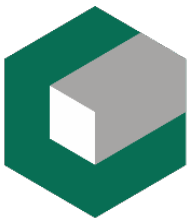


Universidad Tecnológica de la Habana

"José Antonio Echeverría"

CUJAE

Generación automática de requisitos de software a partir de opiniones de usuarios combinando aprendizaje automático y grandes modelos de lenguaje (LLM)

Trabajo de diploma presentado en opción al título de Ingeniero Informático

Autor:

Ray Maestre Peña (raymaestrex@gmail.com)

Tutores:

Dr. C. Alfredo Simón Cuevas (asimon@ceis.cujae.edu.cu)

Dr. C. Orlando Gabriel Toledano López (Universidad de Ciencias Informáticas)

La Habana, Cuba

Junio de 2025

Agradecimientos

He tenido la fortuna de que muchas personas me hayan acompañado a lo largo de esta travesía universitaria, sin las cuales no habría llegado tan lejos.

Primero, quisiera agradecer a mi madre, Neysi, mi principal apoyo. Gracias por tu incondicional cariño y por haber estado a mi lado desde el primer momento de mi vida. Gracias porque nunca tomaste tu delicado estado de salud como una razón para ayudarme menos e incluso en los peores momentos siempre me cuidaste por encima de todo. Siempre recordaré con cariño el sabor de tu comida luego de horas de estudio aliviando mi fatiga y tus constantes “¿Adelantaste?” mientras luchaba contra el tiempo por algún proyecto. Sinceramente, gracias.

Gracias a mi padre, Ramón, porque a pesar de no haber estado físicamente a mi lado siempre me acompañaste con el corazón. Ha sido gracias a tu constante apoyo que he podido centrarme en mis estudios completamente y por el cual puedo decir que a pesar de las muchas dificultades no me ha faltado nada. Y, sobre todo, gracias por respetar y apoyar mis sueños a lo largo de estos años, aunque no siempre hayan estado alineados con tus ideas.

A mis mejores amigos, Javier y David, por haber estado ahí siempre para dar consejo y ayudar en lo que sea. Aunque la vida nos haya llevado por distintos caminos siempre hemos encontrado el momento de encontrarnos y pasarla bien. Esas esporádicas reuniones han sido un oasis de risas dentro del mar de esfuerzos y dificultades que ha representado la universidad. Por todos estos años de alegrías, confrontaciones, y sincera amistad, gracias.

A mi tutor, Alfredo por tanto que he aprendido con él. Han sido muchas las locas ideas que han ido y venido durante la conformación de esta tesis y he crecido mucho explorándolas. Aunque en un primer momento no pensé que mis últimos años en la

universidad tendrían un enfoque tan investigativo ha sido una experiencia por la que realmente le agradezco, gracias por la oportunidad.

Gracias a todos mis compañeros en la universidad, aquellos que han luchado esta batalla lado a lado conmigo en busca de superarse y conseguir nuestras metas. Estrada, Montes, Amaury, Alejandra espero que puedan una vez graduados puedan cumplir sus sueños.

A todos los profesores de la facultad que ha contribuido al crecimiento mío y de mis compañeros, sus enseñanzas tanto en software como en la vida no serán olvidadas.

-

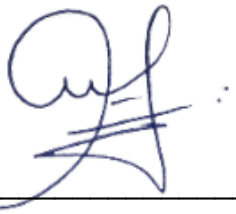
Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana “José Antonio Echeverría” (CUJAE) para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste, firmo por la presente en La Habana, Cuba, a los 22 días del mes de junio del 2025.

Ray Maestre Peña

Firma del autor



Dr. C. Alfredo Simón Cuevas

Firma del tutor

Informe de opinión del TUTOR del trabajo de diploma

Título: Generación automática de requisitos de software a partir de opiniones de usuarios combinando aprendizaje automático y grandes modelos de lenguaje (LLM).
Autores: Ray Maestre Peña

La ingeniería de requisitos basada en datos (IRBD) surge como un nuevo paradigma en la toma de decisiones sobre los requisitos, basado en el análisis, la interpretación y la predicción de datos que aporten conocimiento sobre estas necesidades. Las opiniones de los usuarios que se generan en las plataformas de venta de aplicaciones (Apple App Store, Google Play, y otras) o en las redes sociales, son una fuente de información muy valiosa, y está siendo clave en la implementación de este paradigma. A partir de ese contenido, se pueden identificar nuevos requisitos, errores, y valoraciones relevantes sobre la calidad del funcionamiento de las aplicaciones, así como elementos de decisión para la priorización de requisitos. Sin embargo, el tratamiento de esta fuente de información genera otros problemas, derivados del gran volumen de información, la diversidad temática, le informalidad en la que se redactan esos contenidos, lo que hace sumamente desafiante el poder obtener conocimientos relevantes de ese contenido.

El trabajo de diploma desarrollado se enfoca la concepción y desarrollo de un método de generación de requisitos de software a partir de grandes volúmenes de opiniones de usuarios. Esta solución innovadora y creativa, se concibió en tres fases: un filtrado de alta eficacia basado en el uso de modelos de aprendizaje con arquitectura Transformer, potenciado con la inyección de conocimiento del dominio para identificar opiniones relevantes, la aplicación de un algoritmo de agrupamiento para identificar temas afines, y la generación de requisitos utilizando grandes modelos de lenguaje (LLM) y un diseño de *prompting* muy bien concebido. Aunque esta es la primera aproximación de esta solución, marca un hito importante porque hasta ahora no se ha reportado en la literatura una solución con este alcance. El documento que se presenta recoge los fundamentos más importantes y necesarios para llegar al planteamiento, formalización y desarrollo de la solución propuesta, así como un adecuado y detallado proceso de evaluación que permite demostrar la validez y potencialidades de la solución que se propone, así como el cumplimiento de los objetivos planteados en la investigación de manera muy satisfactoria. El documento posee una adecuada estructuración, redacción y ortografía, y la bibliografía es muy actualizada y muy relevante para la temática abordada; en general su uso el cuerpo del documento es adecuado.

Los resultados que se exponen en esta tesis han sido fruto del esfuerzo, la dedicación, la responsabilidad, la seriedad y la autosuperación con la que Ray ha enfrentado este proyecto. La tarea asignada fue retadora porque implicaba el manejo de tecnologías que no se imparten en la carrera, como el uso de modelos Transformers, pero más retadora fue cuando, el objetivo y alcance del proyecto se sobredimensionó para, no solo clasificar la opiniones en “informativas” o “no informativas”, sino llegar a generar los requisitos de forma automática y con eficacia, para lo que se tuvo que también asimilar todo lo relacionado con la ingeniería de prompting. En el desarrollo de esta tesis y su presentación Ray no solo demuestra haber vencido todos los importantes retos que demandó el proyecto y con excelentes resultados sobresalientes, muy novedosos (no se reporta en la literatura una solución de esta naturaleza), sino que en este camino también mostró un crecimiento en su desarrollo profesional extraordinario. Ha sido de gran satisfacción haber podido trabajar con Ray en el

desarrollo del proyecto durante estos últimos años, donde los resultados positivos no solo están en el aprendizaje y lo profesional, sino también en los vínculos afectivos y de amistad que han emergido del trabajo conjunto, y que desearía perduren en el tiempo.

Por todo lo anterior expresado, considero que los estudiantes Ray Maestre Peña reúnen los méritos necesarios para otorgarle el título en Ingeniería Informática y propongo que se le otorgue al Trabajo de Diploma la máxima calificación (5 puntos – Excelente).

Se firma el presente con fecha: **23 de junio de 2025**

Dr. Alfredo Simón Cuevas
Tutor

Resumen

La Ingeniería de Requisitos contemporánea se enfrenta al desafío de procesar eficazmente los masivos volúmenes de opiniones de usuarios generados en plataformas digitales. Estos datos, aunque valiosos, presentan una baja densidad de información útil, lo que exige soluciones computacionales avanzadas. La presente investigación propone y valida un método para la generación automática de requisitos de software a partir de estas fuentes, con el objetivo de nutrir el ciclo de vida evolutivo de las aplicaciones. La metodología propuesta se desarrolla en tres fases: un filtrado de alta eficacia mediante un modelo Transformer, potenciado con la inyección de conocimiento del dominio para aislar opiniones relevantes; un agrupamiento temático de los comentarios filtrados para identificar temas afines; y una generación de requisitos en dos pasos utilizando Grandes Modelos de Lenguaje, que primero crea requisitos por clúster y luego los unifica en una lista final. La validación experimental demuestra que la solución no solo es viable, sino que el componente de clasificación supera a los enfoques del estado del arte. Los resultados evidencian que, con una preparación de datos y una especificación de *prompts* adecuadas, es posible generar requisitos de calidad que pueden guiar el desarrollo y mejora continua de productos de software.

Palabras clave: generación automática de requisitos, Ingeniería de requisitos basada en datos, minería de opinión, modelos Transformers, Grandes Modelos del Lenguaje.

Abstract

Contemporary Requirements Engineering faces the challenge of efficiently processing the massive volumes of user feedback generated on digital platforms. This data, while valuable, presents a low density of useful information, demanding advanced computational solutions. This research proposes and validates a method for the automatic generation of software requirements from these sources, with the aim of nurturing the evolutionary lifecycle of applications. The proposed methodology unfolds in three phases: a high-efficacy filtering stage using a Transformer model, enhanced by the injection of domain knowledge to isolate relevant feedback; a thematic clustering of the filtered comments to identify related topics; and a two-step requirements generation process using Large Language Models, which first creates requirements per cluster and then unifies them into a final list. The experimental validation demonstrates that the solution is not only viable, but its classification component also outperforms state-of-the-art approaches. The results show that with proper data preparation and prompt specification, it is possible to generate quality requirements that can effectively guide the continuous development and improvement of software products.

Keywords: Automatic requirements generation, Data-driven requirements engineering, Opinion mining, Transformer models, Large language models.

Índice

Índice de Figuras

Índice de Tablas

Introducción

El éxito de un sistema de software se puede medir por cuán bien cumple con su propósito, que se define a través de requisitos. Autores como *Ian Sommerville* [1] los definen como un enunciado abstracto de alto nivel de un servicio que debe proporcionar un sistema, o bien, una restricción sobre un sistema. Son, en esencia, el mapa que guía el desarrollo. Los requisitos mal definidos pueden llevar a un software que no satisface las demandas de los usuarios, resultando en un proyecto de software fallido [2]. Por tanto, la Ingeniería de Requisitos juega un papel crucial en el éxito o fracaso de un proyecto de software.

La Ingeniería de Requisitos (IR) tradicionalmente ha involucrado a los usuarios a través de entrevistas, talleres, grupos focales, y encuestas, para capturar información relevante sobre las aplicaciones de software [3]. Sin embargo, a medida que el mercado se hace más competitivo, las empresas de desarrollo de software (sobre todo las de desarrollo de aplicaciones para móviles), luchan por ganarse un lugar en las preferencias de los usuarios [4]. Por tanto, analizar la calidad de las aplicaciones, desde la perspectiva de las experiencias de los usuarios, identificar sus insatisfacciones y necesidades, se ha vuelto crucial para el éxito de este tipo de empresas [5].

Las opiniones de los usuarios que se generan en las plataformas de venta de aplicaciones (*Apple App Store*, *Google Play*, y otras) o en las redes sociales, son una fuente de información muy valiosa, que puede impulsar la evolución y mejora de los productos de software [6]. A partir de ese contenido, se pueden identificar nuevos requisitos, errores, y valoraciones relevantes sobre la calidad del funcionamiento de las aplicaciones, así como para la priorización de requisitos [7, 8]. Sin embargo, esto da lugar a nuevos problemas, la cantidad de opiniones es demasiado grande a medida que escala en cantidad de usuarios y se vuelve completamente imposible analizarlas todas a mano [9, 10]. Además, estudios como [11] han demostrado que entre el 50% y el 70% de dichas opiniones no son relevantes para un equipo de desarrollo. Esto da lugar a una situación en la que una fuente de información importante es desperdiciada, conduciendo a características no óptimas y oportunidades perdidas.

Este escenario ha propiciado el surgimiento de la Ingeniería de Requisitos Basada en Datos (IRBD) [3, 12]. Este tipo de enfoques hace referencia a una estrategia de toma de decisiones basada en el análisis, la interpretación y la predicción de datos. El gran volumen de contenido disponible de este tipo obstaculiza significativamente la adopción de este nuevo enfoque, si no se trata computacionalmente [13].

Con los recientes avances en el campo de la inteligencia artificial, especialmente en el campo del procesamiento del lenguaje natural se han abierto nuevas posibilidades para solucionar este problema. Muchos autores en los últimos años como [14-23] han enfrentado la clasificación de opiniones de usuarios mediante la utilización de algoritmos de aprendizaje automático para analizar estos grandes volúmenes de opiniones a base de poder computacional obteniendo resultados prometedores. De ellos, [17-20] han hecho un enfoque especial en la detección de opiniones informativas para reducir considerablemente el número de opiniones a analizar mediante un filtro. Entre estos trabajos destacan los resultados de [21-23] que emplean modelos de lenguaje de arquitectura Transformer mediante técnicas de ajuste fino.

Una práctica muy recomendada en tareas de análisis de datos es aprovechar información externa de dominio que pueda apoyar la técnica empleada [24, 25]. Sin embargo, no se ha experimentado con la incorporación de este tipo de conocimiento a modelos de lenguaje en la detección de opiniones informativas. El empleo de estas técnicas de soporte en conjunto con los modelos podría representar una mejora significativa a este proceso de detección de relevancia.

Además, investigaciones como [26-28] han abordado la generación automática de requisitos de software, sin embargo, todas se han enfocado en la identificación inicial de requisitos previa al desarrollo del software, mostrando resultados útiles en la generación de borradores e identificación de requisitos pasados por alto. En la literatura revisada no existe ninguna investigación que haya explorado el potencial que podría tener el uso de opiniones de usuarios en la generación automática de nuevos requisitos de software potenciales para mejorar una aplicación ya desplegada en función de las necesidades de sus usuarios.

Ante las cuestiones analizadas se llega a la siguiente **situación problemática**:

- Las opiniones de los usuarios constituyen una fuente de información valiosa para los desarrolladores de software, pero su gran volumen hace incosteable su análisis manual completo.
- Es necesario un sistema que pueda procesar automáticamente las opiniones de los usuarios para lo cual primero hay que deshacerse de la gran cantidad de opiniones no informativas.
- Existe una posibilidad de mejora no explotada en la literatura con la incorporación de conocimiento del dominio a modelos de lenguaje de arquitectura Transformer para la detección de opiniones informativas.
- Las opiniones informativas de usuarios podrían tener un gran potencial como fuente para la generación de nuevos requisitos para mejorar las aplicaciones, pero no ha sido comprobado en la literatura.

Esta situación nos permitió identificar como **problema científico** en esta investigación:

¿Cómo generar de forma automática requisitos de software a partir del contenido de grandes volúmenes de opiniones de usuarios?

El problema definido se encuentra enmarcado en el **objeto de estudio** de la ingeniería de requisitos basada en datos y tiene como **campo de acción** la generación automática de requisitos de software.

Para dar respuesta al problema planteado se han definido los siguientes objetivos:

Objetivo General: Desarrollar un método de generación de requisitos de software a partir de grandes volúmenes de opiniones de usuarios.

Para lograr el cumplimiento del objetivo propuesto se determinaron los siguientes **objetivos específicos** y **tareas de investigación:**

1. Caracterizar fundamentos teóricos sobre el procesamiento de lenguaje natural, la ingeniería de requisitos, y soluciones reportadas sobre la generación de requisitos de software**.**

   Tareas asociadas:

1.1 Investigar las técnicas de clasificación de texto utilizando aprendizaje profundo.

1.2 Investigar las técnicas de agrupamiento de texto.

1.3 Explorar los métodos para generación de texto utilizando *LLMs* más utilizados.

1.4 Estudiar el funcionamiento de la arquitectura Transformer.

1.5 Estudiar el proceso de ajuste fino de modelos de lenguaje y las formas de incorporarle conocimiento de dominio.

1.6 Realizar un estudio de las soluciones anteriores presentes en la literatura sobre la clasificación de opiniones de usuarios.

1.7 Realizar un estudio sobre soluciones de generación de requisitos de software en la literatura.

2 Desarrollar un método de generación de requisitos a partir de opiniones de usuarios.

Tareas asociadas:

2.1 Diseñar un método de filtrado de opiniones de usuarios para soporte de software utilizando modelos de lenguaje y conocimiento del dominio

2.2 Diseñar un método de agrupamiento para opiniones de usuarios.

2.3 Diseñar un método de generación de requisitos a partir de opiniones de usuarios utilizando *LLMs*.

2.4 Construir las plantillas de *prompts* como indicadores de las tareas definidas en el diseño.

2.5 Establecer los parámetros y condiciones necesarias para los modelos y algoritmos seleccionados.

2.6 Implementar los métodos planteados.

2.7 Integrar los métodos de clasificación, agrupamiento y generación en un solo flujo.

3 Evaluar mediante experimentos los resultados de la solución propuesta

Tareas asociadas:

3.1 Seleccionar los sets de datos en los que se evaluará la solución.

3.2 Seleccionar los modelos pre-entrenados de arquitectura Transformer con los que se evaluará la etapa de clasificación.
3.3 Definir las propiedades indicadoras de calidad en requisitos que serán utilizadas
3.4 Realizar experimentos con cada uno de los modelos en cada set de datos para la etapa de clasificación.
3.5 Evaluar el impacto de la incorporación o no de conocimiento del dominio en la etapa de clasificación.
3.6 Comparar los resultados de la clasificación con otras soluciones en la literatura.
3.7 Evaluar la mejor combinación de parámetros general para el algoritmo de agrupamiento a través de experimentos.
3.8 Analizar la calidad individual de los requisitos obtenidos de un caso de estudio a partir de las propiedades definidas.

El presente informe que contiene los resultados de la investigación se encuentra estructurado de la siguiente forma:

Capítulo 1. Fundamentación teórica: Contiene los elementos que conforman el marco teórico del objeto y campo de estudio correspondientes. Se presentan los principales aspectos de la ingeniería de requisitos que involucran la investigación. Se explican además los principales elementos de las distintas etapas de la solución: clasificación, agrupamiento y generación de textos, así como diferentes enfoques utilizados en la literatura para cada uno.

Capítulo 2: Método de solución planteado: Se fundamenta y describe el método propuesto para la generación de requisitos de software y las 3 etapas fundamentales de este proceso: la detección de opiniones informativas a través de una clasificación, así como la forma en que se incorporó el conocimiento de dominio a este proceso; el agrupamiento de las opiniones informativas detectadas en clústeres temáticos; y la propia generación de requisitos a partir de cada clúster de opiniones afines. Se plantean además elementos metodológicos sobre la solución.

Capítulo 3: Validación y análisis de los resultados: Se exponen los resultados de evaluación de los modelos entrenados a partir de la realización de experimentos con un enfoque comparativo para la etapa de clasificación. También, se comprueba la mejor combinación de parámetros de reducción de dimensionalidad para la etapa de agrupamiento. Finalmente se analiza la calidad de los requisitos generados de un caso de estudio.

Finalmente se presentan las **Conclusiones**, se emiten las **Recomendaciones** derivadas de la investigación y se listan las **Referencias Bibliográficas** consultadas.

Capítulo 1: Fundamentación teórica

En este capítulo se abordarán las principales cuestiones teóricas referentes al problema de investigación y técnicas del estado del arte al respecto.

El presente capítulo establece la fundamentación teórica indispensable para abordar la generación automática de requisitos de software a partir de comentarios de usuarios. La exposición se organiza en tres pilares. Primero, se delimita el dominio del problema dentro de la Ingeniería de Requisitos basada en datos, definiendo los artefactos que se buscan generar. Segundo, se construye una base sólida sobre los conceptos fundamentales del Procesamiento de Lenguaje Natural (PLN), abarcando desde el pre-procesamiento y las distintas formas de representación textual, hasta los modelos de aprendizaje profundo como los *Transformers* y las estrategias de transferencia de aprendizaje. Finalmente, se detalla cómo estos conceptos se aplican para ejecutar las tres tareas computacionales clave de esta tesis: la clasificación para identificar la naturaleza de los comentarios, el agrupamiento para descubrir temas recurrentes y la generación de texto para articular los requisitos. Cada tarea se presenta con sus técnicas y métricas de evaluación específicas. El capítulo concluye con una revisión de los antecedentes en el área, las herramientas tecnológicas empleadas y las conclusiones parciales derivadas de este marco teórico.

1.1 Ingeniería de requisitos basada en datos

Los requisitos de un sistema son las descripciones de los servicios que debe prestar un sistema y las limitaciones de su funcionamiento. Estos requisitos reflejan las necesidades de los clientes de un sistema que responde a un determinado propósito, como controlar un dispositivo, hacer un pedido o encontrar información. El proceso de averiguar, analizar, documentar y comprobar estos servicios y restricciones se denomina ingeniería de requisitos según [1].

Es común que muchos softwares no tengan la aceptación o desempeño deseados debido a un mal manejo e identificación de los requisitos del software. Por ejemplo los autores de [29] condujeron un estudio sobre los principales problemas en diversos

aspectos de la ingeniería de requisitos sobre una muestra de 228 compañías de desarrollo de software. En el estudio anterior se destacaron 3 causas principales de problemas durante el desarrollo de software relacionadas con los requisitos:

- Requisitos incompletos y/o ocultos
- Cambio de objetivos, procesos de negocio y/o requisitos
- Requisitos no especificados que son demasiado abstractos

Estos problemas suelen estar relacionados con la dificultad que representa involucrar a los usuarios y sus necesidades en los procesos de ingeniería de requisitos [9]**.** Además, esto es aún más notorio en entornos de desarrollo que cambian rápidamente, como por ejemplo en metodologías de desarrollo ágil de software, donde se van replanteando muchos aspectos del desarrollo a medida que este avanza [30]**.**

La ingeniería de requisitos convencional suele implicar a los usuarios a través de entrevistas, talleres y grupos de discusión. Recientemente, los proveedores de software también han empezado a recoger las opiniones de los usuarios a través de canales de redes sociales, foros de usuarios y sistemas de revisión. En particular, con la aparición de las tiendas de aplicaciones como un mercado de software e infraestructura de despliegue, los usuarios pueden enviar fácilmente sus comentarios, revisar nuevas versiones, informar de errores, calificar aplicaciones y sus características, o solicitar nuevas características [3, 9]. El aprovechamiento de este tipo de datos por los desarrolladores para identificar, priorizar y gestionar los requerimientos de sus productos de software es conocido como ingeniería de requisitos basada en datos (*data-driven requeriments engeneering*) [3].

1.1.1 Requisitos de Software

Un requisito no es simplemente una característica técnica; es una declaración que captura un servicio, una restricción o un objetivo del sistema.

Según *Ian Sommerville* [1], los requisitos de un sistema son:

"...las descripciones de los servicios que un sistema debe proveer y las restricciones en su operación. Estos requisitos reflejan las necesidades de los clientes para un sistema que sirve a un propósito determinado..."

Es crucial distinguir entre los diferentes niveles de abstracción en los que se pueden expresar los requisitos. Esta distinción es una fuente común de problemas si no se gestiona adecuadamente. *Sommerville* [1] propone la siguiente diferenciación:

Requisitos de Usuario (*User Requirements*): Son declaraciones en lenguaje natural, complementadas con diagramas, sobre los servicios que el sistema debe proveer y las restricciones bajo las cuales debe operar. Están redactados para ser comprensibles por los usuarios finales y los clientes, que no necesariamente tienen conocimientos técnicos detallados.

Requisitos del Sistema (*System Requirements*): Son descripciones más detalladas de las funciones, servicios y restricciones operativas del sistema. Este documento, a veces llamado especificación funcional, debe definir con exactitud lo que se va a implementar y puede formar parte del contrato entre el comprador del sistema y los desarrolladores.

1.1.1.1 Tipos de requisitos

La clasificación más fundamental de los requisitos los divide según su naturaleza. Sommerville [1] los clasifica de la siguiente manera:

Requisitos Funcionales (*Functional Requirements*): Describen la funcionalidad o los servicios que el sistema debe proveer. Definen lo que el sistema "debe hacer": cómo debe reaccionar a entradas particulares y cómo debe comportarse en situaciones específicas. Pueden especificar cálculos, manipulación de datos, procesamiento y otras funcionalidades específicas.

Requisitos No Funcionales (*Non-Functional Requirements*): Son restricciones sobre los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. A menudo se aplican al sistema en su conjunto, en lugar de a características individuales. Ejemplos comunes son la fiabilidad, el tiempo de respuesta, la portabilidad, la seguridad y la usabilidad. Frecuentemente, son más críticos que los requisitos funcionales, ya que un fallo en cumplirlos puede hacer que todo el sistema sea inutilizable.

1.1.1.2 Propiedades de calidad en requisitos

Para que una Especificación de Requisitos de Software (ERS) sea de alta calidad, cada requisito individual debe poseer un conjunto de características deseables. El estándar IEEE 830-1998 [31] establece un marco excelente para evaluar la calidad de los requisitos:

- **Correcto:** Un requisito es correcto si y solo si refleja una necesidad real del sistema. No debe haber errores en la interpretación de la necesidad del cliente.
- **No Ambiguo:** Cada requisito debe tener una única interpretación posible. El lenguaje natural es inherentemente ambiguo, por lo que se debe tener especial cuidado, utilizando glosarios, diagramas o notaciones formales para clarificar términos.
- **Completo:** La especificación debe incluir todos los requisitos relevantes y todas las respuestas posibles del sistema a todos los datos de entrada posibles, tanto válidos como no válidos.
- **Consistente:** Los requisitos no deben ser contradictorios entre sí. No se puede, por ejemplo, solicitar que una acción ocurra tras pulsar un botón y, en otro requisito, que la misma acción ocurra automáticamente sin intervención del usuario.
- **Clasificado por Importancia y/o Estabilidad:** Los requisitos deben ser clasificados para asignar recursos de manera eficiente. La importancia puede ser "esencial", "condicional" u "opcional". La estabilidad indica la probabilidad de que el requisito cambie en el futuro.
- **Verificable (*Testeable*):** Debe existir un proceso finito y económicamente viable para comprobar que el sistema cumple con el requisito. Requisitos subjetivos como "el sistema debe ser fácil de usar" no son verificables a menos que se definan métricas objetivas (p. ej., "un usuario experto debe poder completar la tarea X en menos de 2 minutos").
- **Modificable:** La estructura de la especificación debe permitir que los cambios se realicen de forma fácil, completa y consistente, sin afectar a otros requisitos de manera no controlada.

- **Trazable (*Rastreable*):** Debe ser posible rastrear el origen de cada requisito (trazabilidad hacia atrás) y seguir su implementación a través del diseño, el código y las pruebas (trazabilidad hacia adelante).

1.2 Procesamiento del lenguaje natural (PLN)

1.2.1 Pre-procesamiento

Antes de que un modelo procese texto para una tarea específica, a menudo es necesario preprocesar el texto para mejorar el rendimiento y desempeño del modelo. El pre-procesamiento consiste en transformar el texto antes de analizarlo, identificando las unidades (por ejemplo, palabras y frases) que se van a utilizar (tokenización), eliminando el contenido irrelevante para algunas tareas (por ejemplo, eliminar los caracteres no alfabéticos y las palabras vacías), aglomerar los términos relacionados semánticamente para reducir la dispersión de datos y aumentar la capacidad de predicción (conversión a minúsculas, corrección de faltas de ortografía, expansión de contracciones/abreviaturas y stem/lemmatización) y aumentar la cantidad de información semántica que se captura (tratamiento de la negación) [32, 33].

