



Algoritmo para la traducción de intenciones en el modelo de Gestión de Redes Basada en Intenciones

Luis Miguel Ledo García, Ismanuel Basulto Díaz, Caridad Anías Calderón

RESUMEN

Las Redes Basadas en Intenciones se encuentra entre las propuestas más atractivas dentro del paradigma de gestión autónoma, pues prometen aliviar la complejidad de gestión de las infraestructuras y los servicios de redes de telecomunicaciones al usar lenguaje natural en lugar de comandos de configuración de equipos.

La empresa de Telecomunicaciones de Cuba, junto a la Universidad Tecnológica de La Habana (Cujae), está desarrollando el proyecto “Modelo de Gestión de Redes Basada en Intenciones” y dentro de su Bloque de Traducción, queda por establecer la forma en que deben ser procesadas las intenciones de los usuarios con el fin de convertirlas en un lenguaje entendible por la red. Teniendo en cuenta esta necesidad, en este artículo se presenta un algoritmo para la traducción de intenciones en el Bloque de Traducción del Modelo de Gestión de Redes Basada en Intenciones.

La solución desarrollada logra convertir expresiones en lenguaje natural a una estructura predefinida en formato json, que capta los valores más frecuentes de las intenciones de los usuarios. El algoritmo propuesto, implementado con soluciones de código abierto, posibilita la interacción con el usuario para clarificar la intención en caso de ambigüedades. Además, se considera el uso de Reconocimiento de Entidades Nombradas (NER) como una alternativa efectiva para el procesamiento de lenguaje natural en entornos de bajos recursos computacionales.

Palabras claves: Gestión de Redes Basadas en Intenciones, Procesamiento de Lenguaje Natural, intenciones, json, NER

ABSTRACT

Intent-Based Networking is among the most attractive proposals within the autonomous management paradigm, as it promises to alleviate the management complexity of telecommunications network infrastructures and services by using natural language instead of device configuration commands.

The Telecommunications Company of Cuba (ETECSA), together with the Technological University of Havana (CUJAE), is developing the project "Intent-Based Network Management model." In its Translation Block, the way in which user intents should be processed to convert them into a language understandable by the network remains to be established. Considering this need, this article presents an algorithm for the translation of the intents for the Translation Block in the Intent-Based Network Management model.

The developed solution successfully converts natural language expressions into a predefined json-formatted structure that captures the most frequent values of user intents. The proposed algorithm, which is intended to be implemented using code-based solutions, enables interaction with the user to clarify the intent in cases of ambiguity. Additionally, the use of Named Entity Recognition (NER) is considered as an effective alternative for natural language processing in low-resource computational environments.

Keywords: *Intent-Based Networking Management, Natural Language Processing, intents, json, NER*

Title: *Algorithm for the translation of intents for Intent-Based Network Management model*

Recibido: 06/2026 Aceptado: 09/2026

1. -INTRODUCCIÓN

La creciente complejidad de las infraestructuras de redes y la necesidad de agilidad operacional han impulsado el surgimiento de las Redes Basadas en Intenciones (IBN, del término en inglés *Intent Based Networking*). Este paradigma busca abstraer los detalles técnicos de configuración, permitiendo a los operadores definir el estado deseado de la red ("el qué") mediante lenguaje natural o interfaces de alto nivel, mientras un sistema automatizado se encarga de implementarlo ("el cómo"). En función de ello, se utilizan técnicas de Inteligencia Artificial (AI, del término en inglés *Artificial Intelligence*) y aprendizaje automático que permiten traducir intenciones, monitorizar el estado de la red en tiempo real y realizar ajustes proactivos, reduciendo así errores operativos y la intervención manual [1].

La "intención" en IBN se refiere a una descripción general de los resultados o propiedades que se desean alcanzar en la red, sin que sea necesario especificar cómo se implementan esos resultados a nivel de dispositivos o configuraciones técnicas. De esta manera, se establece un puente de comunicación entre el operador y la red en el lenguaje del operador, situación novedosa que facilita la gestión de la infraestructura [2].

Aunque el concepto de IBN ha generado un gran interés, su implementación práctica y completa sigue siendo un desafío, ya que aún no se ha logrado estandarizar un algoritmo concreto para la implementación de este enfoque en una red real. En este contexto, la Empresa de Telecomunicaciones de Cuba (ETECSA), junto a la Universidad Tecnológica de La Habana (Cujae), está desarrollando un modelo de Gestión de Redes Basadas en Intenciones (IBNM, del término en inglés *Intent Based Networking Management*). Un componente fundamental de su arquitectura es el Bloque de Traducción, responsable de convertir la intención expresada en lenguaje natural por un operador en una representación formal y estructurada que contenga toda la información necesaria para ejecutarla correctamente. Sin embargo, no existe una propuesta estandarizada sobre cómo lograr traducir las intenciones en la arquitectura del modelo IBNM, especialmente, para entornos de bajos recursos computacionales.

Para dar respuesta a lo anterior, en este artículo se presenta la propuesta de un algoritmo para la traducción de intenciones en el Bloque de Traducción del Modelo de Gestión de Redes Basada en Intenciones que hereda del proyecto la premisa de que se pueda emplear en entornos de bajos recursos computacionales.

Las contribuciones científicas fundamentales que se hacen en este trabajo son:

- Una propuesta de plantilla estructurada para la representación de intenciones en entornos IBN, basada en un análisis semántico y sintáctico de las necesidades de gestión de las redes de telecomunicaciones, que permite capturar los elementos esenciales de una intención de manera formal y computable.
- Un algoritmo para la traducción de intenciones en el Bloque de Traducción del modelo de IBNM, concebido específicamente para entornos con restricciones de recursos computacionales, que combina técnicas de Reconocimiento de Entidades Nombradas (NER, del término en inglés *Named Entity Recognition*) con lógica heurística y corrección ortográfica con el propósito de mejorar la robustez en la interpretación del lenguaje natural.

2.- Consideraciones para crear el algoritmo de traducción de intenciones que se propone.

Para crear el algoritmo de traducción de intenciones que se propone, se partió de la arquitectura del Modelo de Gestión de Redes Basada en Intenciones, la cual se muestra en la Figura 1.

Como se observa, la intención que ingresa el usuario en lenguaje natural es procesada por el Bloque de Traducción, el cual elabora un archivo en formato .json con la información extraída de la intención y la envía al Bloque de Planificación y Optimización. El Bloque de Traducción propuesto para dicha arquitectura se muestra en la Figura 2, donde se pueden apreciar las funcionalidades principales de este, al interactuar con el usuario y, usando Procesamiento del Lenguaje Natural (NLP, del término en inglés *Natural Language Processing*), exportar archivos json para el cumplimiento de las intenciones.



Fig 1:
Arquitectura para el Modelo de Gestión de Redes Basada en Intenciones [3].

