UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA DEL SOFTWARE

Desarrollo de una herramienta de análisis de aplicaciones Android mediante una pipeline de machine learning basada en Transformers.

Autor: Alejandro Barreiro Morante

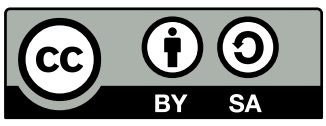
Director: Alejandro Martín García

Madrid, May 2023

Alejandro Barreiro Morante

Desarrollo de una herramienta de análisis de aplicaciones Android mediante una pipeline de machine learning basada en Transformers.
Proyecto Fin de Grado, Friday 5º May, 2023
Director: Alejandro Martín García

E.T.S. de Ingeniería de Sistemas Informáticos
Campus Sur UPM, Carretera de Valencia (A-3), km. 7
28031, Madrid, España



Yo, **Alejandro Barreiro Morante**, estudiante de la titulación **Grado en Ingeniería del Software** de la de **E.T.S. de Ingeniería de Sistemas Informáticos** de la **Universidad Politécnica de Madrid**, como **autor** del Proyecto Fin de Grado titulado:

Desarrollo de una herramienta de análisis de aplicaciones Android mediante una pipeline de machine learning basada en Transformers.

DECLARO QUE

Este proyecto es una obra original y que todas las fuentes utilizadas para su realización han sido debidamente citadas en el mismo. Del mismo modo, asumo frente a la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad del contenido de la memoria presentada de conformidad con el ordenamiento jurídico vigente.

Madrid, a Friday 5º May, 2023

Fdo.: Alejandro Barreiro Morante
Autor del Proyecto Fin de Grado

Resumen

En este trabajo de fin de grado se explora el desarrollo de una herramienta de aprendizaje automático que facilita el análisis de aplicaciones Android. La herramienta se basa en una novedosa arquitectura conocida como Transformer, que ha demostrado su eficacia en múltiples campos relacionados con el modelado de secuencias. La herramienta consta de dos modelos con propósitos diferentes: el primero, un modelo de código llamado CodeT5, se encarga de resumir el código fuente de los métodos de las aplicaciones. El segundo modelo, un modelo de lenguaje natural llamado T5, se encarga de resumir las clases filtrando los resúmenes producidos por CodeT5. Para obtener el código fuente de las aplicaciones utilizaremos un conocido decompilador de Android, Jadx. Finalmente, la herramienta podrá ser accedida a través de una interfaz web construida con el framework de Python Dash.

Por otra parte, este trabajo ofrece una visión general del funcionamiento y partes de la arquitectura Transformer, así como de los modelos más relevantes de Transformers para código fuente publicados y del modelo de lenguaje natural T5. Además, se realiza una revisión de las características principales de Android, plataforma en la que se ejecutan las aplicaciones analizadas.

En definitiva, se demuestra que la herramienta propuesta se muestra prometedora como instrumento para analizar y resumir aplicaciones Android, aunque la calidad de los resúmenes producidos no es la ideal, presentando dificultades para comprender los elementos ofuscados derivados de la decompilación de las aplicaciones. No obstante, con la aplicación de mejores estrategias, la herramienta tiene un gran potencial futuro.

Palabras clave: Transformer;Android;Procesado de código;Procesado de lenguaje natural;Resumen de secuencias;Ciberseguridad

Abstract

In this final degree project, we explore the development of a machine learning tool that facilitates the analysis of Android applications. The tool is based on a novel architecture known as Transformers, which has been shown to be effective in multiple fields related to sequence modeling. The tool consists of two models with different purposes: the first, a code model called CodeT5, is responsible for summarizing the source code of the methods of the Android applications. The second model, a natural language model called T5, is responsible for summarizing the classes by filtering the summaries produced by CodeT5. To obtain the source code of the applications, we use well-known Android decompiler, Jadx. The tool can be accessed through a web interface built with the Python Dash framework.