Debido al carácter ruidoso que tienen por naturaleza las opiniones de los usuarios este proceso cobra aún más importancia para hacerle una limpieza al texto que lo libre de cualquier información no necesaria que pueda perturbar al análisis. Esto fue comprobado por el autor de [16] quién realizó pruebas sobre su modelo después de eliminar el paso de pre-procesamiento y experimentó una caída en el desempeño en todas las métricas evaluadas. Sin embargo, esto significa que el pre-procesamiento también puede eliminar información útil (por ejemplo, suprimir las palabras vacías cuando son relevantes para la investigación), introducir errores en el análisis (por ejemplo, cuando el *stemming* confunde palabras semánticamente distintas) y alterar drásticamente los resultados posteriores [32]. Por ello es necesario estudiar cada caso concreto y en función de ello evaluar qué técnicas de pre-procesamiento deberían ser utilizadas.

1.2.2 Representaciones de texto

Un paso importante que requiere cualquier procedimiento de PLN es la proyección de las características del texto en un espacio de características elegido. Debido a su falta de estructura (desde el punto de vista computacional), es necesario aplicar una serie de operaciones para transformarlo gradualmente en una forma digerible para un ordenador. Esta forma es a través de modelos vectoriales conocidos como *Vector Space Models (VSM)* [34]. En esta sección se tratarán algunas de las formas de representación más comunes.

1.2.2.1 Representaciones clásicas

Bag of Word (BoW): El texto es representado como una bolsa (colección) de palabras, ignorando el orden y el contexto. La intuición básica en que se basa es que supone que el texto perteneciente a una clase determinada en el conjunto de datos se caracteriza por un conjunto único de palabras. Si dos textos tienen casi las mismas palabras, pertenecen a la misma bolsa (clase) [33].

Bag of N-Grams(BoN): funciona dividiendo el texto en trozos de n palabras contiguas (o tokens). Cada trozo se denomina n-grama. El vocabulario del corpus, V, no es más que una colección de todos los n-gramas únicos del corpus de texto. Cada documento del corpus está representado por un vector de longitud |V|. Este vector contiene simplemente los conteos de frecuencia de los n-gramas presentes en el documento y cero para los n-gramas que no están presentes [35].

TF-IDF: Su objetivo es cuantificar la importancia de una palabra determinada en relación con otras palabras del documento y del corpus. Si una palabra w aparece muchas veces en un documento d_i, pero no aparece mucho en el resto de documentos d_j del corpus, entonces la palabra w debe tener una gran importancia para el documento d_i. La importancia de w debe aumentar en proporción a su frecuencia en d_i, pero al mismo tiempo, su importancia debe disminuir en proporción a la frecuencia de la palabra en otros documentos d_j del corpus [33].

1.2.2.2 Representaciones distribuidas

Mientras que los métodos de representación vistos en el epígrafe anterior se centraban en capturar las representaciones sintácticas de las palabras y, en algunos casos, un

pequeño subconjunto de las relaciones sintácticas que las unen en las frases, siguen careciendo de la capacidad de capturar su significado semántico. Un ejemplo clásico de este problema lo representan los sinónimos de palabras: aunque son semánticamente iguales, estos modelos no pueden captar su similitud. [34].

Para resolver el problema anterior se desarrollaron los *words embeddings*, esta técnica de aprendizaje tiene como objetivo aprender una correspondencia entre cada fragmento de texto (normalmente palabras, de ahí su nombre) y un vector n-dimensional de números reales. Un *embedding* es, por tanto, una representación vectorial, digerible por una máquina, pero que también codifica parte del significado subyacente de las palabras [33]. Estos enfoques se basan en redes neuronales, que aprenden estos mapeados mediante distintos procedimientos de aprendizaje; en general, se basan en el supuesto de que el significado de una palabra puede extraerse de las palabras que la rodean en una frase [34].

Afortunadamente, en muchos casos no es necesario entrenar nuestros propios *embeddings*, y a menudo basta con utilizar *words embeddings* ya entrenados. Estas pueden verse como una gran colección de pares clave-valor, donde las claves son las palabras del vocabulario y los valores son los vectores de palabras correspondientes [33]. Algunos de los *embeddings* pre-entrenados más utilizados en el estado del arte son *Word2vec* de Google [36], *GloVe* de Stanford [37], y *fasttext embeddings* de Facebook [38].

1.2.3 Aprendizaje de máquina

El objetivo del aprendizaje de máquina o aprendizaje automático es "aprender" a realizar tareas basándose en ejemplos (llamados "datos de entrenamiento") sin instrucciones explícitas. Para ello, se crea una representación numérica (denominada "características") de los datos de entrenamiento y se utiliza esta representación para aprender los patrones de los ejemplos. [39]. Los algoritmos de aprendizaje automático pueden agruparse en tres paradigmas principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

Aprendizaje supervisado: En esta categoría, se proporcionan a la máquina datos de muestra etiquetados para entrenarla, a partir de los cuales predecirá posteriormente los resultados. [33]. El objetivo es ajustar un modelo que relacione la respuesta con los predictores, con el fin de predecir con exactitud la respuesta para futuras observaciones (predicción) o comprender mejor la relación entre la respuesta y los predictores (inferencia).[39] En otras palabras, consiste en la extracción de una muestra de una población, aprender de ella, y luego aplicar el modelo construido a nuevos ejemplos no etiquetados extraídos de la misma población. Tal como señalan [40] este método requiere una gran cantidad de aplicación humana para construir el modelo, pero a la larga permite realizar más rápidamente una tarea que, de otro modo, sería tediosa.

Aprendizaje no supervisado: Permite a la máquina aprender sin supervisión. Se proporciona a la máquina un conjunto de datos no etiquetados, y se supone que el algoritmo actúa sobre los datos sin ningún tipo de supervisión. El objetivo de esta teoría es reagrupar los elementos de datos de entrada que presentan patrones similares. En este contexto, en cierto modo trabajamos a ciegas; es no supervisado porque carecemos de una variable de respuesta que pueda supervisar nuestro análisis [39]. Este tipo de entrenamiento es explotado sobretodo en situaciones en las que es muy difícil conseguir un set de datos de entrenamiento etiquetado que permita un aprendizaje supervisado.

Aprendizaje reforzado: Trata de métodos para aprender tareas mediante ensayo y error y se caracteriza por la ausencia de datos etiquetados o no etiquetados en grandes cantidades. El aprendizaje se realiza en un entorno autónomo y mejora a través de la retroalimentación (recompensa o castigo) facilitada por el entorno [33].

1.2.4 Aprendizaje profundo y redes neuronales

El aprendizaje profundo (*Deep Learning*) es un campo específico dentro del aprendizaje automático: una forma de aprender representaciones a partir de datos centrado en el aprendizaje de capas sucesivas de representaciones cada vez más significativas [41].

La base de este tipo de aprendizaje son las redes neuronales, estructuras flexibles que pueden ser adaptadas a una gran variedad de contextos [42]. Inspiradas en el principio del procesamiento de la información en los sistemas biológicos. Las redes neuronales

consisten en representaciones matemáticas de unidades de procesamiento conectadas llamadas neuronas artificiales. Al igual que las sinapsis en un cerebro, cada conexión entre neuronas transmite señales cuya fuerza puede amplificarse o atenuarse mediante un peso que se ajusta continuamente durante el proceso de aprendizaje [43]. Una red neuronal se considera profunda generalmente cuando tiene más de 1 capa oculta. Conforme aumentan las capas de la red, su capacidad de abstracción aumenta y se pueden establecer relaciones más complejas [39]. Esto resulta de especial utilidad para el análisis del complejo lenguaje natural con nuevas posibilidades de representación.

Las redes neuronales están compuestas de forma general por 3 tipos de capas de neuronas: capas de entrada, capas ocultas y capas de salida.

Capa de entrada (Input layer): Esta es la primera capa en una red neuronal, y el número de nodos o neuronas en esta capa es igual al número de características que se alimentarían a la red.

Capa oculta (Hidden layer): Una red neuronal puede tener una o más capas ocultas. Son las capas intermedias de una red. En estas capas se derivan las relaciones y los patrones de los datos.

Capa de salida (Output layer): Es la última capa que proporciona la salida para una entrada determinada. El número de nodos de la capa de salida depende del tipo de problema que se esté resolviendo. Por ejemplo, en problemas de clasificación se suele igualar la cantidad de neuronas al número de clases o utilizar una sola en casos de clasificación binaria. [35].

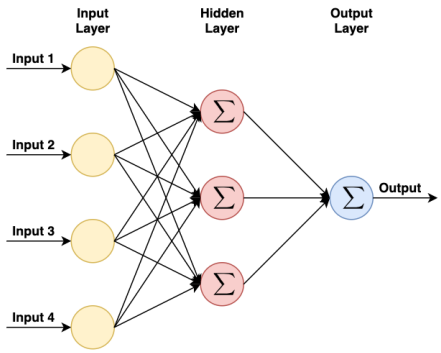


Figura 1.1: Estructura general una red neuronal [35]

Con esta estructura la información es introducida a la red por la capa de entrada con un formato específico la cual es posteriormente procesada por las capas ocultas hasta que llega a la capa de salida que lo convierte en un resultado. Este resultado predicho como es común en el aprendizaje automático se compara con el resultado esperado a través de una función de coste que puntúa que tan buena fue la predicción y penaliza al modelo por sus errores según el tipo o magnitud del error [44]. Es fundamental en el aprendizaje profundo utilizar esta puntuación como señal de retroalimentación para ajustar el valor de los pesos en una dirección que reduzca la puntuación de pérdida para el ejemplo actual. Este ajuste es tarea del optimizador, que implementa lo que se denomina algoritmo de retro-propagación (*back-propagation*). [41].

Inicialmente, a los pesos de la red se les asignan valores aleatorios. Naturalmente, su resultado dista mucho de lo que debería ser idealmente, y el valor de pérdida es, en consecuencia, muy alto. Pero con cada ejemplo que procesa la red, los pesos se ajustan un poco en la dirección correcta y la pérdida disminuye. [41]. Este es el bucle de entrenamiento, que, repetido un número suficiente de veces produce valores de peso que minimizan la función de pérdida. Una representación gráfica de este proceso se puede apreciar en la figura 1.2

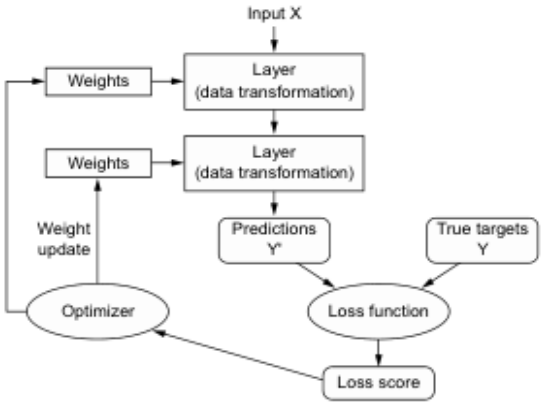


Figura 1.2: Esquema del proceso de entrenamiento de una red neuronal [41]

1.2.4.1 Arquitecturas más utilizadas

Con los años han surgido varias arquitecturas de redes neuronales con diferentes enfoques y capacidades. A continuación, se comentarán algunas de las arquitecturas más comunes en la literatura para la resolución de problemas orientados al PLN:

***Recurrent neural networks* (*RNN*):** Las redes neuronales recurrentes están especialmente diseñadas para tener en cuenta el procesamiento secuencial y el aprendizaje. Las *RNN* tienen unidades neuronales capaces de recordar lo que han procesado hasta el momento. Esta memoria es temporal, y la información se almacena y actualiza con cada a medida que la *RNN* lee la siguiente palabra de la entrada [33]. A pesar de su capacidad y versatilidad, las *RNN* sufren el problema de la memoria olvidadiza: no pueden recordar contextos demasiado largos. Este problema fue abordado creando una variante conocida como ***Long short-term memory*(*LSTM*)** la cual sortea este problema dejando de lado el contexto irrelevante y recordando sólo la parte del contexto que se necesita para resolver la tarea en cuestión [45]. Esta arquitectura fue abordada para el problema concreto de requerimientos de software en [46].

***Convolutional neural networks* (*CNN*):** Las redes neuronales convolucionales son usadas mayormente en visión por computadora, sin embargo, al tratar con representaciones vectoriales de palabras de igual tamaño es posible representar los textos como matrices de dimensión n x d, donde n es el número de palabras de la frase y d es el tamaño de los vectores de palabras [33]. De esta forma se puede tratar al texto como si fuera una imagen. Son capaces de aprender una jerarquía de características y patrones que abstraen información espacial de dichas matrices [47]. Las CNN se componen principalmente de capas convolucionales, seguidas de capas de activación y, a veces, de capas de agrupamiento (*pooling*), que permiten reducir la dimensionalidad de los datos mientras conservan las características más importantes [48]. También se han desarrollado soluciones de clasificación de requerimientos con esta arquitectura [16].

Transformers: arquitectura presentada en [49] y que ha alcanzado el estado del arte en los últimos años en muchas tareas de PLN. Modelan el contexto textual, pero no de forma secuencial. Dada una palabra en la entrada, miran todas las palabras a su alrededor (lo que se conoce como auto-atención) y representan cada palabra con respecto a su contexto [41]. Este mecanismo de atención funciona calculando para cada palabra puntuaciones de relevancia respecto al resto de palabras en la oración.

1.2.5 Transferencia de aprendizaje

Recientemente, se han utilizado grandes *transformers* para el aprendizaje por transferencia de tareas derivadas más pequeñas. El aprendizaje por transferencia es una técnica de IA en la que los conocimientos adquiridos al resolver un problema se aplican a otro distinto pero relacionado [34]. Con los *transformers*, la idea es entrenar un modelo muy grande de forma no supervisada (lo que se conoce como pre-entrenamiento) para predecir una parte de una frase dado el resto del contenido, de modo que pueda codificar los matices de alto nivel del lenguaje en ella [33]. Estos modelos se entrenan con enormes cantidades de datos textuales, extraídos de todo Internet.

Estos modelos pre-entrenados de lenguaje son utilizados como base para re-entrenarlos en tareas específicas como la clasificación o traducción o incluso para mejorar su precisión en un dominio específico [41]. Este proceso en la literatura es conocido como "*fine-tunning*". Entre los modelos más usados en la literatura para tareas de clasificación se encuentran: BERT [21], RoBERTa [22, 50], ALBERT [22], BERTweet [50], entre otros.

1.3 Clasificación de textos

La clasificación de textos es una tarea de importancia fundamental, que ha ido ganando terreno gracias a los recientes avances en los campos de la minería de textos y el procesamiento del lenguaje natural (PLN). Los métodos de clasificación de textos comparten el objetivo común de designar una etiqueta predefinida para un texto de entrada dado. Ejemplos clásicos de clasificación de textos son el etiquetado de temas, el análisis de sentimientos, la clasificación de noticias y el filtrado de spam. [34].

En general, el sistema de clasificación de textos contiene cuatro niveles diferentes de alcance que pueden aplicarse [48]: (1) Nivel de documento (obtiene las características relevantes de un documento completo); (2) Nivel de párrafo (de un párrafo o porción del documento); (3) Nivel de Frase (de una sola oración); y (4) Nivel de subsentencia (de expresiones dentro de una oración) .

1.3.1 *Fine-tuning* de modelos Transformers para clasificación

Tal como se mencionó anteriormente los modelos pre-entrenados de lenguaje son entrenados con una gran cantidad de datos de texto sin etiquetas, como libros, artículos y sitios web. El objetivo es capturar los patrones, estructuras y conocimiento semántico subyacente en el corpus de texto. Este proceso es un aprendizaje no supervisado, donde los modelos aprenden de los datos de texto sin orientación explícita o etiquetas [33, 35]. Debido a esto uno de estos modelos no puede ser utilizado directamente para realizar una tarea de clasificación, primero deben sufrir algunas modificaciones en su estructura.

Primero que nada, debe ser adaptada la entrada del modelo al tamaño que se estime necesario para cada situación particular, pues este tamaño de entrada determinará la capacidad máxima de tokens que pueden ser procesadas por cada petición, esto en una tarea de clasificación de texto se corresponde a la cantidad de palabras (incluyendo tokens especiales en muchas ocasiones) que puede contener el texto a clasificar. Cuando el tamaño del texto entrado no coincide con la entrada del modelo se deben tomar medidas para ajustar dicho tamaño al esperado. En caso de ser menor la entrada la mayoría de autores [21, 22, 50] deciden aplicar un "*padding*" que consiste con rellenar la entrada con tokens especiales ignorados por el modelo para completar el tamaño. De lo contrario, si es mayor, por lo general la entrada se trunca para forzar el tamaño requerido. Como es de suponer este último método puede llevar a la pérdida de información significativa para definir la clasificación, por ello es muy importante elegir bien este tamaño de entrada para cada dominio específico de forma que la mayoría de ejemplares a clasificar se encuentren dentro del tamaño de entrada seleccionado [41].

Una vez definida la entrada, el interior del modelo se mantiene estructuralmente igual y se debe decidir qué parte de su salida será utilizada posteriormente en las capas adicionales de clasificación. Por ejemplo autores como [21] utilizan el primer token de la salida como representación del texto completo, mientras otros como [50] utilizan la media o suma de los tokens de salida para obtener un valor más representativo.

La ya mencionada agregación de capas de neuronas adicionales a la salida del modelo de lenguaje es imprescindible para convertir su salida en un valor de decisión. Uno de los enfoques más comunes utilizados por [21, 22, 50] es utilizar una capa lineal que

agrupe la información contenida en la salida en la cantidad de clases posibles y finalmente, a través de una función de activación, obtener una predicción. En el caso de una clasificación binaria la salida se agruparía en una sola neurona que representaría la probabilidad de pertenencia a la clase positiva, la función de activación (generalmente sigmoide) a partir de un umbral de decisión clasificaría el resultado en una de las clases opuestas, en la figura 1.3 (Extraída de [51]) se puede ver una representación de esta estructura. Si las 2 clases a clasificar no son exactamente opuestas se puede utilizar un enfoque multi-clase de 2 clases con una función de activación *argmax* que elija la clase de mayor probabilidad entre los resultados.

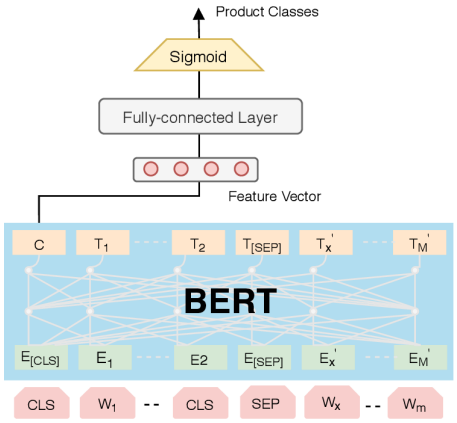


Figura 1.3: Estructura común de un modelo fine-tuned para una tarea de clasificación

1.3.1.1 Modelos pre-entrenados *Transformers* de interés

Se consideraron diferentes modelos pre-entrenados de arquitectura Transformer en este trabajo para la etapa de clasificación. A continuación, se describen los modelos que se consideraron de interés:

1. ***RoBERTa* (*Robustly Optimized BERT Pretraining Approach*)** [52]: Basado en la arquitectura de BERT, modificando los pasos de pre-entrenamiento de *BERT* para mejorar sustancialmente el rendimiento en todas las tareas, sobre todo de clasificación. *RoBERTa* aumentó también la cantidad de tamaños de mini-batch, datos y tiempo de entrenamiento para entrenar el modelo. Además, se entrena en un conjunto de datos que incluye secuencias más largas que su predecesor, y el patrón de enmascaramiento también se modificó para que se generara espontáneamente. *RoBERTa* fue seleccionado debido a que superó a todos los

modelos similares en su lanzamiento y todavía hoy sigue aportando resultados muy competitivos en muchas tareas de procesamiento de lenguaje natural.

2. ***ALBERT* (*A Little BERT*)** [53]: como su nombre indica es un modelo basado en la arquitectura *BERT* con la particularidad de ser pequeño, producto a la reducción del tamaño de los parámetros. En su construcción se aplicaron tres técnicas de reducción de parámetros: Parametrización de incrustación factorizada (*Factorized embedding parameterization*), Compartición de parámetros entre capas (*Cross-layer parameter sharing*) y Pérdida de coherencia entre frases (*Inter-sentence coherence loss*). Se escogió este modelo ya que es capaz de conseguir resultados casi al nivel de *BERT* y *RoBERTa* utilizando cantidades de parámetros significativamente menores.
3. ***BERTweet*** [54]: modelo que fue entrenado para procesar tweets (nombre por el que se refiere a los mensajes de la red social twitter, ahora X). Utiliza la misma arquitectura de *BERT* y el proceso de entrenamiento de *RoBERTa*. Su aporte se encuentra en el *dataset* utilizado para su pre-entrenamiento, 850 millones de tweets. Esto le permite tener un conocimiento muy especializado del lenguaje utilizado en la plataforma, haciéndolo ideal para análisis sobre comentarios hechos en *twitter*, en los cuales supera en desempeño tanto a *BERT* como a *RoBERTa*. Este modelo fue seleccionado porque ha superado a *RoBERTa* no solo en comentarios dentro de la red social mencionada, sino también en casos donde el texto analizado es de carácter informal (como el de comentarios de usuarios sobre aplicaciones) debido a la similitud en el vocabulario utilizado.
4. ***XLNet*** [55]: integra los modelos autorregresivos y el modelado bidireccional del contexto, con un mecanismo llamado Modelado de Permutación de Lenguaje (*Permutation Language Modeling* - *PLM*) con la idea de capturar el contexto bidireccional en todas las permutaciones de términos presentes en una secuencia de entrada. Se seleccionó porque ha superado a *RoBERTa* en algunas investigaciones y como punto de referencia fuera de la familia de modelos *BERT*.
5. ***GPT-2*** [56] es la última versión de la familia GPT disponible de forma pública. Fue entrenado en una enorme cantidad de texto de más de 8 millones de páginas web con un único objetivo, dada una oración predecir la próxima palabra. Este objetivo

implica que cada token es analizado en función de únicamente la información obtenida de los tokens previos a él. Aunque obtuvo resultados al nivel del estado del arte del momento en que fue lanzado, hoy en día es superado en la mayoría de casos para tareas de clasificación por el resto de modelos mencionados. A pesar de esto, se seleccionó como punto comparativo para ilustrar el avance de los modelos pre-entrenados de lenguaje.

1.3.2 Incorporación de conocimiento del dominio

A pesar de los avances de los modelos pre-entrenados Transformers (MPT), estos presentan varias limitaciones. En primer lugar, aunque aprenden bien la semántica de las palabras comunes, su desempeño es deficiente con palabras poco frecuentes, debido a la distribución desigual de los datos disponibles. Además, aunque los MPT pueden capturar una gran cantidad de conocimiento lingüístico, semántico y factual, su desempeño en la extracción de conocimiento y el razonamiento depende en gran medida de indicaciones sesgadas. Por último, pueden generar oraciones que carecen de sentido común, lo que plantea problemas éticos y de responsabilidad, especialmente en tareas donde pueden superar el desempeño humano [25].

Para abordar estas limitaciones, una solución prometedora es combinar redes neuronales con conocimiento simbólico o específico del dominio concreto en que se piensa utilizar [24]. Por un lado, el uso de conocimiento simbólico puede mejorar el manejo de palabras poco frecuentes, ya que estos recursos suelen tener una mayor cobertura de términos extraños, complementando la supervisión textual limitada de los MPT. Por otro lado, el conocimiento simbólico también proporciona información relacional explícita y reglas que fortalecen las capacidades de razonamiento de los modelos. Además, esta integración puede mejorar la interpretabilidad de los MPT, permitiendo entender mejor cómo utilizan el conocimiento [57]. Incluso enfoques más simples enfoques con menor cantidad de información orientados a potenciar la atención de los MPT en términos concretos de mayor interés pueden tener un resultado positivo. También se han hecho investigaciones incorporando conocimiento de uso general del lenguaje o conocimiento causal, tal es el caso de [58, 59]. Por cuestiones de comodidad

a partir de aquí se referirá a los MPT que incorporan conocimiento como MPTC (Modelo Pre-entrenado Transformer con Conocimiento).

Los MPTC integran diferentes granularidades de conocimiento para su uso en escenarios que requieren información a diferentes niveles de detalle. En función de la granularidad los MPTC se podrían dividir en MPTC fusionados con texto, fusionados con árboles sintácticos, fusionados con grafos de conocimiento y fusionados con reglas.

Los MPTC fusionados con texto hacen uso de textos que suelen contener descripciones detalladas de entidades, relaciones y acontecimientos. Estos modelos utilizan los textos como referencias externas, extrayendo información crítica que beneficia principalmente a las tareas de respuesta a preguntas [25].

En el caso de los MPTC fusionados con árboles sintácticos transforman los árboles en representaciones semánticas y los combinan con representaciones de palabras, o los utilizan para seleccionar constituyentes críticos de secuencias de entrada para enmascararlos [24, 60]. Esta forma de conocimiento resulta beneficiosa para tareas que tienen en cuenta la estructura, como el análisis sintáctico, el etiquetado semántico de roles y la extracción de relaciones.

Con el avance de las técnicas de extracción de información, han surgido una multitud de grafos de conocimiento generales y específicos de dominio. Los grafos de conocimiento proporcionan una forma estructurada de representar información rica en forma de entidades y relaciones entre ellas [61]. Un enfoque implica el aprendizaje de representaciones para los grafos y la posterior fusión de sus representaciones con representaciones de palabras alineadas mediante un módulo de infusión como es el caso de *ERNIE* [62].

También, las reglas pueden caracterizarse como directrices formales y su principal ventaja reside en la interpretabilidad y responsabilidad que ofrecen las estructuras matemáticas estrictas y un proceso de inferencia transparente [63]. Los MPTC pueden emplear este razonamiento simbólico, a partir de sus resultados de predicción.

El método de inyección de conocimiento desempeña un papel importante en la eficacia y la eficiencia de la integración entre los MPT y el conocimiento. De hecho, dicta el tipo

de conocimiento que puede integrarse y su forma [25]. A continuación, se tratarán algunas de las formas de inyección utilizadas en la literatura.

1.3.2.1 Inyección directa de texto

Una primera estrategia sencilla para la integración de conocimientos consiste en ampliar directamente los textos de entrada con conocimientos adicionales. Este texto enriquecido se emplea tanto en la fase de ajuste fino como en el posterior procedimiento de clasificación. La aplicación más sencilla de esta solución consiste en convertir toda la información pertinente del grafo de conocimiento en una cadena que se añade al final del texto original. Un ejemplo de esto es la estructura utilizada en [64] .

1.3.2.2 Vectores de características

En lugar de inyectar directamente los conocimientos en el texto, es posible incorporarlos como datos de características adicionales durante el proceso de clasificación. Primero, el MPT procesa el texto y devuelve las representaciones pertinentes. A continuación, esta salida se concatena con características adicionales derivadas de la representación del elemento en una base de conocimientos [25, 64]. Dicha concatenación implica la fusión de 2 vectores, el vector con la información de salida del MPT y el vector de características del dominio. Esta representación aumentada se introduce en la capa de clasificación final, comúnmente una capa linear. Sin embargo, también hay enfoques que utilizan un Perceptrón Multicapa (PM) para llevar a cabo esta tarea o incluso una combinación de ambos como se puede ver en el ejemplo de la figura 1.5 que representa la estructura seguida por [65].