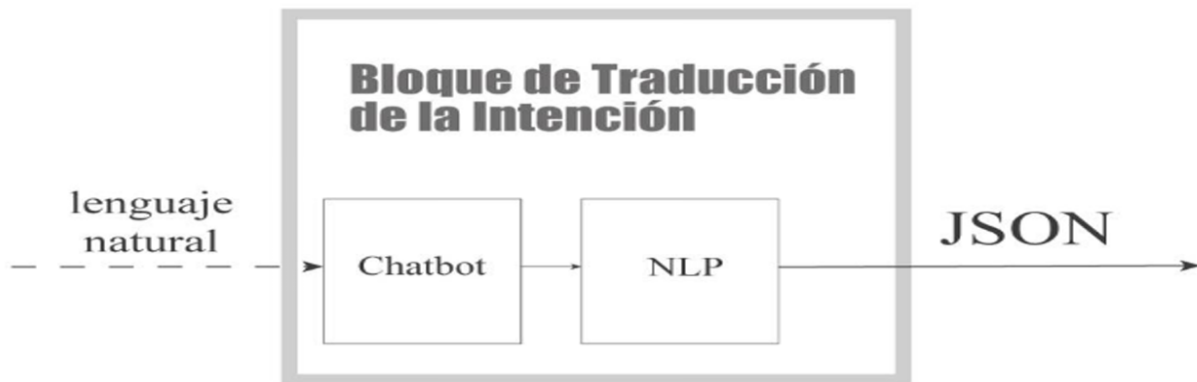


Figura 2:
Bloque de Traducción del Modelo de Gestión Basada en Intenciones [3].

2.1 Características previstas para el traductor de intenciones

En el ámbito de las IBN, procesar las intenciones de los usuarios requiere métodos que equilibren precisión, eficiencia y escalabilidad. Una opción es utilizar modelos *Transformers* que destacan por su precisión en interpretar matices y variaciones lingüísticas, lo que los hace ideales para entornos dinámicos, y para procesar un idioma con las características del español, que posee una rica flexión verbal y variedad léxica. Sin embargo, demandan grandes volúmenes de datos de entrenamiento y potencia computacional elevada para su implementación, lo que puede ser prohibitivo en infraestructuras con restricciones de recursos [4]. Una alternativa intermedia la constituyen los Modelos de Lenguaje Pequeños (SLM, del término en inglés *Small Language Models*), diseñados para operar con requisitos computacionales reducidos y un menor número de parámetros. Si

bien estos modelos resultan más eficientes y pueden ser desplegados en entornos con limitaciones de hardware, su adopción sigue enfrentando barreras significativas en contextos de infraestructura crítica como el que ocupa esta investigación [5].

Atendiendo a estas limitaciones, se propone utilizar un procesamiento basado en NER, mejorado con lógica adicional para ampliar su capacidad de análisis de intenciones. Un sistema de NER basado en reglas o modelos estadísticos tradicionales, a diferencia de uno basado en *Transformers* grandes o SLM, es menos demandante en recursos y permite una implementación más ágil, especialmente cuando las intenciones siguen patrones predefinidos. No obstante, su eficacia depende de la definición manual de entidades y reglas, lo que limita su adaptabilidad ante solicitudes atípicas o estructuras lingüísticas no contempladas [6,7].

Una vez definido el método a emplear, para traducir las intenciones se necesita definir la base de datos con la que se va a ejecutar el NER, y un modelo de AI para NLP donde estén contemplados, al menos, los principales vocablos del idioma español. De esta manera el NER puede establecer las relaciones entidad-valor que se desean (por ejemplo: actualizar-acción, BGP-protocolo, ancho de banda-parámetro y así sucesivamente) [8,9,10].

Dentro de las premisas del proyecto está que no se debe utilizar un modelo de lenguaje demasiado grande, porque comprometería los recursos para el resto de funciones críticas del modelo de IBNM. Por ello, se decide optar por spaCy, librería de código abierto, diseñada sobre Cython, ligera y con modelos de lenguaje de varios tamaños, lo que permite hacer pruebas de rendimiento durante la etapa de validación del algoritmo de traducción de intenciones [11]. Con el objetivo de seleccionar la variante más adecuada para el contexto del proyecto, se realizaron pruebas comparativas entre los tres modelos disponibles para español: *es_core_news_sm* (11 MB, vectores de 96 dimensiones), *es_core_news_md* (41 MB, vectores de 300 dimensiones) y *es_core_news_lg* (546 MB, vectores de 300 dimensiones). La evaluación consideró tres criterios: (1) uso de memoria RAM durante la carga del modelo y el procesamiento de una intención, (2) tiempo de ejecución promedio por intención y (3) precisión en la extracción de las seis entidades de la plantilla, medida sobre un subconjunto de 20 intenciones de prueba con estructuras sintácticas diversas. Los resultados se presentan en la tabla 1.

Tabla 1
Comparación entre modelos de lenguaje de spaCy

Modelo	Tamaño (MB)	RAM utilizada (MB)	Tiempo medio por intención (ms)	Precisión promedio en entidades (%)
sm	11	89	42	72,5%
md	41	134	58	78,3%
lg	546	612	89	84,1%

Aunque el modelo *lg* consume aproximadamente 612 MB de RAM, esta cifra representa solo el 3.8% de los 16 GB disponibles en el entorno de prueba, y el tiempo de procesamiento por intención (89 ms) sigue siendo perfectamente aceptable para un sistema interactivo donde el usuario espera una respuesta en tiempo real. Por el contrario, la ganancia en precisión del modelo *lg* respecto al *sm* (11.6 puntos porcentuales) resulta determinante para reducir la necesidad de completamiento interactivo, que es la operación que más impacta en la experiencia del usuario y en la eficiencia global del sistema. El modelo *lg* ofrece vectores de palabras con 300 dimensiones (frente a las 96 del *sm*), lo que mejora significativamente la captura de relaciones semánticas y la desambiguación contextual, especialmente relevante en un idioma como el español con alta flexión verbal y variación léxica. Adicionalmente, el modelo *lg* incluye un vocabulario más amplio y mejor cobertura de palabras compuestas y expresiones técnicas, lo que reduce los falsos negativos en entidades como “Tecnología” y “Mecanismo”. Por estos motivos, se seleccionó *es_core_news_lg* como modelo base, considerando que el incremento en el consumo de recursos es marginal en relación con la mejora sustancial en la calidad del procesamiento y la alineación con el objetivo de minimizar la intervención manual del usuario."

Finalmente, se reafirma el uso de *json* como salida del Bloque de Traducción ya que presenta una sintaxis simple y legible, los archivos son ligeros y se integra bien con varios lenguajes de programación, lo cual confiere libertad a los futuros desarrolladores del resto del proyecto [12,13].

3.- Diseño y validación del algoritmo propuesto para la traducción de la intención

A continuación, se expone la propuesta para la traducción de la intención. Primeramente, se explicará el diseño de la plantilla json encargada de almacenar el valor de las entidades de la intención, luego se expondrá el algoritmo propuesto para la traducción de la intención, usando NER. Finalmente, se presenta la validación del algoritmo y los resultados obtenidos.

