modelos de LLM, que a pesar de lograr obtener respuestas variadas y poder responder a diferentes “prompts” según el entrenamiento hecho, al momento de aplicar lo generado en el ámbito programático, no son posibles y no tienen la capacidad de ejecutar este código, dando al cliente únicamente la opción de generar y validar el código, pero relegándose a la posibilidad de que el usuario obligatoriamente tenga que probar el código por si mismo, lo cual puede ser en ciertos casos una tarea sobre la cual este no tenga conocimiento previo.

4.2 Trabajo Futuro

Al evidenciar el funcionamiento correcto del prototipo desarrollado y verificar que es una solución factible para la problemática actual de plantear, traducir, desplegar y asegurar requerimientos de red, se puede plantear avances a futuro con el fin de obtener variedad en funcionalidad del chatbot y acoplar diferentes tecnologías que pueden contribuir a mejorar la experiencia del usuario y la identificación y traducción de requerimientos.

Con el fin de mejorar la experiencia del usuario se puede implementar adicionalmente una interfaz de usuario, que permita una interacción más intuitiva y amigable. Esta interfaz podría incorporar elementos visuales atractivos, como iconos representativos y colores vibrantes, para facilitar la comprensión y navegación por parte del usuario.

Asmismo, con relación al trabajo futuro, una expansión prometedora para el chatbot involucra la implementación de reconocimiento de imágenes, específicamente enfocado en la identificación de topologías de redes. Esta capacidad permitiría al chatbot analizar imágenes de diseños de red proporcionadas por los usuarios, extrayendo información crucial sobre la configuración y la disposición de los componentes de la red. A través de algoritmos avanzados de reconocimiento de patrones y aprendizaje profundo, el chatbot podría interpretar la topología de la red, identificar posibles puntos de congestión o vulnerabilidades, y ofrecer

recomendaciones para optimizar o fortalecer la infraestructura. Esta funcionalidad va más allá de la mera comprensión del texto y enriquece la interacción al integrar elementos visuales, mejorando significativamente la capacidad del chatbot para comprender y abordar los desafíos específicos relacionados con la arquitectura de red.

Otro aspecto a considerar en los avances futuros es la expansión de las capacidades lingüísticas del chatbot. Implementar un sistema de procesamiento del lenguaje natural más avanzado permitiría al chatbot entender y responder de manera más efectiva a consultas complejas y expresiones idiomáticas. El análisis de sentimientos proporcionaría al chatbot la capacidad de reconocer las emociones y estados de ánimo del usuario, permitiéndole ajustar su tono y respuestas de manera más empática y personalizada. Cuando se aplica al contexto de las redes, el análisis de sentimientos podría desempeñar un papel crucial en la identificación de las intenciones del usuario en relación con la infraestructura de red. Por ejemplo, al analizar las consultas del usuario, el chatbot podría determinar si el tono de la solicitud indica una urgencia, insatisfacción o simplemente una búsqueda informativa. Esta información emocional podría influir en la priorización de las respuestas y en la forma en que el chatbot aborda las necesidades del usuario en el contexto de la red. La especialización y ejecución del código se puede expandir por medio de la adaptación de un modelo de LLM que logre dar un rango mayor de respuestas y que sea entrenado sobre un dataset sobre conceptos de red, el cual al momento de escribir esta publicación no está disponible.

REFERENCIAS BIBLIOGRAFICAS

Agarwal, B., Bhagwan, R., Das, T., Eswaran, S., Padmanabhan, V. N., & Voelker, G. M. (2009, April). NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge. In *NSDI* (Vol. 9, pp. 349-364).

Bunk, T., Varshneya, D., Vlasov, V., & Nichol, A. (2020). Diet: Lightweight language understanding for dialogue systems. *arXiv preprint arXiv:2004.09936*.

Cesila, C. H., Pinto, R. P., Mayer, K. S., Escallón-Portilla, A. F., Mello, D. A., Arantes, D. S., & Rothenberg, C. E. (2023, June). Chat-IBN-RASA: Building an Intent Translator for Packet-Optical Networks based on RASA. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)* (pp. 534-538). IEEE.

Leivadeas, A., & Falkner, M. (2022). A survey on intent based networking. *IEEE Communications Surveys & Tutorials*.

Massa, J., Forti, S., Paganelli, F., Dazzi, P., & Brogi, A. (2023, June). Declarative Provisioning of Virtual Network Function Chains in Intent-based Networks. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)* (pp. 522-527). IEEE.

Rasa. (s. f.). Componentes. Recuperado el 22 de noviembre de 2023, de https://rasa.com/docs/rasa/components/#whitespacetokenizer.

Sharma, R. K., & Joshi, M. (2020). An analytical study and review of open source chatbot framework, rasa. *Int. J. Eng. Res*, *9*(06), 1011-1014.

Vlasov, V., Mosig, J. E., & Nichol, A. (2019). Dialogue transformers. *arXiv preprint arXiv:1910.00486*.