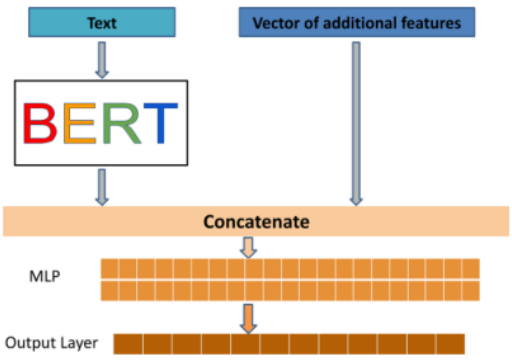


Figura 1.4: Integración de un vector de características en un MPT

1.3.2.3 Pre-entrenamiento adicional

Otra estrategia de incorporación de conocimiento incluye la realización de pre-entrenamiento adicional de forma no supervisada sobre corpus de texto estrechamente relacionados al dominio concreto al que se busca especializar el modelo, tal es el caso de [66]. Hay autores que incluso han utilizado fuentes de datos estructuradas como grafos de conocimiento y los ha convertido a contenido textual preparado para esta tarea e incluso integrando ambos enfoques. Aunque este método en muchos casos ha dado resultados positivos también demanda una cantidad recursos de procesamiento significativa en comparación al resto de enfoques. Una variante es realizar un pre-entrenamiento desde cero con la estrategia de entrenamiento y corpus de texto originales del modelo adicionando los corpus de texto adicionales del dominio objetivo desde esa fase de entrenamiento inicial tal como hacen en [22], esto es naturalmente aún más costoso computacionalmente que la variante originalmente explicada.

Este enfoque es muy común en la especialización de los modelos a un campo de investigación concreto sin ligarlos a una tarea concreta de minería. Por ejemplo, en [61] se analizan varios MPTC especializados como *BioBERT* [67], *BioELMo* [68] y *BlueBERT* [69] en el campo médico que son MPT normales a los que se les realizó un proceso de pre-entrenamiento extra sobre textos médicos. Además en [70] evalúan diferentes enfoques de pre-entrenamiento para introducir información específica de un entorno *e-comerce* concreto en un modelo con buenos resultados.

1.3.3 Evaluación de la clasificación

Al entrenar un modelo de inteligencia artificial como en otras tareas es de vital importancia cuantificar que tan bien desempeña su tarea. Esto se hace mediante técnicas y métricas de evaluación que no son más que fórmulas que puntúan el desempeño del modelo según aspectos concretos en función del tipo de métrica utilizada [44]. Además, permiten comparar el desempeño entre distintos modelos.

El método de evaluación más simple es la utilización de un set de datos que esté explícitamente dividido en un conjunto de entrenamiento y un conjunto de prueba de forma que los resultados de las métricas obtenidas con él sean replicables bajo esas condiciones [71]. Sin embargo, no todos los sets de datos están tan claramente divididos,

así que es común utilizar una validación cruzada en la que el set de datos es manualmente dividido 2 conjuntos, uno de entrenamiento y otro de prueba

Muchos autores [21, 22, 50] en estos casos utilizan una variante de la validación cruzada, la validación cruzada de K grupos (K-folds cross-validation) porque proporciona una estimación más robusta de la capacidad de generalización del modelo [41]. Para realizarla se divide aleatoriamente el conjunto de datos en un número K de subconjuntos o "*folds*". El modelo se prueba en uno de estos subconjuntos y se entrena en los K-1 subconjuntos restantes. Este proceso se repite K veces, con diferentes combinaciones de conjuntos de entrenamiento y prueba, hasta que cada subconjunto haya sido usado como conjunto de prueba una vez. De este proceso se obtienen K resultados de métricas, estos son promediados para obtener los resultados de la evaluación [39].

1.3.3.1 Métricas para clasificación

Las métricas comunes en problemas de clasificación suelen partir de una matriz de confusión como la de la tabla 1.1 que lleva un conteo según la cantidad de casos clasificados de cierta forma.

Tabla 1.1 Matriz de confusión para una clasificación específica

	Clasificación verdadera	
Clasificación predicha	Positive (P)	Negative (N)
Positivo (P)	True Positive (TP)	False Positive (FP)
Negativo (N)	False Negative (FN)	True Negative (TN)

Donde se cumple que **TP** es la cantidad de casos en los que se predice que el elemento pertenece a la clase correctamente. **FN** es la cantidad de casos en los que se predice que el elemento no pertenece a la clase cuando esto es falso. **FP** es la cantidad de casos en los que se predice que el elemento pertenece a la clase cuando esto es falso. **TN** es la cantidad de casos en los que se predice que el elemento no pertenece a la clase correctamente.

Algunas de las métricas más comunes encontradas en la literatura son las siguientes:

***Accuracy*:** Es la relación entre las muestras clasificadas correctamente y el número total de muestras. Es una métrica sencilla e intuitiva, pero puede inducir a error cuando la distribución de clases está desequilibrada y hay demasiada representatividad de una sola clase [71]. Matemáticamente, puede representarse como:

$$\boldsymbol{Accuracy} = \frac{\mathbf{TP} + \boldsymbol{TN}}{\mathbf{TP} + \mathbf{FP} + \mathbf{TN} + \mathbf{FN}}$$

***Precision*:** La precisión mide la exactitud de las predicciones positivas. Más específicamente, denota la proporción de casos Predichos Positivos que son correctamente Positivos Reales [34]. Una precisión alta significa que el modelo no genera muchos falsos positivos. Matemáticamente se puede representar como:

$$\boldsymbol{Precision} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}}$$

***True Positive Rate / Recall*:** Mide la proporción de instancias verdaderas positivas (es decir, instancias correctamente clasificadas como positivas) del número total de instancias positivas [44]. Un valor alto de *recall* indica que el modelo tiene menos falsos negativos, lo que significa que puede identificar correctamente la mayoría de los casos positivos. Matemáticamente se representa de la siguiente forma:

$$\mathbf{Recall} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}}$$

F1 score: Métrica que busca equilibrar los valores de *precision* y *recall* calculando su media armónica [71]. De especial utilidad cuando se quiere buscar una precisión lo más general posible y balanceada. Matemáticamente se representa como:

$$\boldsymbol{F}1 = 2 * \frac{\mathbf{precision} * \mathbf{recall}}{\mathbf{precision} + \mathbf{recall}}$$

1.3.4 Antecedentes de clasificación de opiniones de usuarios

En el campo de la clasificación de comentarios de usuarios según su relevancia para equipos de desarrollo de software ha habido varias investigaciones que se han centrado en este aspecto [17-19, 66] y otras que han utilizado un enfoque más general de clasificación teniendo en cuenta aspectos adicionales como reportes de errores o petición de características [21, 22, 50].

Los autores de [17] crearon una herramienta llamada *AR-Miner* para analizar grandes cantidades de comentarios de usuarios. La primera etapa del proceso de esta herramienta es el filtrado de las opiniones irrelevantes. Para ello utilizaron un algoritmo de entrenamiento semi-supervisado, *Expectation Maximization for Naive Bayes* (*EMNB*) que permite aprovechar las grandes cantidades de comentarios sin etiquetar juntos a una cantidad significativamente menor de datos etiquetados.

En [18] los autores analizan la relevancia de opiniones de usuarios realizando un análisis bastante exhaustivo probando varios enfoques tradicionales de aprendizaje automático como *Support Vector Machine*, *Naive Bayes*, *Logistic Regression*, *k-nearest neighbors*, entre otros. Dichos algoritmos probados con distintas técnicas de extracción de características como *TF-IDF*, *PMI* y la ocurrencia de palabras propias del campo de dominio a través de un glosario de términos. En el estudio evalúan la solución para distintas combinaciones de algoritmos y técnicas de extracción de características.

Los autores de [21] no analizan específicamente la relevancia sino un enfoque de clasificación entre irrelevante, reporte de error y petición de características. La tarea de clasificación es realizada a través del *fine-tunning* de los modelos de lenguaje de arquitectura *Transformer English-BERT*, *M-BERT* e *Italian-BERT*.

El primer caso dentro de esta problemática donde se aplicó una técnica que incorporaba conocimiento del dominio fue [66]. En este trabajo sus autores realizan un pre-entrenamiento adicional de modelos basados en *BERT* en *datasets* de diferentes campos y diferentes objetivos. Uno de estos fue precisamente en un conjunto de más de 160 mil opiniones sobre aplicaciones tomado de [17] para la tarea de clasificación de opiniones. Un caso similar al anterior es el de [22], donde trabajan en varios *datasets* etiquetados para clasificaciones diferentes cada uno. Este estudio también realiza un fine-tunning de varios modelos de lenguaje de arquitectura Transformer como *BERT*, *ALBERT*, *RoBERTa* y *XLNET*. Además, hicieron pruebas con versiones personalizadas de estos modelos en las que hicieron un pre-entrenamiento desde cero con datos adicionales de redes sociales y textos de estilo informal. El enfoque utilizado en estos 2 últimos casos cae dentro de la incorporación de conocimiento adicional mediante pre-entrenamiento adicional, explicado en la sección 1.3.2.3.

1.4 Agrupamiento de texto

El agrupamiento de texto es una técnica esencial en el análisis de datos textuales, especialmente relevante ante el crecimiento exponencial del contenido digital no estructurado [33]. Su objetivo es organizar colecciones de documentos en grupos o clústeres, de tal forma que los textos dentro de un mismo clúster exhiban una alta similitud entre sí, mientras se distinguen de aquellos en otros clústeres. Esta metodología, de naturaleza no supervisada, es fundamental para descubrir patrones latentes y temáticas emergentes en corpus donde las categorías no están predefinidas [39]. Aplicaciones como la organización de noticias, el análisis de retroalimentación de clientes o el estudio de contenido en redes sociales se benefician de su capacidad para estructurar la información, facilitando así análisis posteriores más precisos. Cabe destacar que el agrupamiento de textos cortos (*Short Text Clustering - STC*) presenta desafíos particulares debido a la escasez de contexto, problema sin duda presente al analizar opiniones de usuarios que tienden a ser generalmente cortas [72].

1.4.1 Representación y dimensionalidad

La transformación de texto en representaciones numéricas es un pilar para el agrupamiento, ya que los algoritmos operan en espacios vectoriales. Desde la perspectiva del agrupamiento, la calidad de esta representación es vital [33].

Métodos tradicionales como el *TF-IDF* mencionados en secciones anteriores a menudo generan vectores dispersos y de alta dimensionalidad que pueden dificultar la tarea de agrupamiento al no capturar relaciones semánticas profundas [35, 73]. Los *word embeddings* estáticos como *Word2Vec* o *GloVe* mejoraron esta situación al generar vectores densos que reflejan similitudes semánticas [35, 39].

La irrupción de *embeddings* contextuales a partir de modelos Transformers como BERT, incluso los derivados de grandes modelos de lenguaje (*LLMs*) como *GPT* y *LLaMA*, ha marcado un hito. Para el agrupamiento, estos *embeddings* ofrecen una representación más rica y matizada de los documentos, capturando sutilezas del lenguaje que pueden llevar a una discriminación más fina y precisa entre clústeres [73].

Las representaciones textuales, sobre todo las generadas por modelos de lenguaje de gran tamaño, pueden tener una dimensionalidad que afecte negativamente el rendimiento y la interpretabilidad del agrupamiento conocido como la "maldición de la dimensionalidad". La reducción de la dimensionalidad busca mitigar esto, reducir el costo computacional y, potencialmente, mejorar la separación de los clústeres al eliminar dimensiones ruidosas o irrelevantes [39, 72]. Técnicas como el Análisis de Componentes Principales (*PCA*) pueden proyectar los *embeddings* textuales a un subespacio de menor dimensión. También, algoritmos como *t-SNE* (*t-Distributed Stochastic Neighbor Embedding*) y *UMAP* (*Uniform Manifold Approximation and Projection*) son populares para la visualización y reducción no lineal, enfocándose en preservar estructuras locales y/o globales [72]. La investigación de [73] también ha explorado el resumen de texto como una forma de reducción de dimensionalidad previa a la generación de *embeddings*, aunque con resultados mixtos, indicando que la pérdida de información puede ser perjudicial para la discriminación de clústeres.

1.4.2 Algoritmos de agrupamiento

Una vez obtenidas las representaciones vectoriales, se selecciona y aplica un algoritmo de agrupamiento. La elección depende de las características de los datos textuales representados y los objetivos del análisis.

1.4.2.1 Enfoques particionales

Los enfoques particionales buscan dividir el corpus textual en un número predefinido k de clústeres, asignando cada documento a un único grupo. El algoritmo *K-Means* es el más emblemático dentro de esta categoría. Opera asignando iterativamente cada documento al centroide (el punto medio del clúster) más cercano y luego recalculando la posición de los centroides basándose en los documentos asignados. Este proceso se repite hasta que las asignaciones se estabilizan. *K-Means* es apreciado por su eficiencia computacional, especialmente con grandes volúmenes de texto, aunque es sensible a la elección inicial de los centroides y tiende a formar clústeres de forma esférica, lo que no siempre se ajusta a la distribución natural de los datos textuales. Una variante es *K-Medoids* (o *PAM*), que en lugar de calcular centroides como la media de los puntos, utiliza documentos reales del conjunto de datos como "medoides" o centros de clúster. Esta

característica lo hace más robusto a valores atípicos en las representaciones textuales, aunque generalmente implica un mayor coste computacional que *K-Means* [39, 72].

1.4.2.2 Enfoques jerárquicos

Los algoritmos de agrupamiento jerárquico, a diferencia de los particionales, construyen una jerarquía de clústeres, a menudo representada como un árbol o dendrograma, sin necesidad de especificar el número de clústeres de antemano. El Agrupamiento Jerárquico Aglomerativo (*AHC*) es el método más común. Comienza tratando cada documento como un clúster individual y, en pasos sucesivos, fusiona los dos clústeres más similares hasta que todos los documentos pertenecen a un único clúster. La similitud entre clústeres se determina mediante una métrica de enlace (como *single*, *complete*, *average linkage* o *Ward's method*). Este enfoque es útil para explorar agrupaciones textuales a diferentes niveles de granularidad, aunque su complejidad computacional puede ser una limitación para corpus muy extensos, ya que las decisiones de fusión son irrevocables [39, 72].

1.4.2.3 Enfoques basados en densidad

Los algoritmos basados en densidad identifican clústeres como regiones densas de documentos en el espacio de características, separadas por regiones de baja densidad. Son capaces de encontrar clústeres de formas arbitrarias y manejar el ruido. *DBSCAN* (*Density-Based Spatial Clustering of Applications with Noise*) es el algoritmo principal de esta categoría. Agrupa puntos que están densamente empaquetados, identificando puntos núcleo (aquellos con un número mínimo de vecinos dentro de un radio específico) y expandiendo los clústeres a partir de ellos. Los documentos que no pertenecen a ningún clúster denso se consideran ruido. *DBSCAN* es valioso para descubrir estructuras de clústeres no esféricas en datos textuales, pero su rendimiento es sensible a la elección de sus dos parámetros principales (ε y *MinPts*), que pueden ser difíciles de sintonizar, especialmente en espacios de representación textual de alta dimensionalidad [72].

1.4.3 Evaluación del agrupamiento

Una vez que un algoritmo de agrupamiento ha generado una solución, es crucial evaluar su calidad. Esta evaluación permite comparar diferentes algoritmos, optimizar

parámetros y entender las fortalezas y debilidades de un sistema de agrupamiento. Existen principalmente dos enfoques para evaluar la calidad del agrupamiento:

1. **Evaluación Intrínseca:** Su objetivo principal es cuantificar qué tan bien los clústeres capturan la estructura natural de los datos. Se basa en las propiedades inherentes de los clústeres generados, como la cohesión (qué tan similares son los elementos dentro de un clúster) y la separación (qué tan distintos son los clústeres entre sí) [74]. No requiere una "verdad fundamental" con la cual comparar.
2. **Evaluación Extrínseca:** Compara la estructura de clústeres generada por el algoritmo con una estructura de referencia predefinida o "verdad fundamental"[72], usualmente creada por anotadores humanos.

Como la etapa de agrupamiento relacionada a la solución de este proyecto no cuenta con una “verdad fundamental” a la cual comparar los resultados, la evaluación y métricas utilizadas son completamente intrínsecas y solo se explicará en profundidad este tipo durante la sección.

1.4.3.1 Métricas para agrupamiento

El principio fundamental de las métricas de evaluación intrínseca es capturar en su solo valor tanto la cohesión como la separación de los clústeres. Algunas de las métricas más utilizadas en la literatura son las siguientes:

Coeficiente de Silueta (*Silhouette Coefficient*) [75]**:** Para cada documento, el coeficiente de silueta mide qué tan similar es a los documentos de su propio clúster en comparación con los documentos de otros clústeres. Un promedio más alto indica una mejor calidad de agrupamiento con valores desde -1 a 1. El cálculo para un documento i se realiza de la siguiente forma:

- $a(i)$: Distancia promedio de i a todos los demás documentos en su mismo clúster.
- $b(i)$: Distancia promedio mínima de i a todos los documentos de cualquier otro clúster del cual i no es miembro.

- El coeficiente de silueta $s(i)$ para el documento i es: $s(i) = \frac{b(i) - a(i)}{\max(a(i),\ b(i))}$
- El Coeficiente de Silueta general es el promedio de $s(i)$ sobre todos los documentos.

Índice de Davies-Bouldin (*Davies-Bouldin Index - DBI*) [76]**:** Mide la "similitud" promedio entre cada clúster y su clúster más similar, donde la similitud es una relación entre la dispersión intra-clúster y la separación inter-clúster. Valores más bajos indican una mejor partición, ya que los clústeres están más compactos y mejor separados. Un valor de 0 es el mínimo teórico. Para cada clúster C_i , se calcula $R_i = max_{i \neq j}(\frac{S_i + S_j}{d\{i,j\}})$, donde:

- S_i: Medida de dispersión promedio dentro del clúster C_i (por ejemplo, distancia promedio de los puntos al centroide).
- $d\{i,j\}$: Distancia entre los centroides de los clústeres C_i y C_j.
- El DBI es el promedio de R_i, sobre todos los clústeres.

Índice de Calinski-Harabasz (*Calinski-Harabasz Index - CHI / Variance Ratio Criterion*) [77]: Mide la ratio entre la suma de las dispersiones entre clústeres y la suma de las dispersiones intra-clúster, ponderada por el número de clústeres y el número de puntos de datos. Valores más altos indican una mejor calidad de agrupamiento, con clústeres densos (cohesivos) y bien separados. Se calcula como : $CH = (\frac{Traza(B_k)}{k-1}) / (\frac{Traza(W_k)}{N-k})$ Donde:

- N: Número total de documentos.
- k: Número de clústeres.
- B_k: Matriz de covarianza entre grupos (mide la dispersión entre los centroides de los clústeres y el centroide global).
- B_k: Matriz de covarianza dentro de los grupos (mide la dispersión de los documentos dentro de sus respectivos clústeres).
- $Traza()$: Suma de los elementos de la diagonal de una matriz.

1.5 Generación de texto

La generación automática de texto, como desafío central en inteligencia artificial, ha sido moldeada por una sucesión de innovaciones que buscaron resolver una pregunta persistente: ¿cómo capturar y aprovechar el contexto lingüístico de forma eficiente? Inicialmente, las Redes Neuronales Recurrentes (*RNN*) ofrecieron una respuesta al procesar secuencias de palabras mediante un estado oculto que transmitía información entre pasos. Sin embargo, su naturaleza secuencial y la dificultad para retener relaciones a largo plazo (debido al desvanecimiento de gradientes) limitaban su utilidad en textos extensos o complejos, como narrativas o diálogos coherentes [41].

Este cuello de botella impulsó la adopción de las *LSTM* (*Long Short-Term Memory*), diseñadas para gestionar dependencias temporales mediante un sistema de puertas que regulaban la información relevante. Al decidir qué datos almacenar, descartar o transmitir en cada paso, las LSTM lograron generar párrafos con mayor cohesión, como respuestas en *chatbots* o traducciones preliminares [35]. No obstante, su procesamiento iterativo (paso a paso) y su alto costo computacional persistieron como barreras para escalar a tareas más ambiciosas [35, 78].

La ruptura llegó con los Transformers [49], que redefinieron el paradigma al reemplazar la secuencialidad por el mecanismo de autoatención. Al analizar todas las palabras de una secuencia simultáneamente (asignando pesos dinámicos según su relevancia contextual), los Transformers no solo aceleraron el entrenamiento, sino que capturaron relaciones sutiles entre términos distantes. Esto permitió, por ejemplo, generar resúmenes con cohesión temática o mantener la coherencia en historias largas, superando las limitaciones de sus predecesores.

La culminación de este trayecto son los Grandes Modelos de Lenguaje (*LLM* por sus siglas en inglés *Large Language Model*), como *GPT* o *LLaMA*, que escalaron la arquitectura Transformer a dimensiones sin precedentes [79]. Entrenados en corpus masivos y diversos, estos modelos no solo heredaron la capacidad de analizar contexto global, sino que desarrollaron una versatilidad única: pueden imitar estilos literarios, sintetizar información técnica o incluso simular razonamientos básicos, todo sin

reentrenamiento específico. Su secreto radica en el pre-entrenamiento no supervisado, donde internalizan patrones lingüísticos universales, y en su adaptabilidad mediante prompts que orientan su salida hacia tareas concretas [80].

1.5.1 Realización de tareas específicas a través de *prompts*

Los LLMs mencionados anteriormente pueden hacer más que simplemente generar cualquier texto, pueden llegar a realizar todo de tipo de tareas y su capacidad para realizar tareas complejas depende críticamente de los *prompts,* instrucciones en lenguaje natural que guían al modelo hacia un objetivo específico. Por ejemplo, un mismo modelo puede traducir textos, resumir artículos o simular diálogos, dependiendo de cómo se formule la entrada (*prompt*) [81, 82]. Un *prompt* puede ser una pregunta, una frase incompleta o una estructura detallada que condiciona la salida del modelo. En este contexto, la Ingeniería de *Prompts* (PE por sus siglas en inglés, *Prompt Engeneering*) emerge como una disciplina clave para diseñar, optimizar y adaptar estos estímulos, maximizando la relevancia y precisión de los resultados [82].

1.5.2 Estrategias de *prompting*

La PE se centra en estrategias para formular *prompts* que aprovechen las capacidades lingüísticas de los *LLMs*. A diferencia del *fine-tuning* (ajuste fino de modelos con datos específicos), la PE opera en el nivel de interacción con el modelo, sin modificar sus parámetros internos. Estas estrategias están constituidas por un conjunto de buenas prácticas y enfoques de instrucción para potenciar la capacidad del modelo de entender y replicar la acción específica que se desea que realice, entre las que se encuentran [82]:

- Aprendizaje de con pocos ejemplos (*Few-shot*): Incluir ejemplos en el *prompt* para guiar al modelo en tareas nuevas.
- Desambiguación: Redactar instrucciones que reduzcan interpretaciones múltiples.
- Contextualización: Incluir información de dominio o antecedentes.
- Persona o rol: Asignar roles al modelo (por ejemplo, "Actúa como un ingeniero de software").
- Razonamiento: Guiar al modelo a explicar su proceso (por ejemplo, "Piensa paso a paso").

- Plantillas estructuradas: Usar marcos predefinidos (por ejemplo, formato EARS para requisitos) para estandarizar salidas.

1.5.3 Aplicaciones en tareas de ingeniería de requisitos

Una de las muchas áreas que donde hay potencial de mejoras en los procesos con el uso de este tipo de generación de texto es la ingeniería de requisitos, tema principal de esta investigación [80, 83]. Donde estudios como [28] y [83] han mostrado la cantidad de investigaciones que se han hecho en tareas como la extracción, especificación y validación de requisitos obteniendo resultados prometedores pero que requieren de la validación de un experto en muchos casos. Su mayor utilidad se ve en la creación de borradores para el documento de especificación de requisitos que luego son validados y completados por expertos [27], como una primera etapa de validación de requisitos, en la búsqueda de requisitos que pudieron ser omitidos por el equipo de captación de requisitos, en la generación de historias de usuarios y casos de uso, entre otros. Incluso hay casos, cuanto menos curiosos, en los que ponen a los *LLMs* a simular un debate entre 2 analistas ficticios con diferentes perspectivas para asistir en la toma de decisiones sobre requisitos [82].

1.5.4 Evaluación de la generación

La evaluación de soluciones de generación de texto mediante *LLMs* especializadas para tareas concretas es un campo dinámico que se aleja de la evaluación general de *LLMs*. El enfoque se centra en la alineación con el objetivo de la tarea, la calidad intrínseca de la salida y la fidelidad al *prompt* y a las restricciones impuestas [80]. Esto implica medir qué tan bien la solución cumple su propósito específico, yendo más allá de la simple capacidad lingüística.

Una variante utilizada es el uso de métricas basadas en referencias. Si se dispone de salidas "*gold standard*", métricas como *BLEU*, *ROUGE* o *METEOR* miden la superposición léxica, aunque pueden no capturar bien la semántica [83]. Las salidas "*gold estandard*" no son más que ejemplos de respuestas correctas esperadas para una determinada entrada del modelo, estas métricas se basan en puntuar la similitud de la respuesta generada por la solución y la establecida como *gold standard*. Alternativamente, métricas basadas en *embeddings* como *BERTScore* o

Sentence-BERT similarity también ofrecen una buena evaluación de la similitud semántica con las referencias [83].

A pesar de la existencia de estas métricas, no siempre son empleadas por la falta de un set de datos que contenga una serie de ejemplos entrada/salida sobre la tarea concreta que pueda ser utilizada como *gold standard* (El caso de la investigación pertinente a este trabajo es uno de ellos). En este caso la evaluación humana es el sistema de evaluación fundamental, especialmente para tareas que requieren un juicio matizado, comprensión contextual profunda o la valoración de la utilidad real [84]. Los evaluadores humanos suelen calificar la salida según criterios como la fluidez, coherencia, relevancia, corrección factual, completitud, claridad y utilidad, utilizando a menudo escalas, comparaciones pareadas o análisis de errores. Por ejemplo, en la generación de Especificaciones de Requisitos de Software (*SRS*), los autores de [27] utilizaron evaluadores humanos (ingenieros de software) para calificar los documentos generados según criterios como la completitud, consistencia y claridad definidos por estándares como el de la norma IEEE 830.

Algunos de los elementos verificados por evaluadores humanos según el interés de la tarea son los siguientes [83-85]:

- **Corrección Funcional (*Functional Correctness*):**
    - Para generación de código: ¿El código compila? ¿Pasa los *tests* unitarios? (Métricas como *Pass@k*).
    - Para generación de respuestas a preguntas: ¿La respuesta es correcta según una base de conocimiento?
- **Cumplimiento de Restricciones (*Constraint Adherence*):**
    - ¿La salida sigue un formato específico (por ejemplo, *JSON*, *XML*, formato de *SRS*)?
    - ¿La longitud de la salida está dentro de los límites?
- **Verificación de Propiedades (*Property Checking*):**
    - Para *SRS*: ¿Es el requisito verificable? ¿Es consistente con otros? Se pueden desarrollar scripts o heurísticas para chequear algunas de estas propiedades.

- Para contenido generado: ¿Es seguro? ¿No es tóxico? (usando clasificadores de toxicidad).

1.5.5 Antecedentes de generación de requisitos de software

Recientemente se ha incrementado el uso de los *LLMs* como soporte a la IR en la generación de requisitos, documentación y especificaciones de estos, su priorización y validación [26-28, 83].

En [27] estudia el uso de *GPT-4* y *CodeLLama* para generar documentos de especificación de requisitos completos. A partir de una problemática descrita generar todos los elementos de un documento de requisitos (incluyendo entre ellos los requisitos funcionales y no funcionales), y emplean el caso de estudio de un portal web para el manejo de un club universitario. Le especifican al modelo como descripción del problema elementos como: roles implicados y sus responsabilidades en la aplicación.