3.1. Diseño y elaboración de la plantilla json para la salida del Bloque de Traducción

Después de tener claras las premisas para el Bloque de Traducción y analizar un total de 50 casos de usos de posibles intenciones que puede ingresar un administrador de red se diseñó una plantilla json que puede caracterizar dichas intenciones a través de 6 entidades que consideran posibles valores de entrada (no necesariamente todas las entidades deben tener valor, y puede suceder que dos o más entidades tengan el mismo valor). Estas entidades son:

Acción: Será la entidad que almacenará precisamente la operación que se hará mediante la intención. Siempre el valor de esta entidad será un verbo. “Acción” es imprescindible dentro del NER, ya que contempla las operaciones que pueden ejecutarse dentro de una red de telecomunicaciones. Esta es una entidad que siempre contendrá un valor. Ej: Habilitar, aislar, configurar, enviar, mostrar, optimizar, etc.

Objeto: En esta entidad se almacenará el objeto de la intención. Tomando elementos del análisis sintáctico del idioma español, el valor de “Objeto” debe ser el complemento directo de la oración, pero debe tenerse en cuenta que en algunos casos el valor semántico prima sobre la extracción puramente gramatical. Al igual que “Acción”, esta entidad nunca puede quedar vacía. Ej: en la intención “Actualizar el antivirus”, el valor de “Objeto” es “el antivirus”, porque es sobre quien se ejecuta la acción actualizar.

Tecnología: Esta es una entidad que ilustra cuáles son los componentes de la red que se ven involucrados en la intención a procesar, siempre será rellena con términos propios de una red de telecomunicaciones, como *WiFi*, *LAN*, *Router* etc. Es una entidad que no necesariamente tiene que contener un valor ya que es posible que algunas intenciones no inmiscuyan un componente de red para ejecutarse, por ejemplo: “Aumentar el ancho de banda”, o “Mostrar reportes de amenazas” no contemplan ningún valor para “Tecnología”.

Mecanismo: Esta entidad reflejará el método mediante el cual se ejecutará la acción sobre el objeto de la intención. Recoge términos técnicos de las redes de telecomunicaciones como protocolos, acciones específicas, procedimientos, etc. Ejemplos ilustrativos podrían ser “BGP”, “Balanceo de carga”, “QoS”, etc. Esta entidad confiere robustez al procesamiento y ejecución de las intenciones ya que provee especificidad en cuanto a la manera de ejecutar un proceso dentro de la red. No necesariamente debe contener un valor.

Condición: En ocasiones las intenciones pueden ejecutarse con determinada condición de por medio, estas pueden ser cualitativas o cuantitativas. Por ejemplo: “cancela el tráfico en la VLAN 7 “si” ya son las 10:00 PM o más tarde” (condición cualitativa) o “Elimina el caché “cuando” esta es superior a los 10MB” (condición cuantitativa). “Condición” recoge estas restricciones permitiendo mejorar la caracterización de la intención, haciendo que esta sea más específica. No necesariamente tiene que almacenar un valor.

Parámetros: Es la única entidad que se considera un arreglo de valores, ya que puede haber varios parámetros inmersos en una intención. Se considera “parámetro” a toda aquella propiedad dentro de la red que debe ser observada para garantizar la intención, por ejemplo: “ancho de banda”, “latencia”, “velocidad” etc. Pero también puede ser un dispositivo específico de la red y una propiedad del mismo Ej: “VLAN 10”, “IP 192.168.3.5”, “Tablas de ruta” etc. Esta entidad tiene como propósito brindar la mayor especificidad posible caracterizando de una mejor manera la intención del operador de la red. No necesariamente puede contener un valor, otras veces, algunos de sus valores pueden coincidir con el de otra entidad Ej: “Aumentar ancho de banda” tiene de objeto “Ancho de banda” pero este también es un parámetro de la intención.

De acuerdo con lo anterior, la plantilla json de algunas intenciones de ejemplos sería:

Intención: Añadir una ruta estática hacia 172.16.0.0/16 vía 10.0.0.1

Plantilla json: {

"action": "Añadir",

"object": "Ruta estática",

"technology": "No especificada",

"mechanism": "No especificado",

"condition": "Ninguna",

"parameters": ["IP 172.16.0.0", "vía 10.0.0.1"]

}

Intención: Programar una actualización OTA (del inglés *Over The Air*) para los dispositivos IoT cuando sean las 2 am.

Plantilla json: {

"action": "Programar",

"object": "Actualización",

"technology": "dispositivos IoT",

"mechanism": "OTA",

"condition": "Cuando sean las 2 am",

"parameters": []

}

De esta forma, queda precisada la manera en la que se introducen las intenciones, teniendo en cuenta posibles características semánticas, sintácticas y tecnológicas que puede tener una intención para gestionar una red: ¿qué hacer?, ¿sobre qué hacerlo?, ¿de qué manera?, ¿bajo qué condiciones?. Atendiendo a sus indicadores y empleando determinada tecnología se logran responder estas preguntas en la plantilla del json, de modo que se capture la orden que se desea ejecutar sobre la red. La combinación de estas 6 entidades se considera por los autores un método seguro y completo para procesar intenciones que gestionen una IBN en lenguaje natural.

3.2. Diseño del algoritmo para la traducción de la intención

Una vez definido el resultado final de la traducción, (el archivo json con las entidades anteriormente desglosadas) se debe considerar un algoritmo que convierta las entradas iniciales de los operadores en la estructura json especificada. A este algoritmo, también se le ha nombrado “el ciclo de vida del procesamiento de la intención”, y se muestra en la figura 3.

A continuación, se explica cada etapa que ocurre al procesar las intenciones:

Entrada del usuario (intención): Es una interfaz que permitirá al operador de la red introducir la intención que desea sea procesada, posibilitando que la ingrese manualmente.

Corrección del texto: En caso de que la intención se ingrese manualmente puede causar errores de escritura por parte del usuario que entorpecerán el correcto procesamiento de la intención. Es por eso que, antes de empezar a analizar semánticamente la intención del usuario, se ejecutan algunos algoritmos de preprocesamiento, como, por ejemplo, un corrector de errores tipográficos u ortográficos, permitiendo que algunos vocablos mal escritos se ajusten para ser considerados palabras reales del idioma español.

Tokenización: Permite identificar correctamente el valor de las entidades. Tokenizar es una opción que permite comparar los datos ingresados por el usuario con los del modelo de AI que se usan, por lo cual es un paso fundamental antes de comenzar el NER.

Reconocimiento de entidades nombradas (NER): Una vez tokenizado el texto, se le define al procesador la plantilla json que se desea mostrar como salida con sus respectivas entidades nombradas. Luego, el algoritmo comienza a extraer del texto los valores que deben guardarse en cada una de estas entidades.