In addition to the development of the tool, this work offers a general overview of the Transformer architecture and the most relevant transformer models published for source code and natural language processing. We also provide a review of the key characteristics of the Android platform, on which the analyzed applications are executed.

In summary, the proposed tool shows promise as an instrument for analyzing and summarizing Android applications, although the quality of the produced summaries is not ideal, presenting difficulties in understanding obfuscated elements that are derived from the decompilation of the applications. However, with the implementation of better strategies, the tool has great potential for future use.

Keywords: Transformer;Android;Source code processing;Natural language Processing;Sequence summary;Cybersecurity

Índice general

Índice de figuras

Índice de listados

1. Introducción

En el panorama tecnológico actual, que avanza con rapidez, el uso de dispositivos móviles y de IoT está cada vez más extendido. Se espera que esta tendencia continúe en los próximos años y, con ella, la popularidad del sistema operativo Android.

Como plataforma de código abierto, Android ha dado lugar a una amplia gama de dispositivos, incluidos teléfonos inteligentes, tabletas y dispositivos personales como smartwatches y televisores. La necesidad de herramientas automatizadas para analizar el funcionamiento de las aplicaciones Android en estos dispositivos, con el fin de garantizar la seguridad y la privacidad de los usuarios, es cada vez más importante.

Además, el panorama de la ciberseguridad ha cambiado significativamente en comparación con el pasado. Antes, la mayoría de los ataques eran relativamente sencillos y poco sofisticados. Sin embargo, en los últimos años, los atacantes se han profesionalizado y sofisticado, destacando la existencia de organizaciones de cibercriminales con forma de empresa y utilizando técnicas avanzadas como exploits de día cero y ataques personalizados a objetivos concretos. Como resultado, los ataques de ciberseguridad son ahora más peligrosos y difíciles de comprender que nunca.

Por lo tanto, no basta con disponer de herramientas. También es necesario contar con personal formado que pueda utilizarlas con eficacia. Por desgracia, el número de profesionales de TI, en particular de expertos en ciberseguridad, no crece a un ritmo suficiente para seguir el de los avances tecnológicos.

En este contexto, surge la pregunta: ¿cómo podemos garantizar la seguridad de nuestros sistemas frente a una situación tan compleja?

1.1 Motivación

La motivación para desarrollar esta herramienta nace de la necesidad de responder a la pregunta formulada anteriormente. Para poder garantizar la seguridad de los sistemas, entre otros, es necesario disponer de mejores herramientas en los diversos campos que nos interconectan, para ello, en este trabajo nos hemos centrado en uno en concreto: el universo de Android.

Nos plantemos el desarrollo de técnicas para analizar aplicaciones Android, que si bien no solucionan el problema del personal, con un diseño óptimo, pueden contribuir a reducir la necesidad de este y garantizar una mayor seguridad en nuestros sistemas.

Por otra parte, el aprendizaje automático se ha hecho cada vez más popular porque permite a los ordenadores aprender de los datos y hacer predicciones y tomar decisiones basadas en ellos con resultados sorprendentes. Estos resultados han sido especialmente sorprendentes en los últimos años en el campo del procesado de secuencias, como el procesamiento de imágenes, del habla o el procesamiento del lenguaje natural. El evento que ha causado estos saltos en nivel de rendimiento, fue la publicación de una novedosa arquitectura: el Transformer [1]. Es gracias a esta, que se han logrado estos avances, que en ocasiones logran resultados prácticamente humanos, que resulta especialmente motivante tratar de aplicar su uso a diversos campos, entre ellos el de la ciberseguridad.

No obstante, no fue hasta cursar la asignatura de Machine Learning en la escuela que encontré interés en este campo, viendo con mis propios ojos el alcance de las aplicaciones de esta tecnología: un futuro mejor es posible.

Es en este marco, en el que existía la posibilidad de fusionar mis dos intereses principales dentro de la informática: la inteligencia artificial y la ciberseguridad. Además, mis conocimientos en Java facilitaban la entrada en Android, un campo interesante, amplio y con gran pronóstico de futuro.