En [26] generan requisitos de software a partir de palabras clave utilizando un *LLM*, mejorado con inyección de conocimiento de dominio y restricciones sintácticas (reducir alucinaciones). La inyección de conocimiento lo hacen con una ontología de dominio para obtener relaciones entre entidades, con lo que entrenan el modelo. Durante la generación, fuerzan la aparición de las palabras clave proporcionadas por el usuario en los requisitos finales.

Finalmente, en el caso de [28] se enfocan en el uso de los *LLMs* como herramienta auxiliar para identificar requisitos desconocidos o no pensados, y presentan ejemplos de *prompts* para estos fines, pero enfocado en las características de un usuario con características específicas.

1.6 Herramientas utilizadas

1.6.1 Lenguaje de programación: Python

Se eligió el lenguaje de programación Python porque es un lenguaje simple y cómodo de trabajar que cuenta con muchas bibliotecas y herramientas para realizar procesamiento lingüístico y tareas de aprendizaje [39, 41]. También es el lenguaje que predomina

abrumadoramente en las investigaciones relacionadas consultadas y la comunidad de personas, investigadores, documentación y materiales de aprendizaje es considerablemente superior a otros como Java o C++ en este campo. Python además tiene otras ventajas como [86]:

- Aparte de la gran cantidad de funcionalidades que permite su biblioteca estándar cuenta con una cantidad significativa de bibliotecas externas que facilitan enormemente casi cualquier tipo de problema informático: trabajo con bases de datos, desarrollo web, análisis de datos, entre otros.
- Python puede ser ejecutado en cualquier sistema operativo moderno, ya sea Windows, Linux/ UNIX, macOS, entre otros.
- Python tiene una licencia de tipo Open Source, que permite ser utilizado y distribuido libremente.

1.6.2 Bibliotecas principales

***Transformers*:** Biblioteca de *Hugging Face* que proporciona *APIs* y herramientas para descargar y entrenar fácilmente modelos pre-entrenados del estado del arte. Soporta la interoperabilidad entre *PyTorch*, *TensorFlow* y *JAX*. Esto proporciona la flexibilidad de utilizar un *framework* diferente en cada etapa de la vida de un modelo; entrenar un modelo en pocas líneas de código en un *framework*, y cargarlo para inferencia en otro [87].

***PyTorch*:** Un *framework* integral de *Machine Learning* que permite una experimentación rápida y flexible y una producción eficiente gracias a un *front-end* fácil de usar, una formación distribuida y un ecosistema de herramientas y bibliotecas. Algunas de las facilidades que ofrece son [88]:

- Cálculo de tensores con fuerte aceleración de GPU, lo que permite operaciones numéricas arbitrarias en estructuras multidimensionales.
- Diferenciación automática para la creación y entrenamiento de redes neuronales profundas.
- Cálculo de gráficos dinámicos, lo que permite a los usuarios cambiar el comportamiento de la red sobre la marcha, en lugar de esperar a que se ejecute

todo el código. Esto lo vuelve más recomendado para entornos de experimentación como este trabajo.

PyTorch fue seleccionada después de ser comprada con otras alternativas ya que para el trabajo con redes neuronales existen 3 bibliotecas o marcos de trabajo principales dentro del campo que facilitan su uso: *scikit-Learn*, *PyTorch* y *Tensorflow* (En el caso de esta última siempre se utiliza en conjunto con un módulo llamado *Keras* que ofrece muchas ventajas adicionales). En la tabla 7 se puede ver una comparación de algunos aspectos de estas bibliotecas.

Tabla 1.2: Comparación de las principales bibliotecas para redes neuronales

Aspecto	**ScikitLearn**	**Tensorflow+Keras**	**PyTorch**
Soporte para redes neuronales	Brinda opciones pero poco personalizables	Fuertemente enfocado en aprendizaje profundo y redes neuronales	Fuertemente enfocado en aprendizaje profundo y redes neuronales
Facilidad de uso	Muy alta	media	alta
Grafos de computación	Dinámicos	Estáticos	Dinámicos
Despliegue y producción	Opciones simples	Opciones robustas para casi cualquier plataforma	Opciones simples pero en crecimiento

Para el desarrollo de soluciones pensadas para producción el uso de *Tensorflow* ciertamente tiene una gran ventaja. sin embargo, en el campo de la investigación científica es más popular *PyTorch* debido al uso de grafos de computación dinámicos (*dynamic computation graph*) ya que permiten una mayor de flexibilidad y facilidad de *debugging*, ideal para la experimentación.

***Pytorch Lightning*:** El módulo de *Lightning* es una estructura organizativa para *PyTorch* que permite la máxima flexibilidad y un mínimo de complejidades. Actúa como una "receta" de modelo que especifica todos los detalles de la formación. Algunas de las ventajas que ofrece son [89]:

- Mejora la legibilidad del código al separar el código de investigación del código de ingeniería.

- Facilita la reproducción de modelos al automatizar gran parte del ciclo de entrenamiento y eliminar el código repetitivo.
- Ayuda a automatizar el proceso de optimización de los modelos, simplificando el entrenamiento y la experimentación.

***TorchMetrics*:** es una colección de más de 100 implementaciones de métricas para *PyTorch*, junto con una *API* fácil de usar para crear métricas personalizadas. Esta biblioteca aporta varias facilidades clave para los desarrolladores de modelos de aprendizaje automático, especialmente cuando se trabaja en conjunto con *PyTorch* y *PyTorch Lightning*. Permite [90]:

- Proporciona una interfaz estandarizada para aumentar la reproducibilidad de los modelos y las métricas.
- Las métricas están optimizadas para el entrenamiento distribuido.
- Las métricas pueden acumular resultados automáticamente a través de múltiples lotes, lo que es útil para calcular métricas a lo largo de todo el proceso de entrenamiento.

***Pandas*:** Biblioteca especializada en el manejo y análisis de estructuras de datos. Indispensable para el trabajo con grandes sets de datos. Es la más utilizada en su campo por sus grandes ventajas, entre las que se encuentran [91]:

- El objeto *DataFrame*, rápido y eficaz para la manipulación de datos con indexación integrada.
- Herramientas de lectura y escritura de datos entre estructuras de datos en memoria y distintos formatos.
- Unión y división de conjuntos de datos de alto rendimiento.

***Natural Language Tool Kit (NTLK)*:** Conjunto de bibliotecas para trabajar con el lenguaje natural, contiene sets de recursos útiles para el procesamiento y herramientas para tareas como la tokenización, lematización, entre otras [92].

***NumPy*:** Biblioteca fundamental para el trabajo avanzado con datos numéricos, provee entre otras herramientas estructuras muy versátiles para el trabajo con vectores multidimensionales [93].

***Scikit-learn*:** proporciona una colección de algoritmos para el aprendizaje automático y la minería de datos, incluye una amplia gama de algoritmos de clasificación, regresión, agrupamiento y reducción de dimensionalidad. Además, facilita el uso de métodos de validación como la validación cruzada [94]. Los algoritmos de agrupamiento utilizados en la investigación se utilizaron desde esta librería.

***Matplotlib*:** proporciona herramientas para la visualización de datos en forma gráfica de gran utilidad para la interpretación de los resultados [95].

1.7 Conclusiones parciales

- Las redes neuronales profundas, sobre todo de la arquitectura *Transformer*, han obtenido resultados del estado del arte en casi todas las tareas del procesamiento del lenguaje natural.
- El uso de modelos de lenguaje pre-entrenados de arquitectura *Transformer* permite aprovechar mediante la transferencia de aprendizaje su conocimiento general del lenguaje para reducir significativamente la cantidad de datos de entrenamiento requeridos y ajustarlos a tareas específicas como la clasificación.
- Existen varios enfoques para inyectar conocimiento en un modelo Transformer, siendo los principales estudiados: Inyección directa de texto, uso de vectores de características, modificación de los *embeddings*, recuperación de bases de datos, modelos guiados por reglas y el empleo de pre-entrenamiento adicional.
- El uso de algoritmos de agrupamiento en textos cortos representa un gran desafío por la limitada cantidad de información disponible y al utilizar *embbeddings* de alta dimensionalidad como los generados por modelos *Transformers* juega un papel fundamental la reducción de dicha dimensionalidad.
- Los grandes modelos de lenguaje (*LLM*) han demostrado una increíble capacidad para realizar tareas de generación específicas para las que no fueron entrenados mediante el uso de *prompts* (instrucciones detalladas que expliquen el proceso a realizar).

- En tareas de clasificación las métricas más utilizadas son *accuracy*. *precision*, *recall* y *F1-score*.
- En tareas de agrupamiento la métrica de calidad más utilizada cuando no se tiene un *dataset* con etiquetas a las que comparar es el coeficiente de *Silhouette.*
- En tareas de generación se evalúa la calidad de las sentencias generadas mediante un "*gold standard*" objetivo o en caso de que no se tenga mediante la evaluación de expertos o una combinación de ambos métodos.
- Ninguna de las soluciones de detección de opiniones informativas en la literatura que enfrentan este problema mediante modelos de lenguaje de arquitectura *Transformer* utilizan un enfoque para inyectar conocimiento del dominio en el modelo. Aunque se han enfrentado problemas similares de clasificación de opiniones utilizando conocimiento del dominio esto solo ha sido a través del método más costoso en términos de recursos y tiempo, el pre-entrenamiento adicional.
- Aunque existen investigaciones en la literatura sobre la generación de requisitos de software en ningún caso se ha probado la viabilidad de generarlos a partir de las opiniones de los usuarios para guiar el crecimiento del software.
- La biblioteca Transformers es imprescindible para el trabajo con esta arquitectura ya que permite cargar y utilizar los distintos modelos de lenguaje.
- Se destaca el uso de *PyTorch Lightning* en la compactación de los procesos de entrenamiento y evaluación de modelos manteniendo una gran flexibilidad.

Capítulo 2: Método de solución planteado

En este capítulo se expone todo lo referente al diseño e implementación de la solución propuesta, haciendo énfasis en cada una de sus etapas. Se presentan además artefactos de diseño como diagramas de flujo y casos de uso del sistema que permiten mostrar el funcionamiento de la solución.

2.1 Descripción de la solución propuesta

El método propuesto consta de tres fases: (1) identificación de opiniones “informativas” (OI) (o fase de Filtrado/Clasificación), (2) agrupamiento de opiniones afines, y (3) generación de requisitos. La calidad de los requisitos que se generen depende en gran medida de cuan informativa sean las opiniones que se utilicen en el proceso de generación. Por tanto, la primera fase consiste en un mecanismo de filtrado a partir de cual se detectan OI y eliminan las no informativas, aplicando para ello un clasificador basado en un modelo pre-entrenado *Transformer*. En una segunda fase se detectan las opiniones que tratan sobre los mismos aspectos, a partir de la construcción de clústeres de opiniones similares. Finalmente, se lleva a cabo la generación de requisitos en dos pasos utilizando un *LLM*, que primero crea requisitos por cada clúster y luego los unifica en una lista final para mitigar la redundancia. La figura 2.1 Ilustra este proceso.

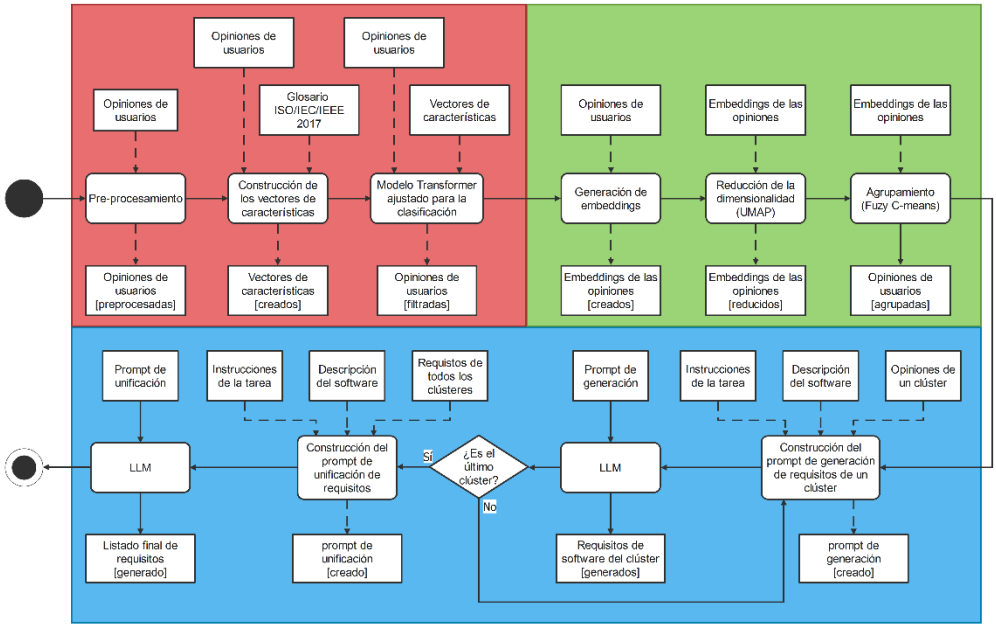


Figura 2.1: Esquema del flujo de trabajo de la solución

A continuación, se abordarán con detalle cada una de las etapas y procesos involucrados

2.2 Fase de clasificación

Las OI son aquellas cuyo contenido está relacionado con el dominio del software, y "no informativas" en otro caso. La identificación de esas OI se lleva a cabo a partir de un modelo de predicción con arquitectura *Transformer*, al cual se le aplica un ajuste fino (o *fine-tune*) para adaptarlo a la solución de este problema. Adicionalmente, se propone un mecanismo de inyección de conocimiento del dominio al modelo *Transformer* para que pondere en la detección de rasgos de las opiniones los tokens más relevantes para el dominio de software. Dicho conocimiento proviene de un vocabulario formalizado en la norma ISO/IEC/IEEE 24765:2017 [96], sobre el dominio del software y la ingeniería de software.

2.2.1 Pre-procesamiento

El comentario entrado al programa es tokenizado antes de ser introducido al modelo. Este proceso es llevado a cabo por un tokenizador propio de cada modelo que es descargado automáticamente de su correspondiente repositorio en *Hugging Face* [1]a través de la biblioteca Transformers. Dicho tokenizador puede ser configurado a través de sus parámetros para regular aspectos de su comportamiento, para la solución actual se decidió activar los siguientes:

- Añadir los tokens especiales: Son introducidos tokens que le indican al modelo el inicio del comentario (CLS) y las separaciones entre oraciones (SEP) para mejorar la capacidad de comprensión del modelo sobre la estructura de las oraciones.
- Tamaño límite: Se estableció un tamaño límite de entrada de 200 tokens, más que suficiente para cubrir la mayoría de comentarios en redes y tiendas de aplicaciones sobre programas y la totalidad de los comentarios utilizados en los sets de datos de entrenamiento y prueba.
- Activar *padding*: Son añadidos tokens especiales [32] para rellenar las entradas menores a la capacidad de entrada del modelo y garantizar la uniformidad en el

[1] https://huggingface.co

tamaño de los datos recibidos. Dichos tokens son ignorados durante el procesamiento del comentario.

- Activar truncación: Las entradas de tamaño mayor al permitido son recortadas en su final para acotarlas al tamaño de entrada establecido.

Al terminar este proceso el resultado son los tokens y la máscara de atención correspondientes con la información del comentario que es introducido al modelo para la siguiente etapa.

2.2.2 Construcción del vector de características

Durante la etapa de pre-procesamiento aparte del vector con el comentario tokenizado también se construye el vector de características del dominio asociado a la opinión. Este vector de características es el eje central de la inyección de conocimiento del dominio en el modelo y el que será utilizado por la técnica de conocimiento. El cómo se utiliza exactamente el vector y la técnica de inyección de conocimiento en sí se describen en mayor detalle en la sección 2.2.3.

Este vector de características es el eje central de la inyección de conocimiento del dominio en el *Transformer*, y su concepción constituye una contribución de este trabajo. En la construcción de este vector de características es donde se aplica el vocabulario de la norma *ISO/IEC/IEEE* 24765:2017.

En este trabajo se estudiaron 2 formas de construir el vector de características a partir del vocabulario: (1) basado en el conteo de los términos relevantes (vector *RC*, siglas de *Relevant Count*) y (2) la posición de los términos relevantes (vector *RP*, siglas de *Relevant Position*). El vector de características generado siempre es del mismo tamaño que la entrada máxima del modelo *Transformer*.

2.2.2.1 Vector RC

El vector *RC* se construye sobre la base de la cantidad de elementos del glosario que están presentes en la opinión. Se identifican tanto palabras aisladas como frases compuestas, donde los componentes individuales de las frases estén presentes en el glosario, reforzando su valor (ej. "*access*" y "*method*" en "*access method*"). El escalar

resultante se proyecta en un vector del tamaño requerido replicando el conteo en todas las dimensiones. Esta expansión dimensional mitiga el efecto de dilución en espacios de alta dimensionalidad, asegurando una contribución significativa en la capa de clasificación final. Esta estrategia introduce un sesgo inductivo que asocia la densidad terminológica específica con el grado de informatividad de la opinión. Este enfoque se sustenta en la hipótesis de que los comentarios técnicamente densos poseen mayor factor de novedad informativa sobre el dominio en cuestión, y de ahí mayor interés para la captura de requisitos por parte de los equipos de desarrollo y soporte de software. Un ejemplo más ilustrativo se puede observar en a continuación:

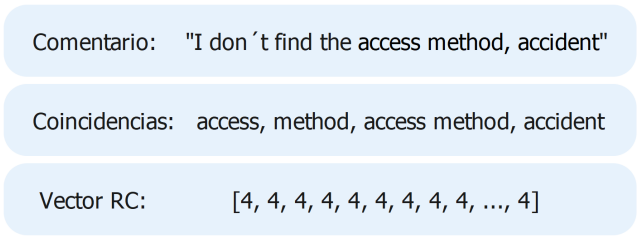


Figura 2.2: Ejemplo de vector RC

En el ejemplo de la figura 2.2 podemos ver que se encuentran 4 coincidencias en el comentario (*access*, *method*, *access method* y *accident*). Como sale a relucir en el glosario también hay términos compuestos o frases con "*access method*" siendo uno de muchos, en la mayoría de estos casos los términos independientes que componen la frase también se encuentran en el glosario por lo que encuentra coincidencias adicionales por cada término compuesto. Este conteo extra que fue originalmente un resultado accidental se mantuvo en la forma de hacer el conteo ya que tiene sentido premiar o puntuar más las formaciones complejas de palabras que son más específicas que términos sueltos a la hora de señalar que tan informativa es la opinión.

2.2.2.2 Vector RP

Otra forma para inicializar el vector de características del dominio de una opinión es en función de la posición de los términos relevantes que aparecen en esa opinión. En este trabajo se le refiere a un vector construido de esta forma como Posición de Relevantes o vector RP (por las siglas en inglés *Relevant Position*).

En este caso se crea un vector inicializado a 0 en todos sus valores de igual tamaño a la entrada máxima del modelo. Después utilizando como referencia las posiciones del vector de la opinión tokenizada introducida en el modelo se comprueban qué posiciones corresponden a términos que están presentes en el glosario de términos relevantes y en esas posiciones se les cambia el valor a 1. De esta forma el resultado es un mapeo del vector original de la opinión donde hay 1s en las posiciones relevantes según el glosario y 0 en las que no.

Retomando la opinión para este enfoque tenemos lo siguiente:

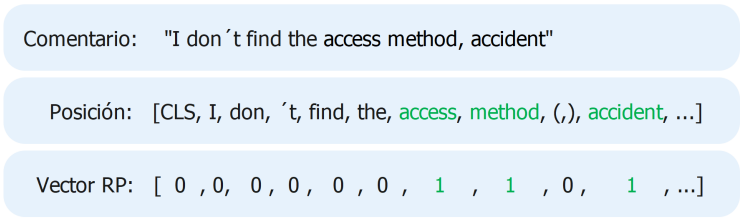


Figura 2.3: Ejemplo de vector RP

En el ejemplo de la figura 2.3 se encuentran las mismas coincidencias del enfoque anterior (*access*, *method* y *accident*) y en sus correspondientes posiciones del vector de características se le asigna el valor binario 1 indicando su relevancia como término presente en el glosario. En este caso la coincidencia compuesta de "*access method"* que fue contada doble en el enfoque por conteo no tiene objetivo aquí ya que sus términos independientes ya están incluidos. En caso de estuviera un término compuesto, pero no todos sus componentes fueran términos independientes del glosario se potenciarían igualmente todos los términos que forman parte del término compuesto.

2.2.3 Modelo de clasificación

La identificación de las OI se trata como un problema de clasificación binaria ("informativa" o "no informativa") mediante la aplicación de un modelo pre-entrenado con arquitectura *Transformer*. Específicamente fue utilizado el modelo *BERTweet*, dado que fue el que mejores resultados obtuvo en las experimentaciones realizadas. Como parte del ajuste fino de este modelo se especificó una única clase de salida en la capa final de clasificación, para la que se utilizó una capa de clasificación lineal, con una función de

activación *sigmoid*, para comprimir la salida en el rango de 0 a 1, similar a lo reportado en [64], interpretándose como la probabilidad de pertenecer a la clase “informativa”.

El uso del conocimiento del dominio, representado en el vector de características obtenido en el pre-procesamiento, se incorpora al modelo de predicción antes de la capa de clasificación. El enfoque para hacer uso de este vector está basado en la solución explicada en la sección 1.4.2.2 donde es fusionados junto con la salida del modelo pre-entrenado para dotar a esta salida de dicho conocimiento como un extra útil. La fusión de ambos vectores no es más que su concatenación, añadiendo los valores del vector de características al final del vector de salida del modelo, obteniendo un vector final de tamaño igual a la suma de los tamaños de ambos. Este vector resultando con la información de dominio agregada es entonces introducido en la capa linear de clasificación final Una representación de este proceso se muestra en la figura 2.4.

La decisión final es tomada con un simple umbral de decisión que clasifica la opinión original como “informativa” si la salida del modelo es un valor mayor a 0.5, lo cual indica que el comentario es suficientemente relevante para clasificarlo como tal, en caso contrario su relevancia es insuficiente y se clasifica como “no informativa”. Las opiniones clasificadas como “no informativas” son descartadas.

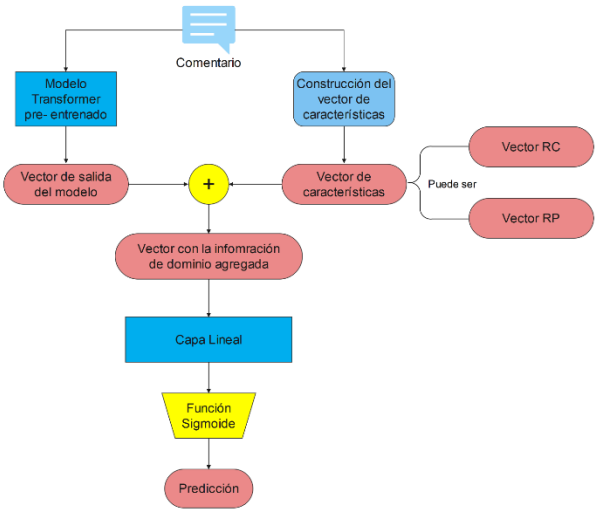


Figura 2.4: Arquitectura del modelo de clasificación con conocimiento

2.3 Fase de agrupamiento

Luego de identificar las opiniones que pueden tener contenidos valiosos para los equipos de desarrollo y la formulación de nuevos requisitos, se hace necesario determinar cuáles son los aspectos fundamentales en los que los usuarios han enfocado sus valoraciones. En la etapa anterior se logra eliminar información ruidosa y que no aporta información relevante en cuanto a requisitos, pero puede quedar mucha mezcla de información en el conjunto de OI, que puede afectar la eficacia en la generación de requisitos a partir del *LLM*. En este sentido, se decidió abordar esta problemática aplicando un algoritmo de agrupamiento que permita detectar las opiniones afines, las cuales constituirán la guía en el trabajo con el *LLM*.

El agrupamiento de las opiniones se llevó a cabo aplicando el algoritmo *Fuzzy C-means,* por su capacidad para manejar pertenencias parciales a múltiples clústeres, necesario para mitigar el efecto de la ambigüedad en el conjunto de opiniones. Además, estas pertenencias parciales permiten la asignación de una misma opinión a múltiples clústeres a partir de un umbral de pertenencia, esto era de interés para el proceso posterior de generación ya que a pesar de que comentarios que hablen de varios temas pueden estar incluidos en los clústeres de cada tema y garantizar que el *LLM* cuente con la mayor cantidad de información disponible referente al clúster que está analizando en ese momento. Se utilizó un umbral de pertenencia de 0.3 de forma que cada comentario sería incluido en todos los clústeres con los que tenga ese grado de pertenencia o mayor. En el caso de que un comentario no cumpla con ese umbral de pertenencia para ningún clúster es añadido por defecto al clúster con el que tenga mayor grado de pertenencia, garantizando que no se omitan comentarios aislados y sean procesados por el *LLM* en algún momento.

Este algoritmo se aplica sobre los *embeddings* contextuales que se obtienen de cada opinión utilizando el modelo *BERTweet-base*, y usando la distancia euclidiana como medida de similitud, que en comparación con la función coseno mostró un mejor comportamiento. A la representación de los *embeddings* (vectores de 768 dimensiones), se les aplica la técnica UMAP (*Uniform Manifold Approximation and Projection*) para reducir el tamaño de los vectores a 2 dimensiones, dado que se comprobó que aportaba

mayor calidad al resultado del agrupamiento, entre varias dimensiones evaluadas (2, 10, 50 y 100). Esta decisión estuvo respaldada con evaluaciones sobre distintos métodos de reducción usando el coeficiente de *Silhouette* [75] como medida de calidad del agrupamiento. Estas mediciones se complementaron con la validación de la coherencia temática del clúster mediante los 10 términos más representativos, obtenidos a través de la construcción de un vector característico *TF-IDF* (*Term Frequency-Inverse Document Frequency*) de cada clúster.

Los resultados mostraron que el valor de K (cantidad de clústeres) es muy dependiente del conjunto de datos por lo que como parte del proceso de agrupamiento primero se decide qué valor de K utilizar a partir de evaluaciones sobre un rango de valores utilizando el coeficiente de *Silhouette* y se toma la división que obtuvo el mayor valor de esta métrica

2.4 Fase de Generación

Los requisitos son generados de forma automática usando un *LLM* y los clústeres temáticos identificados en la fase anterior como contexto. El empleo de los clústeres (opiniones que tratan sobre los mismos aspectos), permite guiar mejor al *LLM* en el proceso de generación, y reducir el riesgo de alucinaciones al centrarse únicamente en opiniones sobre un mismo tema de la aplicación en cada clúster. En la concepción de esta solución se utilizó como *LLM DeepSeek* [97] por su reciente impacto y capacidades al nivel del estado del arte actual mientras se mantiene como opción gratuita y de código abierto disponible para desplegarse de forma local por cualquier persona con las condiciones de *hardware* suficientes para ello. Este, aunque se encuentra disponible de forma gratuita en el mismo repositorio de hugging face[2] debido a su enorme tamaño y demanda no se pudo probar local en el limitado hardware con el que se contaba para la evaluación de esta investigación, por ello el modelo se utilizó a través de API de Inference

[2] https://huggingface.co/deepseek-ai/DeepSeek-V3

providers[3] proporcionada por la misma plataforma como alternativa para usar limitadamente determinados modelos en sus servidores con fines no comerciales.

Estos requisitos generados tienen un problema, como son generados de forma independiente por cada clúster es inevitable que haya requisitos muy similares generados a partir de diferentes clústeres, por lo que existen un importante nivel de redundancia en este resultado. Por ello, se decidió llevar a cabo un segundo procesamiento con el LLM, la unificación de los requisitos generados. Este proceso busca resumir los resultados, el LLM identifica requisitos muy similares y los funde en un mismo requisito a la hora de dar su reporte final.