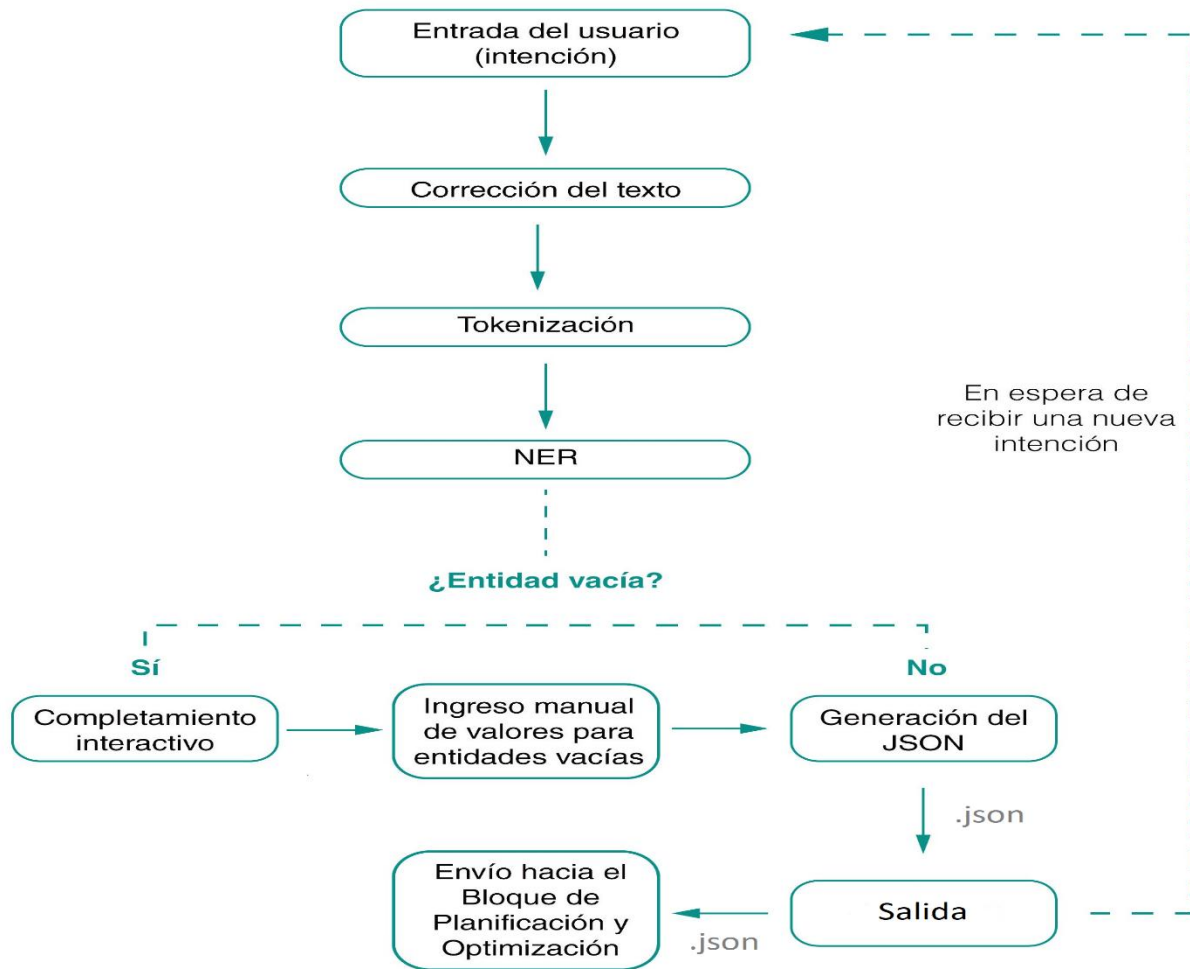


Figura 3:
Algoritmo para la traducción de la intención

Completamiento interactivo: Podría suceder que la intención del usuario no contenga la carga semántica necesaria para rellenar las entidades imprescindibles de la plantilla json. En tal caso, una función de completamiento interactivo notifica al usuario cuáles entidades, luego de ser procesada la intención por el NER, quedaron sin valor, y se le permitirá al usuario el **Ingreso manual de valores para entidades vacías**. De esta manera se asegura que las entidades más importantes se obtengan llenas. Cuando se confirme que estas entidades imprescindibles no están vacías, se avanza a la etapa de Generación del json directamente.

Generación del json: Finalizada la tarea de NER y el completamiento interactivo se valida su estructura según la plantilla json previamente definida, y se convierte el objeto en tipo .json.

Salida: Se muestra en consola la plantilla json con sus entidades cubiertas y se genera el archivo .json que será enviado al Bloque de Planificación y Orquestación. Además, se permite al usuario introducir inmediatamente la siguiente intención.

Dada la extensión del pseudocódigo presentado, se han puesto a disposición recursos complementarios para facilitar su implementación y reproducción. Estos incluyen diagramas de flujo de ejecución, el pseudocódigo completo en formato .txt, y el código fuente funcional en Python, disponible en un repositorio público de GitHub (<https://github.com/ismanbasdia/IBNM/tree/main/bloques/traduccion>). Se invita a la comunidad investigadora y desarrolladora a utilizar, modificar y contribuir a este código, fomentando la colaboración en el procesamiento de lenguaje natural aplicado a redes.

3.3. Validación del algoritmo para la traducción de la intención

Para la validación del algoritmo propuesto se empleó un entorno de desarrollo caracterizado por el uso de Visual Studio Code como IDE, ejecutado sobre un sistema con recursos computacionales limitados: un procesador Intel Celeron N5095A de 2Ghz de frecuencia de trabajo con 16GB de RAM, en correspondencia con las premisas del proyecto. Este entorno permitió implementar y probar el ciclo de vida del procesamiento de la intención mediante un script desarrollado en Python, estructurado de manera modular para garantizar su mantenibilidad y escalabilidad.

La lógica de programación implementada en el script sigue un flujo secuencial que garantiza el correcto procesamiento de cada intención desde su entrada hasta la generación del archivo .json de salida. El script se organiza en cuatro módulos fundamentales, cada uno con funciones específicas que se ejecutan en el orden establecido al correr el programa.

El primer módulo, denominado "Preparación del Entorno de Ejecución", se encarga de importar las librerías necesarias y cargar el modelo de lenguaje. Se utilizó la librería spaCy por su ligereza y su modelo para español "**es_core_news_lg**", cuyo núcleo está constituido por noticias, esto es lo que implica que su vocabulario no sea técnico (por eso core_news). De este tipo existen 3 variantes, se realizaron pruebas con todas y los recursos de cómputo no se vieron comprometidos por ninguno, así que finalmente se seleccionó la más grande (lg de large), de un tamaño de 546 MB con vectores de palabras y 300 dimensiones [14] junto con json para la generación de archivos de salida. Las librerías fuzzywuzzy y process facilitan la corrección de texto, con un umbral de 90% sobre la coincidencia más alta. Además, se usa la librería re para el manejo de expresiones regulares (regex). Adicionalmente, se incorporaron diccionarios con términos técnicos del dominio de telecomunicaciones y regex expansibles que permiten identificar patrones frecuentes en las intenciones de gestión de red, como direcciones IP, MAC o referencias a protocolos específicos. Todo esto se carga en memoria durante la ejecución del script, quedando disponible para los módulos subsiguientes.