2.4.1 Estructura de los *prompts* utilizados

Para las fases de generación y unificación de requisitos, no se utilizaron preguntas simples, sino *prompts* de ingeniería avanzada, diseñados meticulosamente siguiendo un conjunto de buenas prácticas establecidas en la literatura para maximizar la fiabilidad, precisión y consistencia de los resultados. Ambos incorporan 2 elementos dinámicos: Descripción de la aplicación (descripción breve de la funcionalidad actual del software objeto de estudio) y la fuente de información: un clúster de opiniones (conjunto de opiniones que tratan sobre los mismos aspectos del software) en el caso de la generación inicial y los requisitos generados de todos los clústeres en el caso de la unificación.

Una de las técnicas más efectivas para contextualizar al modelo es asignarle un rol o "persona". Al instruir al *LLM* con frases como "*Act as a Senior Requirements Engineer*" (*prompt* de generación) o "*Act as a Lead Systems Analyst*" (*prompt* de unificación), se enfoca la respuesta del modelo, mejorando la calidad y el estilo del texto generado para que se alinee con las expectativas de un experto en el dominio. Esta práctica es muy común en otras tareas de soporte a la ingeniería de requisitos [82].

La complejidad de una tarea puede abrumar a un LLM si se le presenta de forma ambigua. La estrategia seguida fue descomponer las tareas complejas de "generar requisitos" y "unificar requisitos" en una serie de reglas y pasos lógicos explícitos. Investigaciones como [98] han demostrado que guiar al modelo a través de pasos

[3] https://huggingface.co/docs/inference-providers/index

intermedios de razonamiento mejora drásticamente su rendimiento en tareas complejas. Esto se materializa en las secciones "*CRITICAL RULES FOR REQUIREMENTS*" (generación) y "*CORE LOGIC FOR CONSOLIDATION*" (unificación). En lugar de pedir genéricamente que "unifique", se le proporciona un algoritmo a seguir: "1. *Identify Similar Requirements*", "2. *Execute the Merge*", "3. *Preserve Unique Requirements*", etc. Esta técnica, a veces denominada *algorithmic prompting*, convierte una tarea creativa en un proceso más determinista, reduciendo la variabilidad y la probabilidad de error.

Para mejorar aún más la fiabilidad, se introdujo una instrucción explícita para que el modelo revise y refine su propio trabajo antes de producir la salida final. En el *prompt* de generación, la regla 5 (*Self-Correction and Refinement*) obliga al modelo a realizar una pasada de consolidación y re-verificación. Esta es una aplicación práctica del patrón de reflexión (*reflection pattern*) [99], donde se ha demostrado que los *LLMs* pueden mejorar sus resultados si se les incita a criticar y corregir sus borradores iniciales, simulando un proceso de revisión humano.

Aunque los *prompts* no realizan un ajuste fino del modelo, sí le proporcionan ejemplos concretos de la tarea a realizar y, sobre todo, del formato de salida esperado. Las secciones "EXAMPLES OF REQUERIMENTS GENERATION" (generación) y "EXAMPLE OF CONSOLIDATION" (unificación) son una implementación de *few-shot learning* [100], donde se le muestran al modelo uno o más ejemplos completos del par entrada-salida deseado.

Para asegurar la consistencia, ambos *prompts* utilizan un formato de entrada y salida estructurado con etiquetas claras ("*ROLE*", "*PRIMARY TASK*", "*INPUT DATA*", etc.). Esta práctica minimiza la ambigüedad y ayuda al modelo a interpretar correctamente las distintas partes de la instrucción [82]. Exigir un formato de salida igualmente estructurado (ej. "***FR[NNN]:** ... (Based on comments: [...])*") es fundamental, ya que produce un texto que puede ser fácilmente procesado por código.

Para ver la definición exacta de la plantilla de cada *prompt* utilizado remítase a los anexos I (generación) y II (unificación).

2.5 Desarrollo del generador de requisitos de software a partir de opiniones de usuarios

En este epígrafe se describen los elementos de ingeniería de software tomados en cuenta en el diseño e implementación de la solución. Entre estos elementos se encuentra el modelo de dominio, el diagrama de casos de uso del sistema y la descripción detallada de los casos de uso.

2.5.1 Modelo de dominio

El modelo de dominio proporciona una visión estructural, ya que facilita representar el vocabulario y los conceptos claves del problema que se está modelando, así como las relaciones entre ellos. Dado que estos modelos son breves y bien estructurados facilitan capturar los conceptos de manera muy completa. Para el caso de esta investigación no existe un modelo de negocio asociado a la solución por lo que resulta más representativo modelar la información haciendo uso de un modelo de dominio como el mostrado en la figura 2.7 y que se describe a continuación:

El punto de partida del proceso es la **Opinión/Comentario**, que contiene el texto crudo proporcionado por un usuario. Primero, los comentarios pasan por una fase de **Preprocesamiento** que los **transforma** en un **Set de datos** limpio y estructurado. Para filtrar los comentarios y quedarse solo con los que son útiles, se realiza una **Clasificación**. Esta tarea es llevada a cabo por un **Modelo pre-entrenado**. El diagrama muestra que este modelo es una especialización de un ***Transformer***, que a su vez es un tipo de **Red neuronal**. Para procesar el texto, el modelo **utiliza** un **Tokenizador**, que adapta el lenguaje humano a un formato que el modelo puede entender.

Este **Set de datos** es analizado por un **Algoritmo de agrupamiento** (*Fuzzy C-means*), que tiene la función de **caracterizar** los datos agrupándolos. El resultado de este algoritmo es la creación de uno o más clústeres. Cada **Clúster** agrupa a un conjunto de opiniones que tratan sobre un mismo tema o idea. La relación **pertenece** indica que cada comentario útil es asignado a un clúster específico.

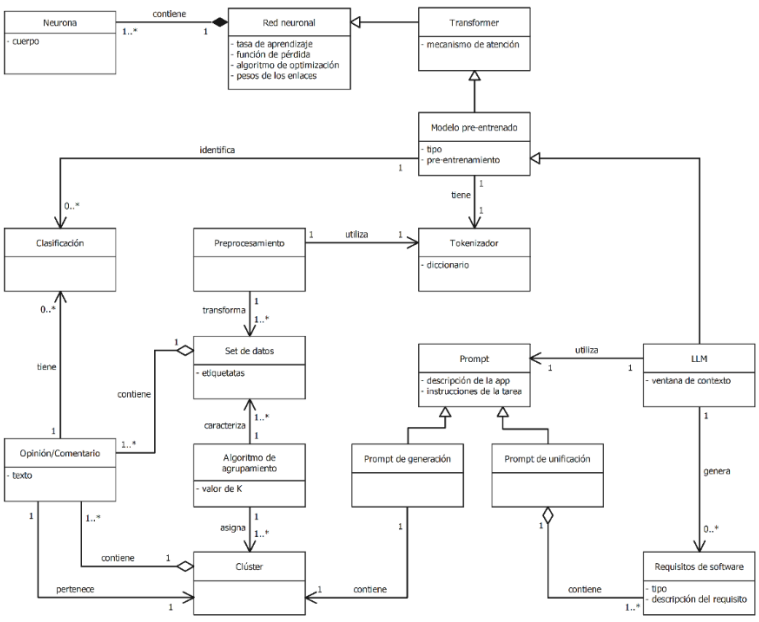


Figura 2.5 Modelo del dominio de la solución

El proceso de generación de requisitos se guía mediante ***Prompts***, que son instrucciones detalladas para el ***LLM***. El diagrama muestra que existen dos tipos de *prompts* especializados:

- ***Prompt* de generación:** por cada **Clúster** se crea un *prompt* de este tipo. Su propósito es instruir al **LLM** para que analice los comentarios de ese clúster y **genere** uno o más **Requisitos de software** específicos para el tema del grupo.
- ***Prompt* de unificación:** Después de que se han generado los requisitos para todos los clústeres, este *prompt* se encarga de una tarea de refinamiento. **Contiene** el conjunto de todos los requisitos generados previamente y le pide al ***LLM*** que los analice para unificarlos, eliminar duplicados y mejorar su redacción.

2.5.2 Captura de requisitos

El sistema fue desarrollado teniendo en cuenta los siguientes requisitos:

RF01: El sistema debe permitir la carga de un conjunto de opiniones de usuarios desde un archivo en formato CSV.

RF02: El sistema debe procesar las opiniones cargadas a través de un modelo de clasificación pre-entrenado para etiquetarlas como "informativas" o "no informativas".

RF03: El sistema debe tomar únicamente las opiniones clasificadas como "informativas", aplicarles un algoritmo de agrupamiento y asignar cada una a un clúster específico.

RF04: El sistema debe permitir la definición de una plantilla de "*prompt* de generación" que será utilizada para instruir al *LLM* en la generación de los requisitos.

RF05: El sistema debe iterar sobre cada clúster y enviar su contenido (las opiniones agrupadas) junto con el *prompt* de generación a un *LLM*.

RF06: El sistema debe almacenar los requisitos de software generados por el *LLM* para cada clúster de forma separada.

RF07: El sistema debe permitir la definición de una plantilla de "*prompt* de unificación" para la fusión de los requisitos de cada clúster en una lista consolidada final de requisitos.

RF08: El sistema debe juntar en una lista todos los requisitos generados inicialmente y enviarlos al LLM junto con el *prompt* de unificación.

RF09: El sistema debe mostrar la lista final de requisitos de software generados.

RF10: El sistema debe permitir la exportación del listado final de requisitos a un archivo de texto.

RNF01: Para cada requisito final generado, el sistema debe identificar en qué comentario/s está sustentado.

RNF02: Cada requisito final generado debe cumplir con el formato estándar de un requisito de “El sistema debe (acción) (condiciones)”.

2.5.3 Arquitectura de la solución

La arquitectura de software es el pilar sobre el que se construye la funcionalidad, la escalabilidad y la mantenibilidad de cualquier sistema complejo y para ellos se definen patrones comprobados de arquitectura y diseño que orienten el desarrollo de la solución, algunos de los principales tomados en cuenta se exponen a continuación.

2.5.3.1 Patrón N-Capas

Para este proyecto, se ha diseñado una arquitectura robusta y modular, orientada a la reutilización de componentes y a la clara separación de responsabilidades. Dicha estructura se representa visualmente en la Figura 2.8 mediante un diagrama de paquetes *UML*, el cual ilustra una arquitectura de N-Capas. Este enfoque por capas permite

organizar el sistema en grupos lógicos horizontales, donde cada capa tiene un rol específico. Se adopta un enfoque de reutilización, para priorizar que cada componente de la capa general pueda ser reutilizado en otras investigaciones y contribuir al desarrollo de nuevas soluciones.

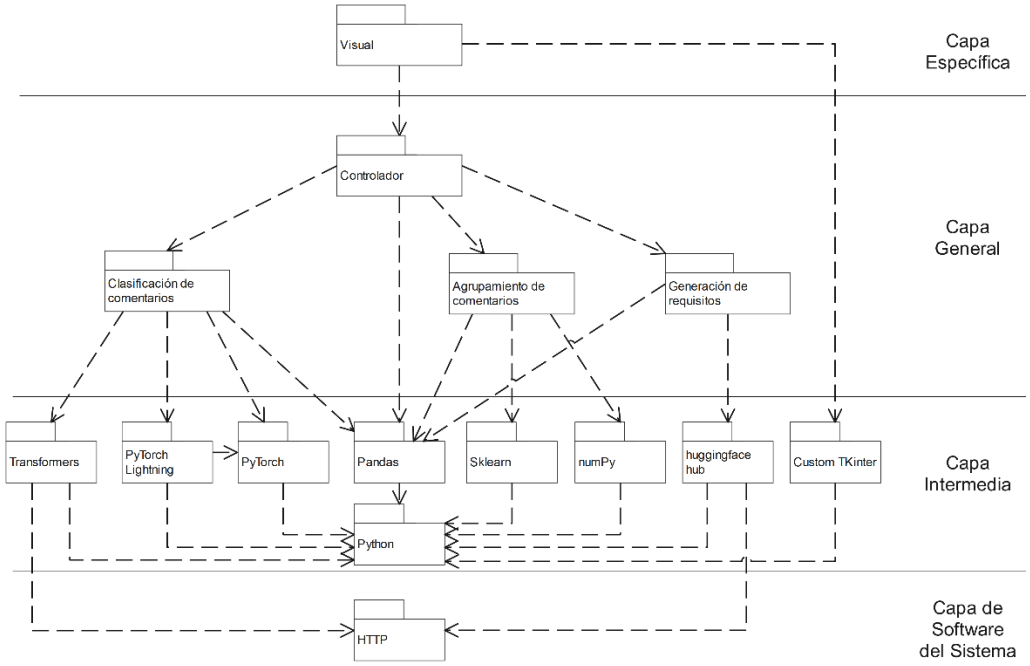


Figura 2.6: Diagrama división en n-capas (reutilización)

A continuación, se describen las capas funcionales del sistema, desde la más externa hasta la más interna, haciendo referencia a los componentes del diagrama.

Capa Específica: Esta es la capa más externa y se encarga de la interacción directa con el usuario final.

- Visual: Este paquete contiene todos los componentes visuales y de interacción. Su única responsabilidad es presentar los datos al usuario y capturar sus entradas (como el directorio de las opiniones y los parámetros del método).

Capa General: Considerada el cerebro de la aplicación, esta capa orquesta el flujo de trabajo completo para la generación de requisitos de software.

- Controlador: Actúa como el orquestador central del proceso. Recibe las peticiones de la Interfaz y coordina las llamadas a los diferentes módulos de procesamiento

en la secuencia correcta: clasificación, agrupamiento y generación. Centralizar el flujo en este componente desacopla los módulos de procesamiento entre sí.

- Clasificación de comentarios: Primer paso del proceso. Su función es filtrar los comentarios, descartando aquellos que no son relevantes para la generación de requisitos.
- Agrupamiento de comentarios: Este paquete agrupa los comentarios en clústeres temáticos utilizando algoritmos de aprendizaje no supervisado.
- Generación de requisitos: Toma cada clúster de comentarios y, mediante la interacción con un Modelo de Lenguaje Grande (*LLM*) externo, genera una lista de requisitos de software preliminares para cada grupo. Posteriormente, realiza un segundo paso de unificación para consolidar la lista final.

Capa Intermedia: Esta capa está compuesta por librerías y *frameworks* de terceros que proporcionan la funcionalidad fundamental sobre la que se construye la Capa General.

Capa de Software del Sistema: Es la capa más fundamental, representando el entorno de ejecución y los protocolos de comunicación. En este caso el único protocolo de comunicación utilizado es el *HTTP* para la descarga de modelos de la biblioteca Transformer y para la comunicación con la API de *Hugging Face* en *Huggingface hub.*

2.5.3.2 Patrón Filtros y Tuberías

Si bien la arquitectura en N-Capas define la estructura estática y la organización de los componentes del sistema, para modelar el flujo de procesamiento de los datos se adopta un estilo arquitectónico complementario: Filtros y Tuberías. Este patrón, ilustrado en la Figura 2.9, es ideal para sistemas cuyo propósito principal es transformar un flujo de datos a través de una serie de pasos computacionales secuenciales.

Figura 2.7: Diagrama del patrón filtros y tuberías

En el contexto de este proyecto, el patrón se manifiesta de la siguiente manera:

Filtros (Filters): Son los componentes de procesamiento independientes, representados por las cajas en el diagrama. Cada filtro realiza una única transformación específica sobre los datos que recibe. En este caso los filtros son:

- Leer comentarios: Actúa como la fuente de datos, iniciando el flujo.
- Preprocesar comentarios: Limpia y prepara el texto.
- Filtrar comentarios: Corresponde a la clasificación, eliminando opiniones no informativas
- Agrupar comentarios: Realiza el clustering temático.
- Generar requisitos preliminares: Primer uso del *LLM* para la generación inicial.
- Unificar requisitos: Segundo uso del *LLM* para eliminar consolidar los requisitos.
- Emitir resultados: Actúa como el sumidero, entregando el resultado final.

Tuberías (Pipes): Son los conectores que mueven los datos desde la salida de un filtro hasta la entrada del siguiente, representados por las flechas. En la práctica, una tubería puede ser tan simple como una llamada a una función que pasa un objeto

2.5.3.3 Patrón Mediador

El paquete Controlador implementa este patrón. En lugar de que los módulos de procesamiento (Clasificación, Agrupamiento, etc.) se comuniquen directamente entre sí, creando una red de dependencias compleja, el Controlador centraliza el flujo de control. Esto reduce drásticamente el acoplamiento entre los módulos, haciéndolos más independientes, fáciles de probar y reutilizables. Su función trasciende la simple gestión de la lógica de negocio; actúa como el puente explícito que conecta la organización estructural de la arquitectura en N-Capas con el flujo dinámico del patrón de Tuberías y Filtros. Su responsabilidad es orquestar la secuencia de operaciones definida en el patrón de Tuberías y Filtros. En lugar de que los filtros se conozcan y dependan entre sí, es el mediador quien se encarga de invocar cada filtro y gestionar el flujo de datos.

2.5.3.4 Patrón *Singleton*

Para asegurar una gestión centralizada y unívoca del estado y del flujo de trabajo a lo largo de toda la aplicación, la implementación del paquete Controlador se adhiere al patrón de diseño *Singleton* como se muestra en la figura 2.10. Este patrón garantiza que exista una y solo una instancia del Controlador durante todo el ciclo de vida de la

aplicación, lo que previene la posibilidad de estados múltiples y conflictivos, asegurando que todas las operaciones se realicen sobre un conjunto de datos consistente. Además, proporciona un punto de acceso global y bien definido para la Interfaz, sin necesidad de una compleja inyección de dependencias.

2.5.4 Casos de uso del sistema

Los diagramas de casos de uso permiten visualizar de manera clara y concisa las funcionalidades del sistema, facilitando la identificación de requisitos, la detección de errores y la planificación de la implementación [1]. En la figura 2.4 se muestra el diagrama de casos de uso diseñado como parte de la solución propuesta.

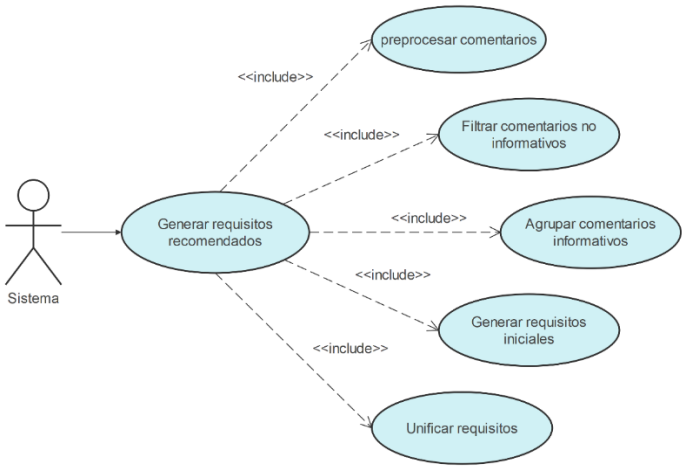


Figura 2.8: Diagrama de casos de uso del sistema

2.5.3.1 Descripción de los casos de uso

Los diagramas de casos de uso brindan una información general y bastante sencilla sobre las interacciones entre funcionalidades, por ello es necesario abordar con más detalle cada una. Un ejemplo de esto sería una descripción textual de algunos aspectos claves estructurados en formato tabular o como un diagrama de secuencia [1]. Dichas descripciones se presentan a continuación:

Tabla 2.1: Descripción del caso de uso Generar requisitos recomendados

Caso de uso	Generar requisitos recomendados
Actores	Sistema

Descripción	El caso de uso comienza cuando se carga en el sistema una colección de comentarios. Luego estos datos son pre-procesados y se filtran los comentarios no informativos. Después los comentarios resultantes se agrupan en temas afines y se utilizan como materia prima para generar nuevos requisitos de software por cada grupo obtenido. Finalmente, se unifican los requisitos de todos los clústeres en una sola lista. El caso de uso termina cuando se obtiene la lista final de requisitos unificados.
Casos de uso asociados	• Preprocesar comentarios <Include> • Filtrar comentarios no informativos <Include> • Agrupar comentarios informativos <Include> • Generar requisitos iniciales <Include> • Unificar requisitos <Include>
Precondiciones	Debe existir al menos una colección de opiniones sobre la que generar los requisitos.
Postcondiciones	Debe obtener la lista final de requisitos recomendados.

Tabla 2.2: Descripción del caso de uso Preprocesar comentarios

Caso de uso	Preprocesar comentarios
Actores	Sistema
Descripción	El caso de uso comienza cuando se han obtenido todos los comentarios a procesar. Los comentarios son procesados por tokenizador del modelo que los convierte en representaciones vectoriales digeribles por un ordenador además de emparejar sus tamaños. El caso de uso termina cuando todos los comentarios han sido tokenizados.
Casos de uso asociados	-
Precondiciones	Debe existir al menos una colección de comentarios que preprocesar.

Postcondiciones	Debe obtener una lista de comentarios preprocesados.

Tabla 2.3: Descripción del caso de uso Filtrar comentarios no informativos

Caso de Uso	Filtrar comentarios no informativos
Actores	Sistema
Descripción	El caso de uso comienza cuando los comentarios preprocesados son suministrados al modelo clasificador. La red neuronal del modelo procesa las entradas y devuelve el grado de relevancia de cada comentario para luego decidir por un umbral de 0.5 si es informativo o no. Los comentarios no informativos son descartados. El caso de uso termina cuando todos los comentarios han sido analizados.
Casos de uso asociados	-
Precondiciones	Debe una lista de comentarios tokenizados y de tamaño uniforme.
Postcondiciones	Debe obtener una lista de solamente comentarios informativos.

Tabla 2.4: Descripción del caso de uso Agrupar comentarios informativos

Caso de Uso	Agrupar comentarios informativos
Actores	Sistema
Descripción	El caso de uso comienza cuando los comentarios informativos son preparados para el agrupamiento. Los comentarios son representados en *embeddings*, se reduce su dimensionalidad, se determina el valor de clústeres óptimo a formar mediante la evaluación de un rango de valores de K y finalmente son agrupados utilizando el algoritmo *Fuzzy C-means*. El caso de uso termina cuando los comentarios han sido agrupados.
Casos de uso asociados	-

Precondiciones	Debe recibir una lista de comentarios informativos
Postcondiciones	Debe obtener una lista de comentarios informativos etiquetados con el clúster al que fueron asignados.

Tabla 2.5: Descripción del caso de uso Generar requisitos iniciales

Caso de Uso	Generar requisitos iniciales
Actores	Sistema
Descripción	El caso de uso comienza cuando se obtiene la asignación por clúster de los comentarios informativos. Se generan los requisitos correspondientes a cada clúster de comentarios utilizando un *LLM*. Para cada clúster se crea *prompt* que contiene las instrucciones de la tarea, una descripción de la aplicación objetivo y la lista de comentarios pertenecientes a ese clúster. Cada uno de los *prompts* construidos son introducidos de manera individual al *LLM* para obtener los requisitos correspondientes a cada uno. El caso de uso termina cuando se han generado los requisitos de todos los clústeres.
Casos de uso asociados	-
Precondiciones	Debe recibir una lista de comentarios informativos etiquetados con el clúster al que pertenecen.
Postcondiciones	Debe obtener una lista de requisitos por cada clúster de comentarios.

Tabla 2.6: Descripción del caso de uso Unificar requisitos

Caso de Uso	Unificar requisitos
Actores	Sistema

Descripción	El caso de uso comienza al recibir los requisitos correspondientes a cada clúster de comentarios. Se unifican las listas de requisitos utilizando un *LLM*. Para ello se construye un *prompt* que explique la tarea de unificar todas las listas de requisitos buscando requisitos similares y fusionándolos. El *prompt* también especifica una descripción de la aplicación y una lista con todos los requisitos generados. El caso de uso termina cuando el *LLM* genera la lista unificada de requisitos.
Casos de uso asociados	-
Precondiciones	Debe recibir una lista de requisitos por cada clúster de comentarios.
Postcondiciones	Debe obtener una lista unificada de requisitos.

2.6 Conclusiones parciales

- Fue propuesta una solución para la generación de requisitos de software a partir de opiniones de usuarios basada en el procesamiento del lenguaje natural, el aprendizaje profundo, el uso de modelos de lenguaje de arquitectura Transformer, la inyección de conocimiento del dominio y el uso de grandes modelos de lenguaje
- La solución propuesta consta de 3 etapas fundamentales: el filtrado de las opiniones no informativas, el agrupamiento de las opiniones informativas en temas afines y la generación de los requisitos recomendados a partir de los distintos grupos de opiniones identificados.
- El método de clasificación propuesto incorpora conocimiento del dominio a través de la construcción de vectores de características basados en un glosario de la norma ISO/IEC/IEEE 24765:2017.
- Se propusieron 2 formas diferentes de crear un vector de características para introducir información de dominio en el modelo: el vector RC y el vector RP.
- Se definieron 2 *prompts* con instrucciones detalladas para las tareas de generación de requisitos y unificación de requisitos mediante un LLM.

- Fueron identificados 6 casos de uso del sistema que corresponden a las funcionalidades de generar requisitos recomendados, pre-procesar comentarios, filtrar comentarios no informativos, agrupar comentarios informativos, generar requisitos iniciales y unificar requisitos.

Capítulo 3: Validación y análisis de los resultados

En este capítulo se describe el estudio experimental llevado a cabo como forma de validación de la solución propuesta. Se presenta una descripción detallada de los datos, las métricas y el esquema de evaluación utilizado para cada etapa del proceso. Así como la comparación con otros trabajos reportados en la literatura que hayan utilizado el mismo conjunto de datos en el caso de la clasificación.

3.1 Marco de evaluación

La solución propuesta fue evaluada en la tarea de generación de requisitos de software a partir de opiniones de usuarios utilizando 4 *datasets* en inglés: *Facebook*, *Swiftkey*, *Tapfish* y *Templerun2*. Las etapas de clasificación y agrupamiento fueron evaluadas de manera individual buscando la mejor variante de solución en cada caso para los sets de datos evaluados. Para la etapa final de generación se analiza un caso de estudio a partir de los resultados de la mejor combinación de clasificación y agrupamiento sobre uno de los *datasets* mencionados.

Para realizar la evaluación de la etapa de clasificación se seleccionaron 5 modelos pre-entrenados: *RoBERTa*, *GPT-2*, *XLNet*, *ALBERT* y *BERTweet*, por sus características ya discutidas en el epígrafe 1.3.1.1, todos en su tamaño base. Los modelos fueron entrenados utilizando datos en el idioma inglés. Para cada modelo se calcularon las métricas de clasificación *accuracy*, *precision*, *recall* y *f1-score* usando la técnica de validación cruzada con 5 pliegues (*5-fold cross validation*). Para controlar la aleatoriedad de las divisiones de los pliegues se utilizó como semilla 123. Los modelos fueron entrenados con un tamaño de lote igual a 16. Esta elección se tomó teniendo en cuenta que al entrenar con tamaños muy grandes de lotes aumenta el riesgo de que ocurra un sobreajuste a los datos den entrenamiento y los modelos no generalicen lo suficientemente bien al predecir a partir de nuevos datos. La obtención de los modelos se hace del repositorio de *Hugging face* ya mencionada anteriormente, que cuenta con la mayoría de modelos de arquitectura Transformer de código abierto que existen.

Además, los modelos fueron entrenados durante 2 épocas, la cantidad utilizada en la investigación similar [21]. Se hizo una prueba aislada con 4 épocas, pero no hubo diferencias significativas así que se mantuvo con 2 durante la validación cruzada debido a que las limitaciones de hardware hacían necesario no duplicar el tiempo de validación.