El segundo módulo comprende la definición de funciones especializadas para la corrección de texto y la extracción de entidades. Para la corrección textual se desarrollaron funciones como `format_acronyms`, que normaliza siglas técnicas escritas incorrectamente (por ejemplo, convirtiendo "qoS" en "QoS"), y `correct_spelling`, que subsana errores ortográficos comunes mediante técnicas de coincidencia aproximada proporcionadas por fuzzywuzzy. Para la extracción de entidades se definieron funciones específicas para cada componente de la plantilla json: `identify_action`, `identify_object`, `identify_technology`, `identify_mechanism`, `identify_condition` e `identify_parameters`. Cada una de estas funciones implementa lógica de búsqueda basada en el análisis gramatical que realiza spaCy sobre el texto tokenizado, complementada con las expresiones regulares y los diccionarios técnicos cargados previamente.

El tercer módulo contiene la función `prompt_to_json`, que constituye el núcleo del procesamiento de lenguaje natural del script. Esta función recibe como parámetro la intención ingresada por el usuario, aplica secuencialmente las funciones de corrección y extracción explicadas en el segundo módulo, y construye un diccionario con los valores obtenidos para cada entidad. La implementación de esta función garantiza que el procesamiento se realice en el orden correcto: primero la corrección del texto, luego la tokenización mediante el modelo de spaCy y finalmente la extracción de cada entidad invocando a las funciones específicas. En caso de que las entidades consideradas imprescindibles (Acción y Objeto) queden vacías tras el procesamiento automático, la función activa un mecanismo de completamiento interactivo que detiene temporalmente la ejecución del script para solicitar al usuario que ingrese manualmente los valores faltantes, garantizando así que la intención quede correctamente especificada antes de proseguir.

El cuarto módulo corresponde a la función principal `main`, que actúa como interfaz entre el usuario y el procesador de intenciones. Esta función se ejecuta en un bucle continuo que permite al usuario introducir múltiples intenciones de forma consecutiva, simulando un entorno de operación real. Por cada iteración, `main` recibe la intención en lenguaje natural mediante entrada de teclado, invoca a `prompt_to_json` para su procesamiento, muestra el resultado en consola con formato legible y genera automáticamente un archivo .json que será enviado al Bloque de Planificación y Orquestación. El script finaliza su ejecución únicamente cuando el usuario ingresa el comando de salida predefinido.

Se realizaron pruebas sistemáticas ejecutando el script con un conjunto de 50 intenciones que abarcaban variaciones sintácticas, diferentes niveles de complejidad semántica y las cinco áreas funcionales de la gestión de redes: configuración, fallos, desempeño, contabilidad y seguridad. Se verificó especialmente el comportamiento del script ante entradas con errores ortográficos, uso de siglas técnicas y estructuras gramaticales no canónicas, así como la efectividad del mecanismo de completamiento interactivo cuando la información proporcionada resultaba insuficiente. Cada ejecución generó el archivo json correspondiente, cuyo contenido fue inspeccionado para confirmar la correcta extracción de las entidades y la adecuada estructuración de la salida.

En relación a la evaluación cuantitativa del desempeño del algoritmo, para cada una de las seis entidades de la plantilla json, se calcularon las métricas estándar de evaluación en sistemas de Reconocimiento de Entidades Nombradas: Precisión (*Precision*), Exhaustividad (*Recall*) y la media entre estas dos métricas (F1-Score), utilizando las definiciones estándar de Verdaderos Positivos (VP), Falsos Positivos (FP) y Falsos Negativos (FN). Un **Verdadero Positivo** se contabilizó cuando el algoritmo asignó correctamente la entidad y su valor coincidía con el valor de referencia definido por un evaluador experto para esa intención. Un **Falso Positivo** se registró cuando el algoritmo asignó un valor a una entidad que no correspondía según la referencia, o cuando asignó un valor incorrecto. Un **Falso Negativo** ocurrió cuando el algoritmo no asignó valor a una entidad que sí debía tenerlo según la referencia (dejándola vacía). Para las entidades que pueden quedar legítimamente vacías (Tecnología, Mecanismo, Condición y Parámetros), los casos en que el valor de referencia era "No especificada" o "Ninguna" y el algoritmo también lo dejó vacío se contabilizaron como Verdaderos Negativos, que no afectan los cálculos de Precisión, Recall y F1-Score ya que estas métricas se centran en la capacidad de detectar valores presentes.

La Tabla 2 presenta los resultados obtenidos durante la ejecución de las pruebas a una muestra de 50 intenciones, incluyendo las métricas de desempeño para cada entidad y el comportamiento general del sistema.

Tabla 2
Resultados cuantitativos de desempeño del algoritmo

Entidad	VP	FP	FN	Precisión (%)	Recall (%)	F1-Score (%)
Acción	46	1	3	97.9	93.9	95.9
Objeto	43	2	5	95.6	89.6	92.5
Tecnología	37	9	4	80.4	90.2	85.1
Mecanismo	35	8	7	81.4	83.3	82.3
Condición	42	5	3	89.3	93.3	91.2
Parámetros	41	3	6	93.2	87.2	90.1

Estos resultados evidencian que el algoritmo logra procesar de forma autónoma la mayoría de las intenciones (68%) sin necesidad de intervención del usuario. La corrección ortográfica resultó necesaria en el 38% de los casos, lo que confirma la utilidad de este módulo para mitigar errores de entrada. Las entidades "Acción" y "Objeto", definidas como imprescindibles, presentaron los mayores porcentajes de acierto (97.9% y 95.6% respectivamente), lo cual es consistente con el diseño del completamiento interactivo que garantiza que estas nunca queden vacías en la salida final. Las entidades "Mecanismo" y "Tecnología" mostraron los porcentajes más bajos (81.4% y 80.4% respectivamente), comportamiento esperado debido a que estas entidades dependen en mayor medida del vocabulario técnico especializado, el cual está parcialmente cubierto por los diccionarios incorporados.

4.- Análisis de resultados

Durante el proceso de análisis de resultados se verificaron principalmente dos funcionalidades del programa. Primero, la capacidad de procesar correctamente variaciones sintácticas que no afecten el significado de la intención, y luego, que fuera capaz de procesar intenciones de las llamadas cinco áreas funcionales de la gestión: configuración, fallos, desempeño, contabilidad y seguridad. A continuación, algunas intenciones procesadas correctamente:

Intención: Configurar una alerta si el uso de CPU del router supera el 90%

```
Resultado:{ "intent":  
{ "action": "Configurar",  
"object": "Alerta",  
"technology": "router",  
"mechanism": "No especificado",  
"condition": "Si el uso de cpu del router supera el 90%",  
"parameters": [ "Router" ]  
} }  
}
```

Intención: Notificar fallos en tiempo real

```
Resultado:{ "intent":  
{ "action": "Notificar",  
"object": "Fallos",  
"technology": "No especificada",  
"mechanism": "No especificado",  
"condition": "En tiempo real",  
"parameters": []  
} }  
}
```

Intención: Supervisar latencia, jitter, pérdida de paquetes y uso de ancho de banda en enlaces WAN y core.

```
Resultado: { "intent":  
{ "action": "Supervisar",  
"object": "Latencia jitter, pérdida de paquetes y uso de ancho de banda",  
"technology": "WAN y core",  
"mechanism": "No especificado",  
"condition": "Ninguna",  
"parameters": [ "Latencia", "Jitter", "Ancho de banda", "Enlaces WAN", "core" ]  
} }  
}
```

A continuación, un ejemplo de la misma intención escrita de dos formas con variaciones sintácticas, y diferentes resultados en consecuencia:

Intención: Registrar el router con IP 192.168.1.1 en el inventario

```
Resultado:{  
"intent": {  
"action": "Registrar",  
"object": "Router",  
"technology": "Router",  
"mechanism": "No especificado",  
"condition": "Ninguna",  
"parameters": [ "Router", "Inventario", "IP 192.168.1.1" ]  
} }  
}
```

Intención: Haz el registro del router con IP 192.168.1.1 en el inventario

```
Resultado:{  
"intent": {  
"action": "Haz",  
"object": "Registro",  
"technology": "Router",  
"mechanism": "No especificado",  
"condition": "Ninguna",  
"parameters": [ "Router", "Inventario", "IP 192.168.1.1" ]  
} }  
}
```

Por último, los resultados de procesar una intención sin completamiento interactivo y luego la misma incluyendo el completamiento interactivo

Intención: Ajustar automáticamente la capacidad de enlaces y servidores en función de la carga prevista o requisitos de servicio

Resultado sin completamiento interactivo:

```
{ "intent":  
{ "action": "Ajustar",  
"object": "Capacidad",  
"technology": "Enlaces y servidores",  
"mechanism": " ",  
"condition": " ",  
"parameters": [ ] } }
```

Resultado con completamiento interactivo:

```
{ "intent":  
{ "action": "Ajustar",  
"object": "Capacidad",  
"technology": "Enlaces y servidores",  
"mechanism": "Automáticamente",  
"condition": "En función de la carga prevista o requisitos del servicio",  
"parameters": [ "enlaces", "servidores", ] } }
```

Los resultados cuantitativos presentados en la Tabla 1 complementan los ejemplos cualitativos expuestos anteriormente. El 68% de procesamiento autónomo y el 32% de intervención requerida indican un equilibrio aceptable entre automatización y control del usuario, especialmente en un entorno donde la diversidad lingüística del español puede generar ambigüedades. El hecho de que el 38% de las intenciones requiriera corrección ortográfica resalta la pertinencia de haber incluido este paso en el preprocesamiento, ya que, sin él, estas intenciones probablemente habrían sido mal interpretadas o no procesadas correctamente.

El análisis detallado de los casos en los que el algoritmo incurrió en falsos positivos o falsos negativos permite identificar patrones de error recurrentes que delimitan sus limitaciones actuales y sugieren vías de mejora.

En la entidad “Tecnología” (80.4% de acierto), los errores más frecuentes estuvieron asociados a la ambigüedad contextual. Se observaron falsos negativos cuando la intención mencionaba un componente de red de forma implícita o mediante términos compuestos no incluidos en los diccionarios. Por ejemplo, en la intención "Aplicar políticas de seguridad perimetral", el algoritmo no logró identificar "seguridad perimetral" como una tecnología, asignando "No especificada" a esta entidad. Por otro lado, se presentaron “falsos positivos” cuando el algoritmo confundió sustantivos genéricos (como "sistema" o "dispositivo") con tecnologías específicas, debido a que el modelo “spaCy” los etiqueta como entidades nominales y el módulo heurístico los asigna erróneamente si aparecen en posiciones sintácticas similares a las de los componentes de red.

La entidad “Mecanismo” (81.4% de acierto) fue la que presentó el mayor número de errores de clasificación. Los “falsos negativos” predominaron cuando el mecanismo se expresaba mediante adverbios de modo (ej. "automáticamente", "progresivamente", "manual") o mediante frases preposicionales no estandarizadas (ej. "por medio de OSPF"), ya que estos patrones lingüísticos no estaban suficientemente cubiertos por las expresiones regulares ni por los diccionarios, que priorizan sustantivos técnicos (ej. "BGP", "QoS"). Los “falsos positivos” ocurrieron con menor frecuencia, pero se detectaron casos en los que el algoritmo asignó a "Mecanismo" términos que semánticamente correspondían a "Condición" (ej. "en caso de sobrecarga") o a "Parámetros" (ej. "protocolo TCP"), revelando un solapamiento semántico que las reglas heurísticas actuales no logran desambiguar correctamente.

Adicionalmente, en la entidad “Acción” (92% de acierto), aunque el porcentaje es alto, los errores detectados fueron sistemáticos y se corresponden con la limitación señalada en las conclusiones: el uso de verbos conjugados en imperativo (ej. "Haz", "Pon") o en otras formas no infinitivas, que el algoritmo no logra lematizar adecuadamente. Esto provoca que el valor almacenado sea la forma conjugada en lugar del infinitivo ("Haz" en lugar de "Hacer" o "Registrar"), lo que, si bien no impide el procesamiento posterior, introduce ruido en la estandarización de la salida.

La elección del modelo *lg*, respaldada por las pruebas comparativas de rendimiento, demuestra que es posible lograr altos niveles de precisión sin comprometer la viabilidad del sistema en entornos de recursos limitados, siempre que se realice un balance cuidadoso entre el consumo de memoria y la ganancia en exactitud.

Estos patrones de error indican que las principales debilidades del algoritmo residen en la cobertura limitada de los diccionarios técnicos para términos compuestos y variantes lingüísticas, así como en la rigidez de las reglas heurísticas para distinguir entidades semánticamente cercanas. Futuras líneas de trabajo deberían enfocarse en la incorporación de un lematizador más robusto, la expansión dinámica de los diccionarios y la implementación de mecanismos de desambiguación contextual basados en la co-ocurrencia de términos.