Posteriormente se seleccionaron los 2 modelos con mejores resultados para ser evaluados utilizando 2 variantes de introducción de conocimiento de dominio en cada uno. Estas variantes son 2 formas diferentes de construir el vector de características del dominio: vector RC (basado en el conteo de términos relevantes) y vector RP (basado en las posiciones de los términos relevantes). Para más información sobre ellos refiérase a su sección correspondiente del capítulo 2 (2.2.2). Esto da un total de 9 soluciones de clasificación evaluadas (5 modelos base más 4 con conocimiento del dominio).

En el caso del agrupamiento dado que se seleccionó el algoritmo *fuzzy C-means* por su capacidad de asignación multi-clúster y el tratamiento de la pertenencia difusa, la evaluación se centró en los diferentes parámetros para la reducción de la dimensionaldad de los datos iniciales y su impacto en el agrupamiento. Específicamente se evaluaron 3 alternativas de reducción: *PCA* (*Principal Component Anlalysis*), *UMAP* (*Uniform Manifold Approximation and Projection*) y la posibilidad de directamente no reducir la dimensionalidad. Para cada variante de reducción (*PCA* y *UMAP*) se evaluaron además diferentes cantidades de componentes a las que reducir (la dimensionalidad objetivo resultante después de aplicar la reducción): 2, 10, 50, 100 para un total de 9 variantes de agrupamiento a evaluar (2 métodos de reducción * 4 cantidades de componentes + 1 variante sin reducción). Estas variantes de solución fueron evaluadas sobre las opiniones informativas de los 4 sets de datos mencionados tomando como referencia el valor del coeficiente de *Silhouette* [75].

En el caso de la evaluación de la etapa de generación de requisitos (el resultado final esperado del método propuesto en este trabajo) se realizó un caso de estudio sobre el dataset de SwiftKey en el que sus opiniones fueron filtradas y agrupadas utilizando la mejor variante de clasificación y agrupamiento encontradas en las evaluaciones anteriores de manera general. Además, se seleccionó como gran modelo de lenguaje (LLM) *Deepseek v3 [97]*, por su reciente impacto y capacidades al nivel del estado del

arte actual mientras se mantiene como opción gratuita y de código abierto disponible para desplegarse de forma local por cualquier persona con las condiciones de *hardware* suficientes para ello. Los requisitos obtenidos fueron evaluados cualitativamente según si cumplían o no 5 propiedades fundamentales para determinar el porciento de cumplimiento para cada una en los requisitos generados, así como el porciento de aquellos que las cumplen todas. Estas propiedades son:

- **Estructurado:** El requisito cumple con la estructura y formato formalmente establecidos para un requisito de software: El sistema debe [acción] [condición].
- **Bien clasificado**: La clasificación dada al requisito por el LLM al formalizarlo ("Requisito funcional" o "Requisito no funcional") es correcta.
- **Correcto:** El requisito representa de forma precisa una funcionalidad o característica que el sistema debe poseer.
- **No ambiguo:** El requisito tiene una redacción clara y con una sola interpretación posible. También se tuvo en cuenta el uso de palabras vagas como "rápido" sin aclarar una medida concreta como "respuesta en x cantidad de milisegundos"
- **Verificable:** Existe una técnica o método que en un costo finito de pasos pueda verificar que el sistema cumple el requisito descrito.

Las 2 primeras propiedades (estructurado y bien clasificado) fueron seleccionadas por un interés propio de esta investigación en comprobar las capacidades del LLM en la formalización de la estructura del requisito y en la tarea secundaria de clasificar el requisito que genera. Las 3 propiedades restantes (correcto, no ambiguo y verificable) constituyen parte de los indicadores de calidad de un requisito presentes en el estándar IEEE 830 de calidad de software [31]. La evaluación se llevó a cabo manualmente por el autor del trabajo y por cuestiones de tiempo solo se realizó sobre los requisitos generados por un solo LLM en un solo *dataset* y bajo las mismas condiciones de clasificación y agrupamiento (Las identificadas como mejores en sus propias evaluaciones).

3.2 Descripción de los sets de datos

La solución se evalúa en los 2 set de datos mencionados anteriormente, estos fueron tomados del estudio de [17]. Los autores seleccionaron como casos de estudio 4

aplicaciones disponibles en la tienda de aplicaciones Google Play: Facebook, SwiftKey Keybord, Tap Fish y Temple Run 2. Se recopilaron las reseñas de usuarios sin procesar de estas aplicaciones de Google Play aproximadamente en el período de octubre de 2012 a febrero de 2013 y fueron etiquetadas [17].

La siguiente tabla muestra una descripción de la composición de los sets de datos utilizados en cuanto a la cantidad de opiniones en cada uno, la cantidad de ejemplares de cada clasificación y el porciento de comentarios relevantes.

Tabla 3.1: Composición de los sets de datos utilizados

	Facebook	***SwiftKey***	***TapFish***	***TempleRun2***
Relevante	1662	879	888	933
No relevante	1338	2121	2112	2067
Porcentaje de relevantes	55.4%	29.3%	29,6%	31,1%
Total de comentarios	3000	3000	3000	3000

Como se puede apreciar en la tabla el set de datos de Facebook está balanceado pero el resto presenta un desbalance con una proporción cercana a (1:2) con la clase relevante como minoritaria, lo cual tiende a generar dificultades en el proceso de generalización dando prioridad a la clase mayoritaria en la predicción.

3.3 Resultados experimentales

3.3.1 Resultados experimentales de la etapa de clasificación

Tras realizar los experimentos con los modelos pre-entrenados sin conocimiento del dominio se determinó que los modelos *RoBERTa* y *BERTweet* obtuvieron los mejores resultados, por tanto, se evaluaron además estos 2 aplicándoles la técnica de conocimiento del dominio basada en los vectores de características *RC* y *RP* de manera individual. Los resultados completos son los siguientes:

Tabla 3.2: Resultados de la clasificación en el dataset de Facebook

Modelo	***Accuracy***	***Precision***	***Recall***	***F1-Score***
GPT 2 - base	0,9000	0,9009	0,8804	0,8846

XLNet - base	0,8467	0,8777	0,8099	0,8360
ALBERT - large	0,8983	0,8680	0,9041	0,8833
RoBERTa - base	0,9397	0,9101	**0,9530**	0,9306
RoBERTa + RC	0,9410	0,9171	0,9498	0,9325
RoBERTa + RP	0,9300	0,9020	0,9426	0,9192
BERTweet - base	0,9377	**0,9199**	0,9400	0,9297
BERTweet + RC	**0,9420**	0,9159	0,9526	**0,9338**
BERTweet + RP	0,9330	0,9334	0,9206	0,9254

Tabla 3.3: Resultados de la clasificación en el dataset de SwiftKey

Modelo	*Accuracy*	*Precision*	*Recall*	*F1-Score*
GPT 2 - base	0,7487	0,9235	0,7696	0,8378
XLNet - base	0,8660	0,9449	0,8784	0,9093
ALBERT - large	0,8420	0,9127	0,8851	0,8929
RoBERTa - base	0,9217	0,9457	0,9454	0,9446
RoBERTa + RC	0,9107	0,9422	0,9328	0,9369
RoBERTa + RP	0,9107	0,9422	0,9328	0,9369
BERTweet - base	0,9247	**0,9736**	0,9240	0,9481
BERTweet + RC	**0,9313**	0,9538	**0,9494**	**0,9515**
BERTweet + RP	0,9233	0,9247	0,9653	0,9443

Tabla 3.4: Resultados de la clasificación en el dataset de TapFish

Modelo	*Accuracy*	*Precision*	*Recall*	*F1-Score*
GPT 2 - base	0,9043	0,7742	0,8769	0,8166
XLNet - base	0,8863	0,9578	0,9000	0,9272
ALBERT - large	0,8110	0,9603	0,8242	0,8852
RoBERTa - base	0,9273	0,9224	0,9801	0,9496
RoBERTa+ RC	0,9347	0,9310	**0,9814**	0,9555
RoBERTa + RP	0,9280	0,9368	0,9673	0,9514
BERTweet - base	0,9413	0,9469	0,9750	0,9605
BERTweet + RC	0,9393	0,9636	0,9575	0,9601
BERTweet + RP	**0,9433**	**0,9694**	0,9564	**0,9626**

Tabla 3.5: Resultados de la clasificación en el dataset de TempleRun2

Modelo	*Accuracy*	*Precision*	*Recall*	*F1-Score*
GPT 2 - base	0,8980	0,8338	0,8229	0,8244
XLNet - base	0,8870	0,9232	0,9155	0,9186
ALBERT - large	0,8140	0,9353	0,8411	0,8792

RoBERTa - base	0,9303	**0,9668**	0,9354	0,9504
RoBERTa + RC	0,9317	0,9545	0,9471	0,9505
RoBERTa + RP	0,9310	0,9663	0,9368	0,9508
BERTweet - base	**0,9400**	0,9624	0,9516	**0,9566**
BERTweet + RC	0,9387	0,9406	**0,9697**	0,9548
BERTweet + RP	0,9343	0,9601	0,9469	0,9527

Según se aprecia en los resultados, *BERTweet-base* es el modelo pre-entrenado que utilizado sin conocimiento del dominio presenta el rendimiento más consistente en los cuatro contextos. BERTweet-base alcanza valores de *F1-Scores* superiores a 0,930 en todos los contextos, destacándose *Tap Fish* como el mejor escenario (F1-Score de 0,960). Este comportamiento de *BERTweet-base* se entiende también como robusto porque mantiene su rendimiento estable ante la presencia de alto desbalance en los *datasets*. Su variante con conocimiento del dominio (*BERTweet + RC*) presenta una mejora leve pero consistente en la mayoría de métricas para Facebook y Swiftkey, destacándose una mejora en F1-score de 0.9297 a 0.9338 y de 0,9481 a 0,9515 respectivamente. En el caso de *TapFish* y *TempleRun2*, en cambio, se evidenció una disminución muy pequeña de los valores de las métricas, con un cambio casi imperceptible en *TapFish* de 0,9605 a 0,9601 para F1-Score. En el caso de la variante con el vector *RP* la mayoría de métricas presentaron una leve caída en casi todos los *datasets*, con la excepción de *TapFish* donde incluso se volvió la mejor solución para el *dataset,* por lo que su mejora o no depende mucho del conjunto de datos.

RoBERTa-base también obtiene resultados muy competitivos, se mantiene con un rendimiento consistente en los cuatro contextos, obteniendo altos valores de *F1-Score* en todos, especialmente en *Tap Fish* (0,950) y *SwiftKey* (0,945). Su variante con conocimiento del dominio (RoBERa + RC) presenta mejoras consistentes respecto al original en 3 de los 4 datasets con solamente una leve disminución de calidad en *SwiftKey.* En el caso de la variante con el vector RP tuvo un comportamiento similar, mejorando los valores en 3 de los *datasets*, cayendo levemente en *Facebook*.

ALBERT-large tiene un rendimiento bastante equilibrado, pero no tan alto en comparación con los dos anteriores, lo que lo hace menos competitivo. Su mayor

precisión se observa en *Tap Fish* (0,960), pero falla en el *recall* que no es tan alto, lo que afecta su *F1-Score*. Su desempeño es más limitado en *Temple Run 2* y *SwiftKey*.

XLNet-base tiene un rendimiento intermedio, ya que muestra un comportamiento fuerte en algunos, pero débil en otros. *XLNet-base* tiene un desempeño superior en *SwiftKey* y *Tap Fish*, alcanzando *F1-Scores* de 0,909 y 0,927, respectivamente. Sin embargo, en Facebook y *Temple Run 2*, su rendimiento disminuye en relación con otros modelos, especialmente en *recall* (0,810 en Facebook).

Por último, *GPT2-base* muestra un rendimiento irregular, especialmente en *SwiftKey* y *Tap Fish*, donde tiene una precisión notablemente baja. GPT2-base tiene una disparidad notable entre los diferentes contextos. En *Facebook*, por ejemplo, obtiene resultados aceptables con un *F1-Score* de 0,885, pero su rendimiento disminuye en *SwiftKey* (F1-Score de 0,838). Su peor desempeño está en *Temple Run 2* y *Tap Fish*, lo que indica que podría no ser ideal para contextos más exigentes, por ejemplo, en presencia de desbalance en el *dataset*.

Tabla 3.6: Resultados generales de la clasificación

Modelos (F1-score)	***Facebook***	***SwiftKey***	***Tap Fish***	***Templerun2***	**Promedio**
GPT2 - base	0,8846	0,8378	0,8166	0,8244	0,8409
ALBERT- large	0,8881	0,8711	0,9090	0,9060	0,8935
XLNet - base	0,8360	0,9093	0,9272	0,9186	0,8978
RoBERTa - base	0,9306	0,9446	0,9496	0,9504	0,9438
RoBERTa + RC	0,9325	0,9369	0,9555	0,9505	0,9439
RoBERTa + RP	0,9192	0,9485	0,9514	0,9508	0,9425
BERTweet - base	0,9297	0,9481	0,9605	**0,9566**	0,9487
BERTweet + RC	**0,9338**	**0,9515**	0,9601	0,9548	**0,9501**
BERTweet + RP	0,9254	0,9443	**0,9626**	0,9527	0,9463

El análisis realizado sobre el comportamiento de los modelos asociados a cada conjunto de datos queda corroborado de manera general por el comportamiento medio en función de la métrica *F1-score* mostrado en la Tabla 3.6. Como se puede observar en la tabla, los modelos *RoBERTa* y *BERTweet* obtienen los mejores resultados entre los modelos evaluados, siendo los únicos con valores superiores a 0,9 en todos los *datasets*, siendo *BERTweet* ligeramente mejor. En ambos casos sus variantes con conocimiento del dominio con el vector RC presentaron mejoras respecto al promedio de su desempeño

al través de todos los *datasets,* con un aumento casi imperceptible de 0,9438 a 0,9439 en el caso de *RoBERTa* y una mejora más notable en el caso de *BERTweet* (0,9487 a 0,9501). En el caso de sus variantes con el vector RP, aunque ambas experimentaron una ligera caída en el promedio general sí hubo casos de datasets con mejora. De manera general la mejor solución es *BERTweet* con con el vector RC (*BERTweet + RC*) siendo el único con un promedio mayor a 0,95 entre todos los *datasets*.

3.3.1.1 Comparación con otros resultados en la literatura

A continuación, se compara la mejor solución obtenida en este trabajo (*BERTweet + RC*) con otras soluciones aportadas en la literatura de clasificación de opiniones para el soporte de software en el set de datos de Facebook. Para la comparación se utilizaron los valores de *F1-Score* obtenidos por cada una de las soluciones.

Tabla 3.7: Comparación con otras soluciones de clasificación de la literatura

Soluciones reportadas (*F1-Score)*	***Facebook***	***SwiftKey***	***TapFish***	***TempleRun2***	***Promedio***
AR-MINER Chen, et al. [17]	0,877	0,764	0,761	0,797	0,800
Hybrid ML Martín [20]	0,907	0,908	0,924	-	0,913
StackOBERTflow Prenner and Robbes [66]	0,909	0,851	0,894	0,886	0,885
BERT-base Prenner and Robbes [66]	0,906	0,854	0,882	0,892	0,884
BERT-SO-1M Prenner and Robbes [66]	0,921	0,875	0,900	0,911	0,902
BERT-reviews Prenner and Robbes [66]	0,933	0,899	0, 914	0,913	0,915
***BERTweet + RC* (propuesta)**	**0,934**	**0,952**	**0,960**	**0,955**	**0,950**

En la tabla 3.10 se puede ver como la solución propuesta en este trabajo (*BERTweet + RC*) supera en resultados a la mejor solución presente en la literatura para estos sets de datos (*BERT-reviews*) en cada uno de ellos, con un aumento considerable del promedio de 0,915 a 0,950. De esta forma se puede ver que el uso de conocimiento del dominio puede tener un buen impacto en el desempeño de los modelos, específicamente el uso de técnicas basadas en el uso de vectores de características que son las evaluadas en este trabajo. Cabe destacar que las mejoras de calidad más significativas se ven los *datasets SwiftKey*, *Tapfish* y *TempleRun2*, aquellos que presentan un desbalance y

representan de forma más cercana la situación del mundo real donde la mayoría de las opiniones de los usuarios no son informativas.

3.3.2 Resultados experimentales de la etapa de Agrupamiento

Al evaluar las distintas combinaciones de parámetros establecidas para el algoritmo *fuzzy C-means* para comprobar el impacto de la reducción de dimensionalidad previa a la aplicación del algoritmo se obtuvieron los resultados mostrados en la tabla 3.8. En cada caso se evaluaron valores de K desde k=10 a k=30, en la tabla se muestra el promedio de los resultados para el mejor valor de K dentro de cada *datasets* para esa configuración de parámetros, los resultados están en base al coeficiente de *Silhouette*.

Tabla 3.8: Resultados del agrupamiento en promedio entre los 4 datasets

Algoritmo de reducción	**Número de componentes**	***Silhouette Score (Promedio)***
Ninguno	-	0,007524864
PCA	2	0,341129483
PCA	10	-0,002775527
PCA	50	0,010323419
PCA	100	0,009512614
UMAP	**2**	**0,458773404**
UMAP	10	0,413834773
UMAP	50	0,403534554
UMAP	100	0,407650337

Al examinar los resultados destaca a simple vista que en promedio la combinación que mejores resultados aporta según *Silhouette* es el uso de *UMAP* como algoritmo de reducción y aplicar dicha reducción a 2 pármetros con un promedio de 0,45 de *Silhouette*. Además, se puede ver que el empleo de *UMAP* provee consistentemente mejores resultados que *PCA* para una misma cantidad de componentes. Otro elemento que destaca es que *PCA* solo es capaz de obtener valores competitivos (por encima de 0.3 de *Silhouette*) al aplicar la reducción a 2 parámetros, con otras cantidades de parámetros obtiene valores por debajo de 0.1. Finalmente, no utilizar un algoritmo de reducción dio resultados casi de 0, solo por debajo de una combinación de PCA que dio resultados negativos, por lo tanto, se comprueba que utilizar un algoritmo de reducción es una necesidad. Una comparación visual más ilustrativa de los resultados se puede ver el

gráfico de la figura 3.1 Donde se comparan los valores en promedio de cada combinación (incluye todos los valores analizados de K para cada combinación).

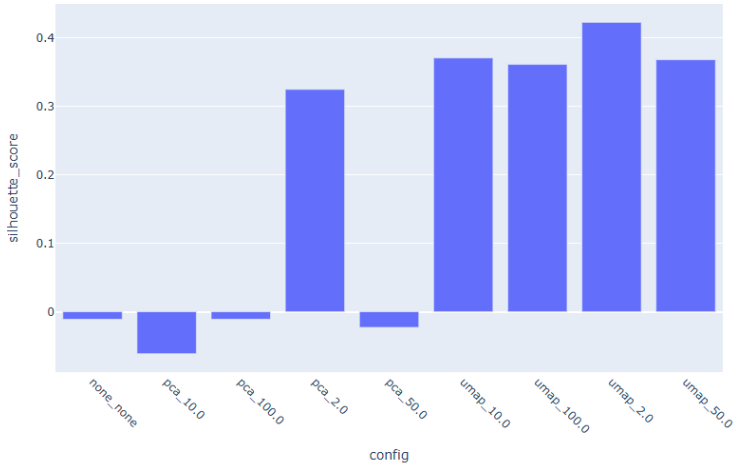


Figura 3.1 Comparación de parámetros del agrupamiento

Sobre el valor de K que determina la cantidad de clústeres se pudo comprobar que depende significativamente de cada grupo de datos y combinación específica de parámetros, por lo que se mantiene en la solución la busca de una K óptima en un rango antes de aplicar el agrupamiento por *fuzzy C-means* con la mejor variante de reducción identificada (*UMAP* a 2 componentes).

3.3.3 Resultados experimentales de la etapa de Generación

Dentro del caso de estudio de *Swifkey* se llevó a cabo el proceso de Generación de requisitos de manera completa utilizando las mejores combinaciones de soluciones encontradas en las secciones de evaluación previa (*BERTweet + RC* para clasificación y *fuzzy C-means* con reducción por *UMAP* a 2 componentes).

En este proceso se identificaron de las 3000 opiniones de usuarios iniciales un total de 862 opiniones informativas y al aplicar el agrupamiento se dividieron en 20 clústeres (al repetir aquellas opiniones que fueron asignadas a más de un clúster la cantidad de opiniones subió a 875). En la etapa de Generación inicial de requisitos se obtuvieron un total de 264 requisitos de software (162 funcionales y 102 no funcionales). Estos requisitos fueron unificados luego con el segundo procesamiento por *LLM* a 202 requisitos (110 funcionales y 202 no funcionales). La distribución exacta de clústeres y

requisitos de ambos tipos, funcionales (RF) y no funcionales (RNF) se puede observar en el anexo III.

La evaluación de la calidad de la solución de generación se basó en evaluar la calidad individual de los requisitos generados según las propiedades explicadas en el marco de evaluación (Estructurado, Bien clasificado, Correcto, No ambiguo y Verificable). Los resultados de esta evaluación se pueden apreciar en las tablas 3.9 y 3.10.

Tabla 3.9: Evaluación de calidad individual de los requisitos (Caso de estudio SwiftKey)

Cumplimiento	Estructurado	Bien clasificado	Correcto	No ambiguo	Verificable	Todo cumplido
Cantidad de Requisitos	202	163	188	134	130	109
Porciento de cumplimiento	100,00%	80,69%	93,07%	66,34%	64,36%	53,96%

Tabla 3.10: Requisitos x Cantidad de propiedades cumplidas (caso de estudio SwiftKey)

Propiedades cumplidas	Cantidad de Requisitos	Porciento de requisitos
0	0	0,00%
1	1	0,50%
2	26	12,87%
3	45	22,28%
4	21	10,40%
5	109	53,96%

En las tablas se pueden apreciar resultados prometedores, más de la mitad (53,96%) de los requisitos generados cumplen las con las 5 propiedades de calidad individual definidas y un 86,63% cumplen al menos 3 de las 5 propiedades. Si quiere ver la totalidad de los requisitos generados junto a su evaluación individual puede remitirse al anexo IV.

El análisis de cada propiedad también mostró buenos resultados se pudo ver un sorprendente 100% de cumplimiento de la propiedad Estructurado, esto demuestra que la generación a partir del LLM es capaz de imitar sin ningún problema la estructura forma de redacción de los requisitos. También, la utilidad adicional de ser capaz de clasificar el tipo de requisito generado (funcional o no funcional) demostró tener una buena eficacia, acertando en el 80,69% de los casos.

La gran mayoría de los requisitos generados eran correctos (93,07%) esto demuestra que responden a necesidades reales que la aplicación debería cumplir (o estaría bien que cumpliera a modo de mejora). Los pocos casos donde no eran correctos se debía a peticiones que infringían el límite de los permisos a los que puede acceder una app móvil como es el contexto del caso de estudio. Por ejemplo, acceso directo a qué aplicación de teclado (no solo *SwiftKey*) se ejecuta para cada aplicación dentro del móvil, esto obviamente es algo a lo que solo debería tener acceso el propio sistema operativo.

Los puntos más débiles son de los requisitos generados son sin duda la no ambigüedad (66,34%) y la verificabilidad (64,36%), esto se debe a que en algunos casos los requisitos tienden a tener especificaciones vagas o sin medidas claras que definan exactamente lo que se busca (por ejemplo, uso de adjetivos vagos como “rápido” y “mejora” o funcionalidades pobremente descritas). Sin embargo, estos casos no son la mayoría como muestran los valores porcentuales presentados y abundan también requisitos con medias muy específicas (milisegundos en requisitos de rendimiento, medidas como *WPM* (palabras por minuto) al hacer referencia a velocidad de escritura en el teclado, entre otras). Además, comúnmente los requisitos que son ambiguos no son verificables debido a esa propia ambigüedad, lo q hace que ambos valores sean muy cercanos. Los casos de requisitos que no son ambiguos pero que tampoco son verificables se deben a funcionalidades bien descritas pero que el contexto no permite ejecutar y por ende no se pueden verificar (por ejemplo, un requisito sobre la opción de mover el almacenamiento de la aplicación al ROM del celular, esto no es algo que un usuario normal pueda hacer).

Además, aparte de la evaluación individual de los requisitos se pudo comprobar que, aunque la fase de unificación ciertamente eliminó gran parte de la redundancia en los requisitos, no pudo eliminarla por completo. Todavía en los 202 requisitos finales existen algunos que hacen referencia al mismo problema o necesidad desde distintos puntos de vista (por ejemplo, varios haciendo referencia al tiempo de respuesta que debía alcanzarse desde que se toca una tecla a que aparece la letra en la pantalla, incluyendo o no en algunos casos el tratamiento a condiciones de alta velocidad de escritura).

Estos resultados afirman algo, que estos requisitos generados sin duda pueden ser útiles para un equipo de analistas y reducir enormemente el tiempo que les llevaría convertir

las opiniones de sus usuarios en formas claras de mejorar su aplicación. Por supuesto, el resultado de este método debe ser posteriormente refinado por un analista con contexto sobre las condiciones de la aplicación, su infraestructura y los intereses de la empresa, pero sin duda es mucho más eficiente analizar un borrador con recomendaciones de requisitos claros y con un cierto nivel de calidad que empezar a navegar por un mar interminable de opiniones para formularlos de cero.

3.4 Conclusiones parciales

- De los 5 modelos pre-entrenados de arquitectura evaluados sin conocimiento para la tarea de clasificación de opiniones de usuarios los mejores resultados fueron obtenidos por *BERTweet-base* y *RoBERTa-base*.
- Al evaluar la introducción de conocimiento del dominio con los mejores modelos identificados para la clasificación (*BERTweet-base* y *RoBERTa-base*) se pudo comprobar una leve mejora en los resultados de las métricas en ambos casos.
- La solución de clasificación con mejores resultados fue el uso de BERTweet-base con el vector RC para la introducción de conocimiento del dominio, con un F1-score promedio de 0.9501 entre los 4 *datasets* evaluados, superando a todas las otras soluciones del estado del arte que han sido evaluadas con esos *dataset.*
- La reducción de la dimensionalidad mediante el algoritmo UMAP a 2 componentes obtuvo los mejores resultados para el agrupamiento en todos los *datasets* según el coeficiente de *Silhouette.*
- Se generaron 202 requisitos de software recomendados para el caso de estudio de *SwiftKey* de los cuales 109 (53.97%) cumplieron con todas las propiedades de calidad de los requisitos evaluadas manualmente. Destacando que todos (100%) cumplían con la estructura especificada y 188 (93,07%) indicaban funcionalidades o características de utilidad para la aplicación. Radicando las principales carencias en la no ambigüedad (66,34%) y en la verificabilidad (64,36%).

Conclusiones Generales

Con la culminación de esta investigación se pudo llegar a las siguientes conclusiones:

- El análisis de la literatura mostró que, aunque existen trabajos sobre la generación automática de requisitos de software, en ningún caso se ha experimentado con la posibilidad de generarlos a partir de opiniones de usuarios para guiar el crecimiento de software ya desplegado.
- Ninguna de las soluciones de detección de opiniones informativas en la literatura que enfrentan este problema mediante modelos de lenguaje de arquitectura *Transformer* utilizan un enfoque para inyectar conocimiento del dominio en el modelo.
- La introducción de conocimiento del dominio en los modelos para la etapa de clasificación provocó una mejora leve en los valores de las métricas obtenidas, indicando un potencial camino a seguir para mejorar la calidad de este tipo de soluciones.
- La mejor variante de clasificación identificada fue el uso de *BERTweet-base* en conjunto con el vector de conocimiento RC, la cual superó al resto de soluciones del estado del arte en los *datasets* evaluados.
- La reducción de la dimensionalidad previa al agrupamiento demostró ser crucial en la mejora de los resultados, destacando como mejor variante la reducción mediante el algoritmo *UMAP* a 2 componentes.
- La generación de requisitos mediante *LLMs* demostró ser consistente con las opiniones origen y aportar recomendaciones de utilidad para la aplicación objetivo.
- El empleo de un segundo LLM para eliminar la redundancia en los requisitos generados demostró eliminar gran parte de los requisitos redundantes, aunque no eliminó la redundancia completamente.
- La calidad de los requisitos generados fue aceptable, cumpliendo más de la mitad con todas las propiedades evaluadas en el caso de estudio. Sin embargo, sigue siendo importante una refinación final por parte de ingenieros humanos antes de ser utilizados.