A partir de estas y algunas otras intenciones de validación se obtienen los siguientes resultados:

- El traductor es sensible a variaciones sintácticas sin que estas impliquen variaciones en su significado, lo cual es una dificultad a superar.
- El traductor procesa intenciones de las 5 áreas funcionales de la gestión (las intenciones mostradas pertenecen a las áreas de desempeño, configuración y fallos), lo cual es alentador teniendo en cuenta que el modelo de lenguaje con el que se trabaja tiene un vocabulario genérico que no presenta vocablos técnicos necesarios para la gestión de las redes. Esto es posible gracias a la ayuda de los diccionarios (con palabras técnicas muy utilizadas en la gestión de las redes) y las *regex* (con varias sintaxis empleadas en la gestión de las redes como los puntos que separan las direcciones IP) incorporados en la preparación del entorno de ejecución.
- El algoritmo permite que el usuario ingrese de forma manual el valor de las entidades imprescindibles que quedan vacías, lo cual brinda la posibilidad de que en caso de que el programa no haga la traducción correctamente por sí solo, la intención sea traducida tal y como el administrador desea.

4.1. Análisis de la sensibilidad a variaciones sintácticas

Uno de los hallazgos más relevantes durante la validación fue la sensibilidad del algoritmo ante variaciones sintácticas que, sin modificar el significado semántico de la intención, producen cambios en los valores extraídos para las entidades. Esta

limitación se manifiesta de forma especialmente acusada en la entidad “Acción”, donde el algoritmo depende en gran medida de que el verbo principal aparezca en infinitivo y en posición temprana de la oración (típicamente al inicio). Cuando el usuario emplea verbos conjugados en imperativo (ej. “Haz el registro...”), en presente de indicativo (ej. “Registro el router...”) o formas perifrásticas (ej. “Vamos a configurar...”), el modelo spaCy etiqueta correctamente la raíz verbal, pero el módulo heurístico de extracción no siempre logra normalizar la forma a infinitivo, almacenando la forma flexionada (ej. “Haz” en lugar de “Hacer”). Si bien esto no impide el procesamiento posterior, introduce inconsistencia en la salida y puede afectar a módulos descendentes que esperen verbos en infinitivo.

Otra fuente de variación sintáctica está en el orden de los constituyentes. El español permite una gran libertad posicional: el objeto directo puede aparecer antes del verbo (ej. “El router con IP 192.168.1.1 regístralo en el inventario”), o el complemento circunstancial puede interponerse entre el verbo y el objeto. El algoritmo actual, basado en un recorrido secuencial de los tokens etiquetados por spaCy y en reglas heurísticas que asumen un orden canónico (Sujeto + Verbo + Objeto + Complementos), falla en estos casos, extrayendo valores incorrectos o dejando entidades vacías que luego deben ser completadas por el usuario. Por ejemplo, en la intención “Del router, verifica la tabla de rutas”, el algoritmo extrajo “Verifica” como acción, pero asignó “Router” a la entidad Objeto en lugar de “Tabla de rutas”, debido a que el sintagma preposicional inicial “Del router” fue interpretado como el objeto principal.

Para abordar esta limitación en futuras iteraciones del sistema, se proponen las siguientes líneas de trabajo concretas, todas ellas orientadas a mantener la ligereza computacional que exige el proyecto:

1. Análisis de dependencias gramaticales (*Dependency Parsing*): En lugar de confiar únicamente en el etiquetado POS (*Part-of-Speech*) secuencial, se propone explotar las relaciones de dependencia que spaCy ya calcula (mediante el atributo “dep_” y “head”). Estas relaciones identifican de forma inequívoca el núcleo del predicado (raíz verbal), el sujeto (nsubj), el objeto directo (dobj) y los complementos circunstanciales (advcl, prep). Implementar una lógica de extracción basada en el árbol de dependencias permitiría localizar la entidad “Acción” como la raíz del predicado independientemente de su posición en la oración, y “Objeto” como el dependiente con relación “dobj”. Esta técnica es computacionalmente ligera porque spaCy ya genera estas estructuras durante el procesamiento, sin necesidad de modelos adicionales.
2. Lematización avanzada con normalización de formas verbales: Actualmente, el algoritmo utiliza la forma original del token para la entidad “Acción”. Se propone incorporar una capa de lematización que convierta cualquier forma verbal (conjugada, imperativa, perifrástica) a su infinitivo canónico. spaCy proporciona el lema de cada token (atributo lemma_), que para verbos devuelve el infinitivo. Mejorar el módulo de extracción para que, al identificar el token raíz del predicado, almacene su lema en lugar de su forma superficial. Esto resolvería los casos de “Haz” → “hacer”, “Registro” → “registrar”, etc., sin coste computacional adicional.
3. Reconocimiento de patrones sintácticos flexibles mediante expresiones regulares extendidas: Si bien ya se usan regex para identificar direcciones IP, MAC, etc., se podría ampliar su uso para capturar estructuras sintácticas no canónicas, como sintagmas preposicionales desplazados al inicio (“Del router,...”), mediante patrones que identifiquen la presencia de preposiciones y sustantivos antes del verbo, y los reubiquen en la entidad correspondiente (en este caso, Tecnología en lugar de Objeto). Esto requiere un diseño cuidadoso de las regex para no introducir falsos positivos, pero es una solución ligera.
4. Validación mediante sinónimos y paráfrasis: Enriquecer los diccionarios con sinónimos de verbos de acción (“registrar”, “añadir”, “incorporar”) y de objetos técnicos, de modo que variaciones léxicas no afecten la extracción. Además, se podría implementar un pequeño módulo de paráfrasis que reescriba oraciones no canónicas a una forma canónica antes de la extracción, utilizando reglas de transformación sintáctica basadas en patrones (por ejemplo, mover los complementos circunstanciales iniciales al final). Esta técnica, aunque más compleja, es factible con un conjunto limitado de reglas para el dominio de gestión de redes.

La implementación de estas mejoras no requeriría modelos de lenguaje adicionales ni un incremento significativo en el consumo de recursos, ya que se apoyan en las capacidades ya presentes en spaCy (dependencias, lematización) y en expansiones controladas de los diccionarios y reglas heurísticas. Con ello, se espera reducir la tasa de completamiento interactivo y aumentar la robustez del sistema ante la diversidad lingüística del español, acercándolo al objetivo de una auténtica gestión por intención.

5.- CONCLUSIONES

El desarrollo del traductor de intenciones para el Modelo de Gestión de Redes Basada en Intenciones demuestra la viabilidad de emplear lenguaje natural en entornos con restricciones computacionales. La implementación, basada en Reconocimiento de Entidades Nombradas, corrección ortográfica y reglas heurísticas, logra procesar intenciones en español sin infraestructuras de alto rendimiento, alcanzando un 68% de procesamiento autónomo y aciertos superiores al 80% en las entidades principales (Acción, Objeto, Tecnología y Parámetros). Estos resultados confirman que el enfoque propuesto constituye una alternativa viable a los modelos de lenguaje de gran tamaño (LLM) y pequeños (SLM) para tareas específicas de traducción de intenciones en contextos de recursos limitados.

La plantilla json estructurada en seis entidades (Acción, Objeto, Tecnología, Mecanismo, Condición y Parámetros) captura los elementos esenciales de una intención de gestión con suficiente granularidad para su procesamiento posterior, mientras que el uso exclusivo de software de código abierto (*spaCy*, Python, Visual Studio Code) garantiza reproducibilidad y elimina barreras de adopción, demostrando que es posible construir herramientas funcionales para IBN sin depender de soluciones propietarias.