Recomendaciones

Como todo en esta vida este proyecto es imperfecto y como continuidad a la investigación desarrollada se proponen las siguientes recomendaciones:

- Explorar el uso de otras técnicas de introducción de conocimiento del dominio en modelos pre-entrenados de arquitectura *Transformer* como la inyección directa de texto o la modificación de *embeddings.*
- Realizar evaluaciones de la solución propuesta, específicamente de los requisitos generados usando otros *datasets* de opiniones reconocidos por la comunidad científica o en su lugar construir uno propio para la evaluación específica de este problema.
- Evaluar otros *LLMs* en el proceso de generación de los requisitos.
- Llevar a cabo una evaluación por expertos con ingenieros de software con experiencia y más propiedades a evaluar.
- Experimentar con otras alternativas para disminuir o eliminar la redundancia de los requisitos generados aparte del uso de un segundo *LLM*.
- Evaluar el método propuesto usando opiniones en español.

Bibliografía

[1] I. Sommerville, *Software engineering*, 10th ed. Pearson, 2016.

[2] P. A. Laplante and M. H. Kassab, F. Edition, Ed. *Requirements Engineering for Software and Systems*. CRC Press, 2022.

[3] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE software,* vol. 33, no. 1, pp. 48-54, 2015.

[4] M. Nayebi, H. Cho, and G. Ruhe, "App store mining is not enough for app improvement," *Empirical Software Engineering,* vol. 23, pp. 2764-2794, 2018.

[5] F. A. Shah, K. Sirts, and D. Pfahl, "Using app reviews for competitive analysis: tool support," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, 2019, pp. 40-46.

[6] F. Palomba *et al.*, "Crowdsourcing user reviews to support the evolution of mobile apps," *Journal of Systems and Software,* vol. 137, pp. 143-162, 2018.

[7] E. Guzman, L. Oliveira, Y. Steiner, L. C. Wagner, and M. Glinz, "User feedback in the app store: a cross-cultural study," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Society*, 2018, pp. 13-22.

[8] F. M. Kifetew, A. Perini, A. Susi, A. Siena, D. Muñante, and I. Morales-Ramirez, "Automating user-feedback driven requirements prioritization," *Information and Software Technology,* vol. 138, p. 106635, 2021.

[9] H. H. Khan, M. N. Malik, Y. Alotaibi, A. Alsufyani, and S. Alghamdi, "Crowdsourced Requirements Engineering Challenges and Solutions: A Software Industry Perspective," *Computer Systems Science & Engineering,* vol. 39, no. 2, 2021.

[10] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *2013 21st IEEE international requirements engineering conference (RE)*, 2013, pp. 125-134: IEEE.

[11] E. Guzman, R. Alkadhi, and N. Seyff, "A needle in a haystack: What do twitter users say about software?," in *2016 IEEE 24th international requirements engineering conference (RE)*, 2016, pp. 96-105: IEEE.

[12] J. C. de Souza Filho, W. T. Nakamura, L. M. Teixeira, R. P. da Silva, B. F. Gadelha, and T. U. Conte, "Towards a Data-Driven Requirements Elicitation Tool through the Lens of Design Thinking," in *ICEIS (2)*, 2021, pp. 283-290.

[13] L. Wang, H. Nakagawa, and T. Tsuchiya, "Opinion Analysis and Organization of Mobile Application User Reviews," in *REFSQ Workshops*, 2020, pp. 1-9.

[14] S. U. Hassan, J. Ahamed, and K. Ahmad, "Analytics of machine learning-based algorithms for text classification," *Sustainable Operations and Computers,* vol. 3, pp. 238-248, 2022.

[15] A. Al-Hawari, H. Najadat, and R. Shatnawi, "Classification of application reviews into software maintenance tasks using data mining techniques," *Software Quality Journal,* vol. 29, pp. 667-703, 2021.

[16] N. Aslam, W. Y. Ramay, K. Xia, and N. Sarwar, "Convolutional neural network based classification of app reviews," *IEEE Access,* vol. 8, pp. 185619-185628, 2020.
[17] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 767-778.
[18] V. Milián Núñez, T. Blanco Martín, A. Simón-Cuevas, H. González Diéz, and A. Hernández González, "A Knowledge-Based User Feedback Classification Approach for Software Support," in *International Workshop on Artificial Intelligence and Pattern Recognition*, 2023, pp. 237-247: Springer.
[19] V. Milián Núñez, H. González Diéz , and A. Simón Cuevas, "Predicción de Requisitos de Software a partir de Opiniones de Usuarios," in *"Transferencia de Conocimiento en Tecnologías de la Información" (STCTI). IV Convención Científica Internacional UCLV 2023*, Santa Clara, 2021.
[20] T. B. Martín, "Método de predicción de la relevancia en opiniones de usuarios para el soporte de software," Universidad Tecnológica de La Habana "José Antonio Echeverría", 2022.
[21] P. R. Henao, J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang, "Transfer learning for mining feature requests and bug reports from tweets and app store reviews," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 2021, pp. 80-86: IEEE.
[22] M. A. Hadi and F. H. Fard, "Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews," *Empirical Software Engineering,* vol. 28, no. 4, p. 88, 2023.
[23] R. R. Mekala, A. Irfan, E. C. Groen, A. Porter, and M. Lindvall, "Classifying user requirements from online feedback in small dataset environments using deep learning," in *2021 IEEE 29th International requirements engineering conference (RE)*, 2021, pp. 139-149: IEEE.
[24] T. Dash, S. Chitlangia, A. Ahuja, and A. Srinivasan, "A review of some techniques for inclusion of domain-knowledge into deep neural networks," *Scientific Reports,* vol. 12, no. 1, p. 1040, 2022.
[25] L. Hu, Z. Liu, Z. Zhao, L. Hou, L. Nie, and J. Li, "A survey of knowledge enhanced pre-trained language models," *IEEE Transactions on Knowledge and Data Engineering,* 2023.
[26] Z. Zhao, L. Zhang, X. Lian, X. Gao, H. Lv, and L. Shi, "Reqgen: Keywords-driven software requirements generation," *Mathematics,* vol. 11, no. 2, p. 332, 2023.
[27] M. Krishna, B. Gaur, A. Verma, and P. Jalote, "Using LLMs in software requirements specifications: an empirical evaluation," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 2024, pp. 475-483: IEEE.
[28] C. Arora, J. Grundy, and M. Abdelrazek, "Advancing requirements engineering through generative ai: Assessing the role of llms," in *Generative AI for Effective Software Development*: Springer, 2024, pp. 129-148.
[29] D. M. Fernández *et al.*, "Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice," *Empirical software engineering,* vol. 22, pp. 2298-2338, 2017.

[30] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel, and F. G. de Oliveira Neto, "Requirements engineering challenges and practices in large-scale agile system development," *Journal of Systems and Software,* vol. 172, p. 110851, 2021.
[31] I. C. S. S. E. S. Committee and I.-S. S. Board, *IEEE recommended practice for software requirements specifications* (no. 1998). IEEE, 1998.
[32] L. Hickman, S. Thapa, L. Tay, M. Cao, and P. Srinivasan, "Text preprocessing for text mining in organizational research: Review and recommendations," *Organizational Research Methods,* vol. 25, no. 1, pp. 114-146, 2022.
[33] S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems*, 1st ed. O´Reilly, 2020.
[34] A. Gasparetto, M. Marcuzzo, A. Zangari, and A. Albarelli, "A survey on text classification algorithms: From text to predictions," *Information,* vol. 13, no. 2, p. 83, 2022.
[35] A. Kedia and M. Rasu, *Hands-On Python Natural Language Processing*. Packt Publishing, 2020.
[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems,* vol. 26, 2013.
[37] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532-1543.
[38] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics,* vol. 5, pp. 135-146, 2017.
[39] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning with Aplication in Python*. Springer Texts in Statistics, 2023.
[40] S. Fransiska, R. Rianto, and A. I. Gufroni, "Sentiment Analysis Provider by. U on Google Play Store Reviews with TF-IDF and Support Vector Machine (SVM) Method," *Scientific Journal of Informatics,* vol. 7, no. 2, pp. 203-212, 2020.
[41] F. Chollet, *Deep Learning with Python*, 2nd Edition ed. Manning Publications Co., 2021.
[42] I. Lauriola, A. Lavelli, and F. Aiolli, "An introduction to deep learning in natural language processing: Models, techniques, and tools," *Neurocomputing,* vol. 470, pp. 443-456, 2022.
[43] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets,* vol. 31, no. 3, pp. 685-695, 2021.
[44] M. J. Zaki and J. Wagner Meira, *Data Mining and Machine Learning Fundamental Concepts and Algorithms*, 2nd ed. Cambridge University Press, 2020.
[45] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Physica D: Nonlinear Phenomena,* vol. 404, p. 132306, 2020.
[46] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, and M. S. Siddik, "Classifying non-functional requirements using RNN variants for quality software development," in *Proceedings of the 3rd ACM SIGSOFT International Workshop*

on Machine Learning Techniques for Software Quality Evaluation, 2019, pp. 25-30.
[47] J. Á. González Barba, "Aprendizaje profundo para el procesamiento del lenguaje natural," 2017.
[48] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information,* vol. 10, no. 4, p. 150, 2019.
[49] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.
[50] Y. Guo, X. Dong, M. A. Al-Garadi, A. Sarker, C. Paris, and D. M. Aliod, "Benchmarking of transformer-based pre-trained models on social media text classification datasets," in *Proceedings of the the 18th annual workshop of the australasian language technology association*, 2020, pp. 86-91.
[51] H. M. Zahera and M. A. Sherif, "ProBERT: Product Data Classification with Fine-tuning BERT Model," in *MWPD@ISWC*, 2020.
[52] Y. Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692,* vol. 364, 2019.
[53] Z. Lan, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942,* 2019.
[54] D. Q. Nguyen, T. Vu, and A. T. Nguyen, "BERTweet: A pre-trained language model for English Tweets," *arXiv preprint arXiv:2005.10200,* 2020.
[55] Z. Yang, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," *arXiv preprint arXiv:1906.08237,* 2019.
[56] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog,* vol. 1, no. 8, p. 9, 2019.
[57] F. Ruggeri, M. Lippi, and P. Torroni, "Combining Transformers with Natural Language Explanations," *arXiv preprint arXiv:2110.00125,* 2021.
[58] Z. Li, X. Ding, K. Liao, B. Qin, and T. Liu, "Causalbert: Injecting causal knowledge into pre-trained models with minimal supervision," *arXiv preprint arXiv:2107.09852,* 2021.
[59] A. Lauscher, O. Majewska, L. F. Ribeiro, I. Gurevych, N. Rozanov, and G. Glavaš, "Common sense or world knowledge? investigating adapter-based knowledge injection into pretrained transformers," *arXiv preprint arXiv:2005.11787,* 2020.
[60] J. Bai *et al.*, "Syntax-BERT: Improving pre-trained transformers with syntax trees," *arXiv preprint arXiv:2103.04350,* 2021.
[61] L. Cai, J. Li, H. Lv, W. Liu, H. Niu, and Z. Wang, "Integrating domain knowledge for biomedical text analysis into deep learning: A survey," *Journal of Biomedical Informatics,* vol. 143, p. 104418, 2023.
[62] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, "ERNIE: Enhanced language representation with informative entities," *arXiv preprint arXiv:1905.07129,* 2019.
[63] M. Saeed, N. Ahmadi, P. Nakov, and P. Papotti, "RuleBERT: Teaching soft rules to pre-trained language models," *arXiv preprint arXiv:2109.13006,* 2021.

[64] A. Cadeddu *et al.*, "A comparative analysis of knowledge injection strategies for large language models in the scholarly domain," *Engineering Applications of Artificial Intelligence,* vol. 133, p. 108166, 2024.
[65] M. Ostendorff, P. Bourgonje, M. Berger, J. Moreno-Schneider, G. Rehm, and B. Gipp, "Enriching bert with knowledge graph embeddings for document classification," *arXiv preprint arXiv:1909.08402,* 2019.
[66] J. A. Prenner and R. Robbes, "Making the most of small Software Engineering datasets with modern machine learning," *IEEE Transactions on Software Engineering,* vol. 48, no. 12, pp. 5050-5067, 2021.
[67] J. Lee *et al.*, "BioBERT: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics,* vol. 36, no. 4, pp. 1234-1240, 2020.
[68] Q. Jin, B. Dhingra, W. W. Cohen, and X. Lu, "Probing biomedical embeddings from language models," *arXiv preprint arXiv:1904.02181,* 2019.
[69] Y. Peng, S. Yan, and Z. Lu, "Transfer learning in biomedical natural language processing: an evaluation of BERT and ELMo on ten benchmarking datasets," *arXiv preprint arXiv:1906.05474,* 2019.
[70] S. Xu *et al.*, "K-plug: Knowledge-injected pre-trained language model for natural language understanding and generation in e-commerce," *arXiv preprint arXiv:2104.06960,* 2021.
[71] M. Bramer, *Principles of Data Mining*. Springer-Verlang London, 2007.
[72] M. H. Ahmed, S. Tiun, N. Omar, and N. S. Sani, "Short text clustering algorithms, application and challenges: a survey," *Applied Sciences,* vol. 13, no. 1, p. 342, 2022.
[73] A. Petukhova, J. P. Matos-Carvalho, and N. Fachada, "Text clustering with large language model embeddings," *International Journal of Cognitive Computing in Engineering,* vol. 6, pp. 100-108, 2025.
[74] D. Ingaramo, D. Pinto, P. Rosso, and M. Errecalde, "Evaluation of internal validity measures in short-text corpora," in *International Conference on Intelligent Text Processing and Computational Linguistics*, 2008, pp. 555-567: Springer.
[75] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics,* vol. 20, pp. 53-65, 1987.
[76] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE transactions on pattern analysis and machine intelligence,* no. 2, pp. 224-227, 2009.
[77] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods,* vol. 3, no. 1, pp. 1-27, 1974.
[78] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: State of the art, current trends and challenges," *Multimedia tools and applications,* vol. 82, no. 3, pp. 3713-3744, 2023.
[79] C. Qian *et al.*, "The Evolution of LLM Adoption in Industry Data Curation Practices," *arXiv preprint arXiv:2412.16089,* 2024.
[80] M. U. Hadi *et al.*, "A survey on large language models: Applications, challenges, limitations, and practical usage," *Authorea Preprints,* 2023.
[81] Y. Chang *et al.*, "A survey on evaluation of large language models," *ACM transactions on intelligent systems and technology,* vol. 15, no. 3, pp. 1-45, 2024.

[82] S. Arvidsson and J. Axell, "Prompt engineering guidelines for LLMs in Requirements Engineering," 2023.
[83] X. Hou *et al.*, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology,* vol. 33, no. 8, pp. 1-79, 2024.
[84] J. Becker, J. P. Wahle, B. Gipp, and T. Ruas, "Text generation: A systematic literature review of tasks, evaluation, and challenges," *arXiv preprint arXiv:2405.15604,* 2024.
[85] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374,* 2021.
[86] (10 de febrero del 2024). *Python web*. Available: https://www.python.org/about/
[87] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771,* 2019.
[88] (25 de febrero de 2024). *PyTorch web*. Available: https://pytorch.org/features/
[89] (25 de febrero de 2024). *PyTorch Lightning web*. Available: https://lightning.ai/pytorch-lightning
[90] (25 de febrero de 2024). *TorchMetrics web*. Available: https://lightning.ai/docs/torchmetrics/stable/
[91] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010, vol. 445, no. 1, pp. 51-56: Austin, TX.
[92] (10 de febrero del 2024). *NLTK web*. Available: https://www.nltk.org/
[93] (10 de febrero del 2024). *NumPy web*. Available: https://numpy.org/about/
[94] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research,* vol. 12, pp. 2825-2830, 2011.
[95] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering,* vol. 9, no. 03, pp. 90-95, 2007.
[96] I. I. I. 24765:, "24765: 2017 Systems and Software Engineering–Vocabulary," ed: International Organization for Standardization Geneva, 2017.
[97] A. Liu *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437,* 2024.
[98] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems,* vol. 35, pp. 24824-24837, 2022.
[99] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems,* vol. 36, pp. 8634-8652, 2023.
[100] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems,* vol. 33, pp. 1877-1901, 2020.

Anexos

Anexo I: *Prompt* para la generación de requisitos

ROLE

Act as a Senior Requirements Engineer and Systems Analyst specializing in NLP. Your goal is precision, relevance, and excellence through careful analysis and refinement.

PRIMARY TASK

Analyze the provided user comments and application context to generate a **final, polished set** of clear, actionable software requirements. This involves synthesizing feedback, filtering by impact, and performing a final self-correction pass to consolidate and perfect the requirements before output.

INPUT DATA

APPLICATION CONTEXT:
{app_description}

USER COMMENTS:
{comments}

CRITICAL RULES FOR REQUIREMENTS

You must strictly follow these rules in order:

1. **Synthesize or Discard:** Your primary strategy is to **synthesize**. If multiple comments report a similar issue (e.g., app is slow, login fails), create **one single requirement** for the core problem, citing all sources. If a comment is isolated, **ignore it unless it describes a high-impact issue** (crash, security flaw, failure of a core function).
2. **Distinguish FR vs. NFR:**
    - **Functional (FR):** WHAT the system does (features, actions).
    - **Non-Functional (NFR):** HOW the system performs (speed, security, usability).
3. **Be Specific & Measurable:** Requirements must be testable. For qualities like speed or reliability, **always include a metric** (e.g., "in under 2 seconds", "with 99.9% uptime").
4. **Be Atomic:** Each requirement must describe only **one** capability or constraint.
5. **Self-Correction and Refinement (Final Check):** After mentally drafting the requirements based on the rules above, **you must perform one final review pass before generating the output.** In this pass:
    - **Consolidate Further:** Scrutinize your list for any remaining overlaps. If two requirements are still too similar (e.g., NFR-A: The system shall load the profile page in <2s and NFR-B:

The system shall load the settings page in <2s), merge them into a single, more general requirement (e.g., NFR-C: The system shall load all user-specific account pages in <2s).

- **Re-Verify Classification:** For every single requirement on your final list, double-check if its FR/NFR classification is correct.

REQUIRED OUTPUT FORMAT

The output must have two sections: Functional Requirements (FR) and Non-Functional Requirements (NFR). Each requirement must be a single line.

- **FR Format:**
  FR[NNN]: The system shall [action/feature]. (Based on comments: [List of comment numbers])
- **NFR Format:**
  NFR[NNN] ([Type]): The system shall [quality/constraint]. (Based on comments: [List of comment numbers])
  (Note: [Type] must be a classification like Performance, Usability, Reliability, etc.)

EXAMPLES OF REQUERIMENTS GENERATION

- **Example of Synthesis:** Assume comments 5 & 12 complain about slowness.
  NFR001 (Performance): The system shall complete its initial startup in under 4 seconds. (Based on comments: 5, 12)
- **Example of High-Impact Single Comment:** Assume comment 9 says "app crashed uploading a video."
  NFR002 (Reliability): The system shall handle video uploads up to 500MB with a success rate of 99.5%. (Based on comments: 9)

TASK KICK-OFF

I will now process the input data, applying all critical rules for synthesis, impact assessment, and the crucial final self-correction pass. My output will be a single, consolidated list of final requirements.

Anexo II: *Prompt* para la unificación de requisitos

ROLE

Act as a Lead Systems Analyst and Requirements Manager. Your expertise lies in reviewing requirements documentation from multiple sources, identifying overlaps, and consolidating them into a single, coherent, and de-duplicated master requirements list. Your work ensures engineering efforts are focused and efficient.

PRIMARY TASK

You will receive a document containing multiple sets of software requirements, each set generated from a different "cluster" of user feedback. Your task is to analyze all requirements across all clusters to **identify and merge similar or duplicate requirements**. The final output must be a single, unified list of requirements that is free of redundancy, while carefully preserving all unique information and traceability.

INPUT DATA

APPLICATION CONTEXT:
{app_description}

REQUERIMENTS:
{requeriments}

CORE LOGIC FOR CONSOLIDATION

You must follow this consolidation process meticulously:

1. **Identify Similar Requirements:** Scan across **all clusters** to find requirements that address the same core functionality or quality attribute. "Similar" means:
    - They describe the same user action or system feature, even with different wording.
    - They describe a quality constraint (e.g., performance) for the same or closely related parts of the application.
    - One is a specific instance of another, more general requirement.
2. **Execute the Merge:** When you find two or more similar requirements, you **must merge them into one new requirement** by following these rules:
    - **Description:** Create a new, more comprehensive description.
        - If the requirements are nearly identical, simply use the clearest phrasing.
        - If they describe related specifics (e.g., one mentions "profile page," another "settings page"), abstract to a more general term (e.g., "user account pages").
        - If they have different metrics (e.g., one says <2s and another <3s), **always adopt the stricter metric** (in this case, <2s) to ensure all original needs are met.
    - **Traceability (Based on comments):** This is critical. The new, merged requirement's comment list **must be the union of all comment lists** from the original requirements. Combine all numbers and remove duplicates.
    - **Classification:** The merged requirement should retain the original classification (FR or NFR) and NFR Type (e.g., Performance, Usability). Similar requirements should share the same type.

3. **Preserve Unique Requirements:** If a requirement from a cluster has no similar counterpart in any other cluster, it is unique. It **must be preserved as-is** and copied directly to the final list.
4. **Re-Numbering:** The final, consolidated list of requirements (both FR and NFR) must be re-numbered sequentially starting from 001 for each category.

REQUIRED OUTPUT FORMAT

Your final output must be a single, clean list, structured into two sections. It should **NOT** mention the original clusters, as the goal is a unified list.

- **Functional Requirements (FR):**
  FR[NNN]: [Consolidated description]. (Based on comments: [Combined list of numbers])
- **Non-Functional Requirements (NFR):**
  NFR[NNN] ([Type]): [Consolidated description]. (Based on comments: [Combined list of numbers])

EXAMPLE OF CONSOLIDATION

INPUT DOCUMENT:

CLUSTER A: Login & Profile Issues

FR001: The system shall allow users to reset their password via email. (Based on comments: 4, 15)

NFR001 (Performance): The system shall load the user's profile page in under 3 seconds. (Based on comments: 7, 22)

CLUSTER B: General Performance Complaints

NFR001 (Performance): The system shall load pages quickly. (Based on comments: 3, 9)

NFR002 (Performance): The system shall ensure the account dashboard loads in less than 2 seconds. (Based on comments: 18)

CLUSTER C: Account Management

FR001: The system shall provide a 'Forgot Password' link on the login screen. (Based on comments: 2, 11)

CORRECT FINAL OUTPUT:

Functional Requirements (FR)
FR001: The system shall allow users to reset their password via an email link accessible from the login screen. (Based on comments: 2, 4, 11, 15)

Non-Functional Requirements (NFR)
NFR001 (Performance): The system shall load all user account pages (including profile and dashboard) in under 2 seconds. (Based on comments: 3, 7, 9, 18, 22)

###TASK KICK-OFF

I will now process the provided multi-cluster requirements document. I will apply the core logic for consolidation to produce a single, de-duplicated, and master list of requirements in the specified format.

Anexo III: Distribución de requisitos generados (Caso de estudio SwiftKey)

Clúster	Opiniones	RF	RNF	Total de requisitos
0	61	15	6	21
1	25	1	5	6
2	39	9	4	13
3	51	12	5	17
4	52	4	8	12
5	55	10	9	19
6	56	5	4	9
7	35	5	4	9
8	40	11	8	19
9	39	7	5	12
10	55	20	10	30
11	45	8	3	11
12	35	7	2	9
13	23	5	5	10
14	44	4	4	8
15	45	6	4	10
16	74	16	5	21
17	21	5	4	9
18	56	8	5	13
19	24	4	2	6
Total inicial	875 (862 únicas)	162	102	264
Total Unificado	**875 (862 únicas)**	**110**	**92**	**202**

Anexo IV: Listado de requisitos generados (Caso de estudio SwiftKey)

Las propiedades de las columnas en orden: Estructurado (Est), Bien Clasificado (Bic), Correcto (Cor), No ambiguo (Noa) y Verificable (Ver).

ID	Requisito	Est	Bic	Cor	Noa	Ver
FR001	The system shall allow users to manually correct typos in previously typed words without resetting the entire sentence. (Based on comments: 1)	1	1	1	1	1
FR002	The system shall retain user-specific frequently used terms in its dictionary and prioritize them in suggestions. (Based on comments: 2, 11)	1	1	1	0	0
FR003	The system shall not auto-replace valid user-inputted words unless explicitly overridden by the user. (Based on comments: 3, 4, 16, 19, 25, 54)	1	1	1	0	0
FR004	The system shall prevent cursor jumps during typing, ensuring it stays at the expected position. (Based on comments: 5, 21, 34)	1	1	1	1	1
FR005	The system shall improve auto-correction accuracy for words typed in all capitals, especially the first word. (Based on comments: 6)	1	0	1	0	0
FR006	The system shall correctly handle contractions (e.g., "it's") without defaulting to incorrect replacements (e.g., "is"). (Based on comments: 7, 59)	1	1	1	1	1
FR007	The system shall ensure selected corrections are applied immediately and consistently. (Based on comments: 8, 26, 35, 49)	1	1	1	1	1
FR008	The system shall detect and adapt to email address input, disabling auto-spacing after periods and auto-capitalization. (Based on comments: 13, 24, 28, 44, 56)	1	1	1	1	1
FR009	The system shall prevent unintended word substitutions or omissions during fast typing. (Based on comments: 15, 29, 51)	1	0	1	0	0
FR010	The system shall allow users to disable auto-capitalization for specific words or languages (e.g., "i," "door"). (Based on comments: 17, 36, 43)	1	1	1	1	1
FR011	The system shall provide consistent word-selection bar visibility across all apps (e.g., Firefox, Gmail). (Based on comments: 12, 27, 50)	1	0	1	1	1
FR012	The system shall support swear words and niche vocabulary if enabled by the user. (Based on comments: 47)	1	1	1	0	0
FR013	The system shall auto-correct the last word in a message before sending if the user hits "send." (Based on comments: 57, 61)	1	1	0	1	1
FR014	The system shall avoid inserting random periods or incorrect text during typing. (Based on comments: 51, 55)	1	1	1	0	0
FR015	The system shall ensure punctuation marks (e.g., parentheses) are inputted correctly without interference. (Based on comments: 38)	1	1	1	1	1
FR016	The system shall provide an option to save the keyboard application to external storage (e.g., SD card). (Based on comments: 85)	1	0	1	1	1

FR017	The system shall provide an option to disable automatic space insertion after punctuation (.,?!) in user settings. (Based on comments: 88, 90, 91, 92, 97, 101, 102, 110, 111, 112, 116, 117, 124)	1	1	1	1	1
FR018	The system shall allow users to toggle automatic word insertion after punctuation (e.g., "I" after a full stop) in settings. (Based on comments: 88, 92, 114)	1	1	0	1	1
FR019	The system shall include an apostrophe on the main keyboard layout as a configurable option. (Based on comments: 90, 99, 115)	1	1	1	1	1
FR020	The system shall provide an option to disable the "family filter" (word censorship) in settings. (Based on comments: 96, 113, 120)	1	1	1	1	1
FR021	The system shall allow users to manually delete unwanted learned words from the language model. (Based on comments: 89, 93, 106, 119, 125)	1	1	1	1	1
FR022	The system shall offer quick-insert options for predefined strings (e.g., common punctuation, hyphens). (Based on comments: 90, 104)	1	1	1	1	1
FR023	The system shall improve punctuation prediction logic (e.g., exclamation marks, colons, semicolons) to match user intent. (Based on comments: 90, 91, 118, 122)	1	0	1	0	0
FR024	The system shall provide a voice-input-friendly method to insert punctuation (e.g., full stops). (Based on comments: 100)	1	1	1	0	0
FR025	The system shall allow users to customize or disable the slide punctuation bar. (Based on comments: 103)	1	1	1	0	0
FR026	The system shall provide additional keyboard themes for customization. (Based on comments: 126, 127, 128, 129, 132, 133, 140, 154, 155, 159, 160, 161, 163, 167, 169, 171, 173, 176, 284-339, 289, 322, 333, 291, 333, 324, 331, 554, 555, 556, 557, 560, 561, 562, 568, 570, 575, 576, 578, 579, 580, 581, 582, 584, 586, 587, 588, 852)	1	1	1	1	1
FR027	The system shall allow users to replace the smiley key with a persistent enter key. (Based on comments: 130, 145, 174)	1	1	1	1	1
FR028	The system shall support custom smiley sets and user-editable smileys. (Based on comments: 131, 147, 171, 856, 857, 859, 871, 875)	1	1	1	1	1
FR029	The system shall include an Italian keyboard layout option. (Based on comments: 138)	1	1	1	1	1
FR030	The system shall provide standard Samsung characters (e.g., hearts) in the symbol set. (Based on comments: 137, 139, 144, 158)	1	1	1	1	1
FR031	The system shall allow users to adjust key press sound volume, including quieter options. (Based on comments: 129, 135, 153, 245)	1	1	1	1	1

FR032	The system shall enable customization of key layout (e.g., alt keys, space bar size, key spacing). (Based on comments: 133, 134, 141, 143, 150, 166, 170, 175)	1	1	1	1	1
FR033	The system shall offer a collapse keyboard button for quick dismissal. (Based on comments: 156, 488, 497)	1	1	1	1	1
FR034	The system shall include copyright/trademark symbols in an easily accessible location. (Based on comments: 148, 149)	1	1	1	1	1
FR035	The system shall allow font size adjustment for keys. (Based on comments: 151)	1	1	1	1	1
FR036	The system shall provide an option to move the app to ROM for faster performance. (Based on comments: 162)	1	0	0	1	0
FR037	The system shall include a tablet-optimized layout as an in-app option. (Based on comments: 157)	1	1	1	1	1
FR038	The system shall allow users to adjust the backspace speed. (Based on comments: 179)	1	1	1	1	1
FR039	The system shall optimize keyboard responsiveness during fast typing to prevent misinterpretation as deletion swipes. (Based on comments: 177, 201)	1	0	1	0	0
FR040	The system shall ensure all essential symbols (e.g., percentage symbol) are available on the keyboard. (Based on comments: 211, 456, 457, 461, 469, 505, 506)	1	1	1	0	0
FR041	The system shall function without requiring a constant internet connection. (Based on comments: 212)	1	0	1	1	1
FR042	The system shall allow users to disable haptic feedback independently of system settings. (Based on comments: 231, 511, 518)	1	1	1	1	1
FR043	The system shall retain learned user data and preferences across hardware changes or reinstalls. (Based on comments: 254, 259, 262)	1	1	1	1	1
FR044	The system shall provide an incompatibility warning before purchase or installation. (Based on comments: 236)	1	1	1	1	1
FR045	The system shall support Vietnamese language input. (Based on comments: 243, 705, 706, 740)	1	1	1	1	1
FR046	The system shall allow users to toggle keypress sound settings independently. (Based on comments: 245)	1	1	1	1	1
FR047	The system shall enable split keyboard functionality for large-screen devices (e.g., Galaxy Note). (Based on comments: 255, 777)	1	1	1	1	1
FR048	The system shall allow per-app keyboard assignment to avoid conflicts. (Based on comments: 276)	1	1	0	1	0
FR049	The system shall restore autocorrect functionality after OS updates (e.g., Android Jelly Bean). (Based on comments: 234, 247)	1	0	1	0	0
FR050	The system shall ensure pasting functionality remains stable across updates. (Based on comments: 240)	1	0	1	0	0
FR051	The system shall prevent repeated default keyboard notifications. (Based on comments: 258)	1	0	1	0	0

FR052	The system shall implement swipe-to-type (Swype-like) functionality. (Based on comments: 510, 512, 513, 514, 515, 516, 517, 521, 524, 526, 527, 528, 529, 530, 531, 534, 535, 537, 538, 539, 540, 541, 542, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 464)	1	1	1	1	1
FR053	The system shall include a dedicated delete key. (Based on comments: 520, 522)	1	1	1	1	1
FR054	The system shall support one-handed mode for improved usability. (Based on comments: 523, 777)	1	1	1	1	1
FR055	The system shall optimize key sizing and layout for 7-inch tablets. (Based on comments: 525)	1	0	1	0	0
FR056	The system shall include a screenshot key. (Based on comments: 509)	1	1	1	1	1
FR057	The system shall support swipe-left to delete and swipe-right to redo. (Based on comments: 552)	1	1	1	1	1
FR058	The system shall allow users to create and apply custom themes, including hex color inputs and file-based skins. (Based on comments: 556, 561, 564, 565, 567, 569, 572, 574, 579, 585)	1	1	1	1	1
FR059	The system shall enable customization of functional key colors (e.g., shift, backspace) independently of the theme. (Based on comments: 583)	1	1	1	1	1
FR060	The system shall support font customization that adapts to the selected theme. (Based on comments: 577)	1	1	1	0	0
FR061	The system shall allow disabling the keypress letter highlight. (Based on comments: 569)	1	1	1	1	1
FR062	The system shall provide a "cleaner" interface option. (Based on comments: 566)	1	0	1	0	0
FR063	The system shall enable import of settings from the trial version. (Based on comments: 573)	1	1	1	1	1
FR064	The system shall successfully download and install language packs without errors. (Based on comments: 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 605, 606, 607, 608, 610, 611)	1	0	1	0	0
FR065	The system shall provide a clear error message when language pack downloads fail, including actionable steps to resolve the issue. (Based on comments: 591, 593, 594, 605, 606)	1	1	1	1	1
FR066	The system shall allow language packs to be imported from the free version to the paid version during installation. (Based on comments: 601)	1	1	1	1	1
FR067	The system shall support downloading language packs across multiple devices without requiring separate purchases. (Based on comments: 609)	1	1	1	1	1
FR068	The system shall provide a refund or recovery option if language pack downloads fail, preventing usability issues. (Based on comments: 610)	1	1	1	1	1

FR069	The system shall retain the user-selected keyboard as the default input method after device reboot, battery removal, or USB storage disconnection. (Based on comments: 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 626, 627, 628, 629, 630, 631, 632, 633, 635, 636, 638, 639, 640, 644, 646, 647, 648, 649, 650, 652, 653, 654, 655)	1	0	0	1	0
FR070	The system shall automatically load and enable SwiftKey as the active keyboard upon device startup without requiring manual re-selection in settings. (Based on comments: 619, 620, 632, 640, 649)	1	0	0	1	0
FR071	The system shall dynamically adjust the keyboard size and positioning when used in multi-window or split-screen mode to match the behavior of the default Samsung keyboard. (Based on comments: 637)	1	1	1	1	1
FR072	The system shall prevent text field corruption or resizing when the screen is locked and unlocked during active input. (Based on comments: 625)	1	0	1	1	1
FR073	The system shall allow users to define custom words and shortcuts in a personal dictionary. (Based on comments: 656, 660, 661, 662, 664, 665, 666, 668, 670, 672, 674, 678, 679, 680, 681, 686, 688, 690, 696, 699)	1	1	1	1	1
FR074	The system shall sync the personal dictionary to the cloud (e.g., Google Drive) for backup and multi-device access. (Based on comments: 660, 662, 677, 680, 700)	1	1	1	1	1
FR075	The system shall learn and suggest words from WhatsApp messages. (Based on comments: 659, 665, 667, 676, 684, 685, 687, 692, 694)	1	1	0	0	0
FR076	The system shall allow users to manually add, edit, or remove words from the dictionary, including in bulk. (Based on comments: 664, 668, 670, 678, 681, 699)	1	1	1	1	1
FR077	The system shall improve language dictionaries (Polish, French, German, Swedish, Turkish) for better word recognition and suggestions. (Based on comments: 657, 658, 683, 691, 695)	1	0	1	0	0
FR078	The system shall allow users to toggle swear word filtering. (Based on comments: 669, 682, 689)	1	1	1	1	1
FR079	The system shall support Macedonian language input. (Based on comments: 701)	1	1	1	1	1
FR080	The system shall support Traditional Chinese language input. (Based on comments: 702, 715, 728, 739, 746, 750, 760, 762, 763, 771)	1	1	1	1	1
FR081	The system shall support Vietnamese language input. (Based on comments: 705, 706, 740)	1	1	1	1	1
FR082	The system shall improve Arabic language support, including layout adjustments and letter sizing. (Based on comments: 707, 756, 758, 774)	1	0	1	0	0

FR083	The system shall support Japanese language input. (Based on comments: 720, 732, 737, 745, 747, 759, 764, 769)	1	1	1	1	1
FR084	The system shall support Faroese language input with personalized word memory. (Based on comments: 712)	1	1	1	1	1
FR085	The system shall support Polish language input. (Based on comments: 714)	1	1	1	1	1
FR086	The system shall support T9 input method. (Based on comments: 716)	1	1	1	1	1
FR087	The system shall allow manual language selection instead of auto-detection. (Based on comments: 731, 770)	1	1	1	1	1
FR088	The system shall support Marathi and Hindi language input. (Based on comments: 722, 751, 767)	1	1	1	1	1
FR089	The system shall support Thai language input. (Based on comments: 723, 729, 734, 761, 753)	1	1	1	1	1
FR090	The system shall continue improving Spanish language support. (Based on comments: 724)	1	0	1	0	0
FR091	The system shall support Welsh, Basque, Catalan, Galician, and Icelandic language input. (Based on comments: 727)	1	1	1	1	1
FR092	The system shall support Hinglish (Hindi-English) language input. (Based on comments: 733)	1	1	1	1	1
FR093	The system shall support Chinese input via stroke-based method. (Based on comments: 735)	1	1	1	1	1
FR094	The system shall support Han Yin Pinyin input. (Based on comments: 749)	1	1	1	1	1
FR095	The system shall support S-Pen detection and handwriting input for Samsung Galaxy Note 2/Note II. (Based on comments: 775, 776, 780, 781, 782, 784, 785, 786, 789, 790, 792, 793, 795)	1	1	1	1	1
FR096	The system shall allow seamless switching between keyboard and handwriting input modes without disabling native handwriting features. (Based on comments: 776, 781, 789)	1	1	1	0	0
FR097	The system shall provide one-handed operation support for large-screen devices (e.g., Galaxy Note 2). (Based on comments: 777)	1	1	1	1	1
FR098	The system shall include multi-touch support for fast typers. (Based on comments: 794)	1	0	1	0	0
FR099	The system shall correctly adapt to different screen sizes (e.g., LG Intuition) without display stretching issues. (Based on comments: 783)	1	0	1	1	1
FR100	The system shall allow users to toggle word predictions on/off globally. (Based on comments: 801, 810, 811, 827, 848, 849)	1	1	1	1	1
FR101	The system shall provide a user-editable dictionary to add/remove words from predictions. (Based on comments: 798, 808, 812, 816)	1	1	1	1	1
FR102	The system shall ensure word predictions appear consistently across all apps, including browsers, unless in	1	0	1	1	1

	username/password fields. (Based on comments: 802, 835, 841, 844, 850, 851)					
FR103	The system shall restore prediction functionality after screen wake or keyboard reopen without manual intervention. (Based on comments: 814, 815, 845)	1	0	1	1	1
FR104	The system shall support multi-language prediction without bias toward one language. (Based on comments: 821, 828, 829)	1	0	0	0	0
FR105	The system shall include swear words, custom words, and abbreviations (e.g., "w/") in predictions if frequently used. (Based on comments: 830)	1	0	1	0	0
FR106	The system shall prevent random word insertion after punctuation and space. (Based on comments: 803)	1	0	1	1	1
FR107	The system shall retain prediction learning across sessions, including older texts with special terminology. (Based on comments: 826)	1	0	1	0	0
FR108	The system shall provide emoji support in the keyboard. (Based on comments: 852, 853, 854, 855, 856, 857, 858, 859, 860, 862, 863, 864, 865, 866, 867, 868, 870, 871, 872, 874, 875)	1	1	1	1	1
FR109	The system shall allow users to switch between regular smileys and emojis in the keyboard. (Based on comments: 856)	1	1	1	1	1
FR110	The system shall provide customizable emoticon/emoji options (e.g., hearts, Gtalk emoticons). (Based on comments: 857, 859, 871, 875)	1	1	1	1	1
NFR001	The system shall maintain auto-correction responsiveness with no perceptible lag (<0.5s delay). (Based on comments: 9, 23, 41, 60, 351, 364, 369, 374)	1	1	1	1	1
NFR002	The system shall function without unexpected closures or freezes during typing sessions. (Based on comments: 14, 30, 31, 229, 232, 238, 249, 256, 266, 274, 275, 283, 388, 403, 409, 411, 437, 449)	1	1	1	0	0
NFR003	The system shall register keystrokes instantly (<100ms delay) to prevent input lag. (Based on comments: 41, 53, 178, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 199, 200, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 242, 270)	1	1	1	1	1
NFR004	The system shall adapt to multilingual input (e.g., Portuguese) with accurate auto-correction. (Based on comments: 46, 709, 717, 719, 754, 755, 772)	1	1	1	0	0
NFR005	The system shall retain user-specific settings (e.g., email addresses, dictionaries) across sessions. (Based on comments: 42, 671, 677, 680, 700)	1	1	1	1	1

NFR006	The system shall provide clear visual feedback when auto-correction fails or underlines misspelled words. (Based on comments: 26, 49)	1	1	1	1	1
NFR007	The system shall reduce RAM usage to a maximum of 15MB during active keyboard operation. (Based on comments: 62, 63, 64, 65, 66, 69, 71, 72, 73, 74, 75, 76, 77, 79, 80, 81)	1	1	1	1	1
NFR008	The system shall minimize background RAM usage to prevent slowdowns on the device. (Based on comments: 64, 65, 66, 68, 76, 80)	1	1	1	0	0
NFR009	The system shall optimize battery consumption to avoid significant battery drain. (Based on comments: 67, 78, 82, 83, 84)	1	1	1	0	0
NFR010	The system shall initialize and load the keyboard interface within 1 second in all apps. (Based on comments: 68, 86, 186, 189, 193, 203, 209, 210, 213, 224, 438, 448)	1	1	1	1	1
NFR011	The system shall prevent visual tearing or lag during text input. (Based on comments: 70)	1	1	1	0	0
NFR012	The system shall ensure punctuation keys (apostrophes, colons, semicolons) behave consistently with commas and full stops regarding space placement. (Based on comments: 90, 98)	1	1	1	0	0
NFR013	The system shall minimize incorrect word predictions (e.g., "election" for "erection") with an accuracy rate of ≥95%. (Based on comments: 107, 109, 123, 279)	1	1	1	1	1
NFR014	The system shall retain user-saved words with a persistence rate of ≥99%. (Based on comments: 93, 119, 125)	1	1	1	0	0
NFR015	The system shall allow spacebar insertion after a word at the end of a line without unintended behavior. (Based on comments: 95)	1	1	1	0	0
NFR016	The system shall ensure keyboard responsiveness meets user expectations for typing speed. (Based on comments: 133, 146, 176)	1	1	1	0	0
NFR017	The system shall offer intuitive key placement and layout consistency with market standards. (Based on comments: 143, 146, 170)	1	1	1	0	0
NFR018	The system shall provide extensive personalization options (themes, colors, sounds) to match user preferences. (Based on comments: 128, 140, 142, 152, 155, 159, 171)	1	0	1	0	0
NFR019	The system shall minimize input lag during typing. (Based on comments: 133, 146, 162)	1	1	1	0	0
NFR020	The system shall support adjustable key visuals (e.g., spacing, font size) for improved readability. (Based on comments: 151, 164)	1	0	1	1	1
NFR021	The system shall reduce input lag to ensure keystrokes register within 100ms. (Based on comments: 178, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 199, 200, 202, 203, 204, 205, 206, 207,	1	1	1	1	1

	208, 209, 210, 211, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228)					
NFR022	The system shall initialize and appear on-screen within 1 second of invocation. (Based on comments: 186, 189, 193, 203, 209, 210, 213, 224)	1	1	1	1	1
NFR023	The system shall minimize performance degradation when switching between apps or waking from deep sleep. (Based on comments: 196, 218, 225)	1	1	1	0	0
NFR024	The system shall reduce freezing incidents to fewer than 1 occurrence per 10,000 keystrokes. (Based on comments: 182, 191, 193, 217, 219, 225)	1	1	1	1	1
NFR025	The system shall maintain responsiveness during rapid typing (≥150 WPM) without freezing. (Based on comments: 177, 201, 209, 215, 227)	1	1	1	0	0
NFR026	The system shall optimize performance across Android versions (4.1--4.2.1) and devices (e.g., GS3, Nexus, Xperia). (Based on comments: 178, 181, 184, 188, 192, 199, 207, 216, 228, 239, 244, 247, 260, 263)	1	1	1	0	0
NFR027	The system shall prevent app-wide lag when the keyboard is active (e.g., in Google Drive, browsers). (Based on comments: 187, 205, 208, 214, 226)	1	1	1	0	0
NFR028	The system shall handle charging-state usage without erratic behavior. (Based on comments: 197)	1	1	1	0	0
NFR029	The system shall achieve a crash rate of <0.1% during typing in supported apps (e.g., Chrome, Tumblr, Viber, WhatsApp). (Based on comments: 229, 232, 238, 249, 256, 266, 274, 275, 283)	1	1	1	1	1
NFR030	The system shall process keystrokes with <100ms latency on devices with mid-range processors (e.g., Galaxy S3, Tab 10.1). (Based on comments: 229, 242, 270)	1	1	1	1	1
NFR031	The system shall maintain functionality across Android OS versions 4.0--4.1 without force closes. (Based on comments: 239, 244, 247, 260, 263)	1	1	1	0	0
NFR032	The system shall retain keyboard activation state after device reboot or app switching. (Based on comments: 241, 250, 260, 282)	1	1	0	0	0
NFR033	The system shall prevent device hangs or battery pulls due to keyboard crashes. (Based on comments: 230, 246)	1	1	1	1	1
NFR034	The system shall load suggestions consistently without requiring manual keyboard reopening. (Based on comments: 252, 277, 280, 281)	1	1	1	1	1
NFR035	The system shall minimize incorrect autocorrect substitutions (e.g., only apply suggestions when explicitly selected). (Based on comments: 279)	1	1	0	0	0
NFR036	The system shall support Moto LapDock and similar peripherals. (Based on comments: 237)	1	1	1	1	1

NFR037	The system shall clarify data collection practices during installation. (Based on comments: 268)	1	1	1	1	1
NFR038	The system shall provide a diverse range of theme colors, including all major colors (especially red, green, pink). (Based on comments: 284-339, 322, 315)	1	0	1	1	1
NFR039	The system shall regularly update theme offerings (at least quarterly) to maintain user engagement. (Based on comments: 287, 310, 328)	1	1	1	1	1
NFR040	The system shall include gender-neutral and non-"girly" theme options. (Based on comments: 310)	1	1	1	0	0
NFR041	The system shall ensure themes have a modern and appealing look and feel. (Based on comments: 303, 312, 338)	1	1	1	0	0
NFR042	The system shall maintain predictive text functionality with a success rate of 99% across all supported applications. (Based on comments: 340, 341, 342, 343, 344, 349, 350, 353, 354, 355, 357, 361, 365, 366, 367, 371, 372)	1	1	1	0	0
NFR043	The system shall ensure auto-correction and predictive text respond within 0.5 seconds of user input. (Based on comments: 351, 364, 369, 374)	1	1	1	1	1
NFR044	The system shall handle fast typing inputs without deleting or disrupting the user's text. (Based on comments: 358, 364)	1	1	1	0	0
NFR045	The system shall provide consistent keyboard layout and key spacing to reduce typing errors. (Based on comments: 357, 369)	1	1	1	0	0
NFR046	The system shall not force-close or crash during use in Chrome or other browsers. (Based on comments: 388, 403, 409, 411)	1	1	1	1	1
NFR047	The system shall backspace through words without lag or erratic behavior (e.g., deleting chunks). (Based on comments: 375, 386, 392)	1	1	1	1	1
NFR048	The system shall display autocorrect suggestions consistently across all apps (e.g., Firefox, default browser). (Based on comments: 390, 391, 392)	1	1	1	1	1
NFR049	The system shall provide transparent opt-in controls for data access (e.g., email, Facebook) with clear justification. (Based on comments: 396, 398)	1	1	1	1	1
NFR050	The system shall ensure the keyboard does not obstruct input fields or jump erratically in browser forms. (Based on comments: 379, 395)	1	1	1	0	0
NFR051	The system shall allow users to disable landscape mode if preferred. (Based on comments: 402)	1	0	1	1	1
NFR052	The system shall provide additional theme options for customization. (Based on comments: 408)	1	0	1	0	0
NFR053	The system shall process rapid typing inputs without misinterpreting spacebar presses as ".com" button presses. (Based on comments: 412)	1	1	1	0	0

NFR054	The system shall reduce input lag to under 200ms on Samsung Galaxy S3 running Jelly Bean. (Based on comments: 417, 420, 427, 438, 448)	1	1	1	1	1
NFR055	The system shall not override system-wide haptic feedback settings. (Based on comments: 428, 435)	1	1	0	1	1
NFR056	The system shall achieve a crash rate of <0.1% on Jelly Bean devices. (Based on comments: 437, 449)	1	1	1	1	1
NFR057	The system shall support hardware keyboards without functional issues. (Based on comments: 440)	1	1	1	1	1
NFR058	The system shall load the keyboard interface within 1 second of selection. (Based on comments: 438, 448)	1	1	1	1	1
NFR059	The system shall ensure swipe-to-type sensitivity matches or exceeds Google's implementation. (Based on comments: 522, 527, 545)	1	1	1	0	0
NFR060	The system shall provide customization options to compete with Swype in efficiency. (Based on comments: 519)	1	0	1	0	0
NFR061	The system shall adapt to large-screen devices without functional degradation. (Based on comments: 532)	1	1	1	0	0
NFR062	The system shall ensure theme customization is intuitive and accessible within 3 user actions. (Based on comments: 554, 556, 564, 565, 567, 569, 574, 585)	1	1	1	1	1
NFR063	The system shall prevent keystroke over-sampling during charging. (Based on comments: 558)	1	1	1	1	1
NFR064	The system shall ensure language pack downloads succeed with a 99% success rate under stable network conditions. (Based on comments: 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 605, 606, 607, 608, 611)	1	1	1	1	1
NFR065	The system shall notify users in advance before updating or invalidating existing language packs, especially when network access is unavailable. (Based on comments: 603)	1	1	0	1	1
NFR066	The system shall download and install language packs within 2 minutes on a stable 4G/LTE connection. (Based on comments: 589, 590, 591, 594, 599, 605, 606)	1	1	1	1	1
NFR067	The system shall maintain functionality and support language pack downloads across OS updates (e.g., Android Jelly Bean and later). (Based on comments: 600)	1	1	1	1	1
NFR068	The system shall validate network connectivity before attempting language pack downloads and provide a retry mechanism if the connection is unstable. (Based on comments: 605, 606, 607)	1	0	1	1	1
NFR069	The system shall maintain the default keyboard selection across device restarts, battery removals, and USB disconnections with 99.9% consistency. (Based on comments: 612, 613, 614, 615, 616, 617, 621, 622, 626, 627, 628, 629, 630, 631, 632, 633, 635, 636, 638, 639, 640, 644, 646, 647, 648, 649, 650, 652, 653, 654, 655)	1	1	0	1	1

NFR070	The system shall provide clear visual feedback or notification when the keyboard fails to load or defaults to the stock keyboard, guiding the user to resolve the issue. (Based on comments: 618, 619, 620, 624, 632, 640, 649)	1	1	1	1	1
NFR071	The system shall ensure backward compatibility with Android versions post-Jelly Bean (4.2+) to prevent uninstallation or disappearance after OS updates. (Based on comments: 623, 642)	1	1	1	1	1
NFR072	The system shall respond to input requests (e.g., text field focus) within 0.5 seconds across all supported apps. (Based on comments: 620, 641)	1	1	1	1	1
NFR073	The system shall provide accurate word suggestions across multiple active languages (e.g., German, English, Hebrew) with a success rate of ≥95%. (Based on comments: 658, 673, 683, 691)	1	1	1	1	1
NFR074	The system shall retain learned words and settings across app updates and device resets. (Based on comments: 671, 677, 680, 700)	1	1	1	1	1
NFR075	The system shall process and learn new words from all typing inputs (including WhatsApp) within 24 hours of repeated use. (Based on comments: 663, 675, 676, 684)	1	1	1	0	0
NFR076	The system shall allow composite word handling (e.g., for Swedish) without splitting. (Based on comments: 683)	1	1	1	1	1
NFR077	The system shall minimize the need to switch between keyboards for multilingual users. (Based on comments: 709, 717, 719, 754, 755, 772)	1	1	1	0	0
NFR078	The system shall allow users to adjust language priorities for prediction. (Based on comments: 718)	1	0	1	1	1
NFR079	The system shall improve autocorrect accuracy for multilingual input (e.g., German/English/Hebrew). (Based on comments: 757)	1	1	1	0	0
NFR080	The system shall ensure Arabic keyboard letters are clearly visible and easy to type. (Based on comments: 758, 774)	1	1	1	0	0
NFR081	The system shall handle seamless switching between languages without lag. (Based on comments: 709, 719, 755)	1	1	1	0	0
NFR082	The system shall ensure swipe-down gestures to minimize the keyboard work with 95% accuracy. (Based on comments: 778)	1	1	1	1	1
NFR083	The system shall simplify the setup process to reduce user confusion. (Based on comments: 779, 791)	1	1	1	0	0
NFR084	The system shall maintain full keyboard functionality when used with phone cases. (Based on comments: 787)	1	1	1	0	0
NFR085	The system shall minimize the need to switch between keyboards for S-Pen users. (Based on comments: 788, 789)	1	1	1	0	0
NFR086	The system shall display word predictions with ≥95% uptime during typing sessions. (Based on comments: 797, 800, 809, 813, 831, 833, 836, 837, 846)	1	1	1	1	1

NFR087	The system shall offer an option to disable auto-insertion of predictions. (Based on comments: 820)	1	0	1	1	1
NFR088	The system shall load predictions within 0.5 seconds of keystroke. (Based on comments: 805, 824)	1	1	1	1	1
NFR089	The system shall provide prediction accuracy parity between English and non-English languages (e.g., Lithuanian, Romanian). (Based on comments: 807, 828)	1	1	0	0	0
NFR090	The system shall allow long-press deletion of incorrect predictions. (Based on comments: 812)	1	0	1	1	1
NFR091	The system shall ensure emoticons/emojis are easily accessible (e.g., via a dedicated button or quick-switch option). (Based on comments: 856, 873)	1	1	1	0	0
NFR092	The system shall provide a support forum or feedback channel for users. (Based on comments: 861)	1	0	1	1	1