Las principales limitaciones identificadas son la dependencia de verbos en infinitivo para una extracción robusta de la entidad Acción, la sensibilidad ante ambigüedades semánticas entre Mecanismo y Condición, y la cobertura incompleta de los diccionarios para términos técnicos compuestos. Aunque el corrector ortográfico y el completamiento interactivo mitigan parcialmente estas restricciones, los patrones de error detectados (falsos negativos en Mecanismo y falsos positivos en Tecnología) evidencian áreas de mejora.

Los trabajos futuros se orientarán a: (1) explotar el análisis de dependencias gramaticales de *spaCy* para extraer entidades independientemente del orden sintáctico, (2) incorporar lematización avanzada que normalice formas verbales conjugadas a infinitivo, (3) enriquecer dinámicamente los diccionarios técnicos y (4) refinar las reglas heurísticas para desambiguar entidades semánticamente cercanas. El modelo *spaCy es_core_news_lg*, a pesar de su entrenamiento con corpus generales, proporcionó una base suficiente para la tokenización y el análisis gramatical, siendo suplidas las deficiencias del vocabulario técnico mediante los recursos léxicos y reglas incorporados.

REFERENCIAS

1. Hurtado R, Picón C, Muñoz A, Hurtado J. Survey of Intent-Based Networks and a Methodology Based on Machine Learning and Natural Language Processing. En: Lecture Notes in Networks and Systems. 2024;696:363-82.
2. Leivadeas A, Falkner M. A Survey on Intent-Based Networking. IEEE Commun Surv Tutor. 2023;25(1):625-55.
3. Basulto Díaz I, Anías Calderón C. Propuesta de arquitectura para una Red Basada en Intenciones. Ing Electrónica Automática Comun. 2024;45(2):59-69.
4. Xu G, Hao Z, Shen L, Luo Y, Sun F, Wang X, et al. CoFormer: Collaborating with Heterogeneous Edge Devices for Scalable Transformer Inference. arXiv:2508.20375 [Preprint]. 2025. Disponible en: <https://arxiv.org/abs/2508.20375>.
5. Lu Z, Li X, Cai D, Yi R, Liu F, Liu W, et al. Demystifying Small Language Models for Edge Deployment. En: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers); Vienna, Austria. Association for Computational Linguistics; 2025. p. 14747-64. doi:10.18653/v1/2025.acl-long.718.
6. Newman-Griffis D, Fain Lehman J, Rosé C, Hochheiser H. Translational NLP: A New Paradigm and General Principles for Natural Language Processing Research. En: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; 2021; Online. Association for Computational Linguistics; 2021. p. 4125-38. doi:10.18653/v1/2021.naacl-main.325.
7. Aristizabal M, et al. Intent-Based Networking with Lightweight NLP: A Case Study. En: 2023 IEEE Latin-American Conference on Communications (LATINCOM); 2023. p. 1-6.
8. Zhao WX, Zhou K, Li J, Tang T, Wang X, Hou Y, et al. A Survey of Large Language Models. arXiv:2303.18223 [Preprint]. 2023. Disponible en: <https://arxiv.org/abs/2303.18223>.

9. Rothman D. Transformers for Natural Language Processing: Build, train, and fine-tune deep neural network architectures for NLP with Python, Hugging Face, and OpenAI's GPT-3, ChatGPT, and GPT-4. 2nd ed. Birmingham: Packt Publishing Ltd.; 2022.
10. Naseer S, Mudasar Ghafoor M, Alvi S bin K, Kiran A, Ur Rahmand S, Murtaza G, et al. Named Entity Recognition (NER) in NLP Techniques, Tools Accuracy and Performance. Pak J Multidiscip Res. 2022;2(2):293-308.
11. Honnibal M, Montani I, Van Landeghem S, Boyd A. spaCy: Industrial-strength Natural Language Processing in Python. Zenodo; 2020. doi:10.5281/zenodo.1212303.
12. Gerrans J, Sherratt RS. Comparing XML and JSON Characteristics as Formats for Data Serialization Within Ultralow Power Embedded Systems. IEEE Embed Syst Lett. 2024;16(4):489-92. doi:10.1109/LES.2024.3450576.
13. Mora-Castillo JA. Serialización/deserialización de objetos y transmisión de datos con JSON: una revisión de la literatura. Tecnol Marcha. 2016;29(1):118-33. doi:10.18845/tm.v29i1.2544.
14. Explosion AI. Spanish · spaCy Models Documentation [Internet]. Disponible en: <https://spacy.io/models/es>

CONFLICTO DE INTERESES

Ninguno de los autores manifestó la existencia de posibles conflictos de intereses que debieran ser declarados en relación con este artículo.

CONTRIBUCIONES DE LOS AUTORES

Ing. Luis Miguel Ledo García: investigación, redacción, programación, redacción-borrador original

Ing. Ismanuel Basulto Díaz: administración del proyecto, conceptualización, metodología, investigación, visualización, revisión.

Dr. C Caridad Anías Calderón: validación, supervisión y redacción- revisión y edición.

AUTORES

Ing. Luis Miguel Ledo García, Ingeniero en Telecomunicaciones y Electrónica, Empresa de Telecomunicaciones de Cuba S.A(ETECSA), La Habana, Cuba, correo electrónico: lucho45tele@gmail.com, ORCID ID: <https://orcid.org/0009-3892-9138>. Especialista B en telemática dentro de la Supervisión Nacional de ETECSA. Sus principales intereses son el desarrollo de software y la investigación sobre nuevas formas de gestión de las Redes.

Ing. Ismanuel Basulto Díaz, Ingeniero en Telecomunicaciones y Electrónica de la Universidad Tecnológica de La Habana José Antonio Echeverría (Cujae), Especialista D en Telemática en la Unidad de Desarrollo e Innovación de la Empresa de Telecomunicaciones de Cuba, correo electrónico: ismanuelbasulto00@gmail.com, ORCID ID: <https://orcid.org/0009-0007-9820-6435>. Como aspirante a doctorado, sus principales intereses de investigación son Gestión Autónoma de Redes de Telecomunicaciones, Inteligencia artificial, gestión de infraestructuras de redes y la nube.

Caridad Anías Calderón, Ingeniera en Telecomunicaciones, Master en Telemática y Doctora en Ciencias Técnicas y Honoris Causa. Profesora titular y de mérito de la Universidad Tecnológica de La Habana, Cujae, Cuba, correo electrónico: catcha@tesla.cujae.edu.cu, ORCID ID: <https://orcid.org/0000-0002-5781-6938>. Sus principales intereses de investigación están centrados en las redes y servicios de telecomunicaciones y en especial en su gestión.



Esta revista se publica bajo una [Licencia Creative Commons Atribución-No Comercial-Sin Derivar 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/)