En este trabajo se propone desarrollar una herramienta de análisis de aplicaciones Android que se base en una pipeline compuesta por tres etapas:

1. Decompilación de la aplicación.
2. Extracción de los métodos y elaboración de un resumen de estos con un modelo Transformer para código fuente pre-entrenado.
3. Agrupación por clases de estos métodos resumidos y a su vez resumen de estas agrupaciones mediante un modelo Transformer de procesado de lenguaje natural pre-entrenado.

1.2 Objetivos

El objetivo de esta tesis es desarrollar una herramienta para el análisis de aplicaciones Android utilizando la arquitectura Transformer para el resumen de código. Esta herramienta será capaz de descompilar un archivo APK y extraer los métodos Java que contiene. A continuación, estos métodos se resumirán uno a uno utilizando un Transformer para código. Posteriormente, los resúmenes producidos serán resumidos a su vez, habiéndose agrupado por clases utilizando el modelo T5, lo que permitirá un análisis y una comprensión más eficientes del código.

El objetivo general de este proyecto es crear una herramienta que pueda ayudar en el análisis de aplicaciones Android resumiendo automáticamente su código utilizando modelos Transformer. Esto permitirá a los investigadores y desarrolladores comprender más fácilmente la funcionalidad y el comportamiento de la aplicación, e identificar potencialmente cualquier posible problema o vulnerabilidad. El uso de la arquitectura Transformer para resumir el código permite obtener resúmenes más precisos y completos, en comparación con los métodos tradicionales.

Para lograr este propósito, será necesario completar una serie de objetivos más específicos:

- Investigar y revisar las herramientas y técnicas existentes para descompilar aplicaciones Android y extraer métodos Java.
- Desarrollar e implementar un proceso para descompilar un archivo APK y extraer los métodos Java que contiene.
- Analizar la literatura disponible relacionada con Transformers para código fuente y en concreto para resumir código fuente.
- Investigar los modelos Transformer para resumen de texto que mejor se adapten al problema.
- Desarrollar una pipeline que incorpore estos dos modelos adaptando las distitnas entradas de cada uno
- Generar una interfaz de usuario para mejorar su usabilidad y funcionalidad, haciéndolo más accesible y útil para los investigadores y desarrolladores que trabajan con código de aplicaciones Android.

Investigar y revisar las herramientas y técnicas existentes para descompilar aplicaciones Android y extraer métodos Java.

1.3 Estructura de la memoria

El documento está estructurado en 6 partes, descritas a continuación:

En la primera de ellas, capítulo 1 se contextualiza la herramienta y se expone la motivación y objetivos para el desarrollo de esta.

En el capítulo 2, se profundiza en detalle en las distintas tecnologías que abarca este trabajo, desde aquellas con las que no interactúa directamente como podría ser el campo de trabajo ,Android, como aquellas con las que si tiene contacto directo, como los decompiladores o los distintos modelos Transformer.

Por otra parte, en el capítulo 3, se desarrolla la metodología seguida para elaborar la herramienta, exponiendo los pasos seguidos para elaborar la herramienta correctamente así como explicar su flujo de funcionamiento.

A continuación, en el capitulo 4, se muestra la ejecución de nuestra herramienta sobre distintos tipos de muestras de aplicaciones, los resultados obtenidos, y las limitaciones encontradas, algunas de ellas superadas y otras no.

Más adelante, en el capitulo 5 se comenta el impacto social de la herramienta, comentando tanto los aspectos positivos como los negativos.

Finalmente, en el capítulo 6, se detallan las conclusiones a partir del desarrollo de la herramienta y del análisis de diversas muestras. A su vez se plantean futuras líneas de desarrollo del proyecto.

2. Estado del arte

2.1 Android

¿Qué es Android?

Android es un sistema operativo móvil desarrollado por Google. Se basa en el núcleo de Linux y está diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes y tabletas. Android es de código abierto, lo que significa que cualquiera puede utilizarlo, modificarlo y distribuirlo libremente.

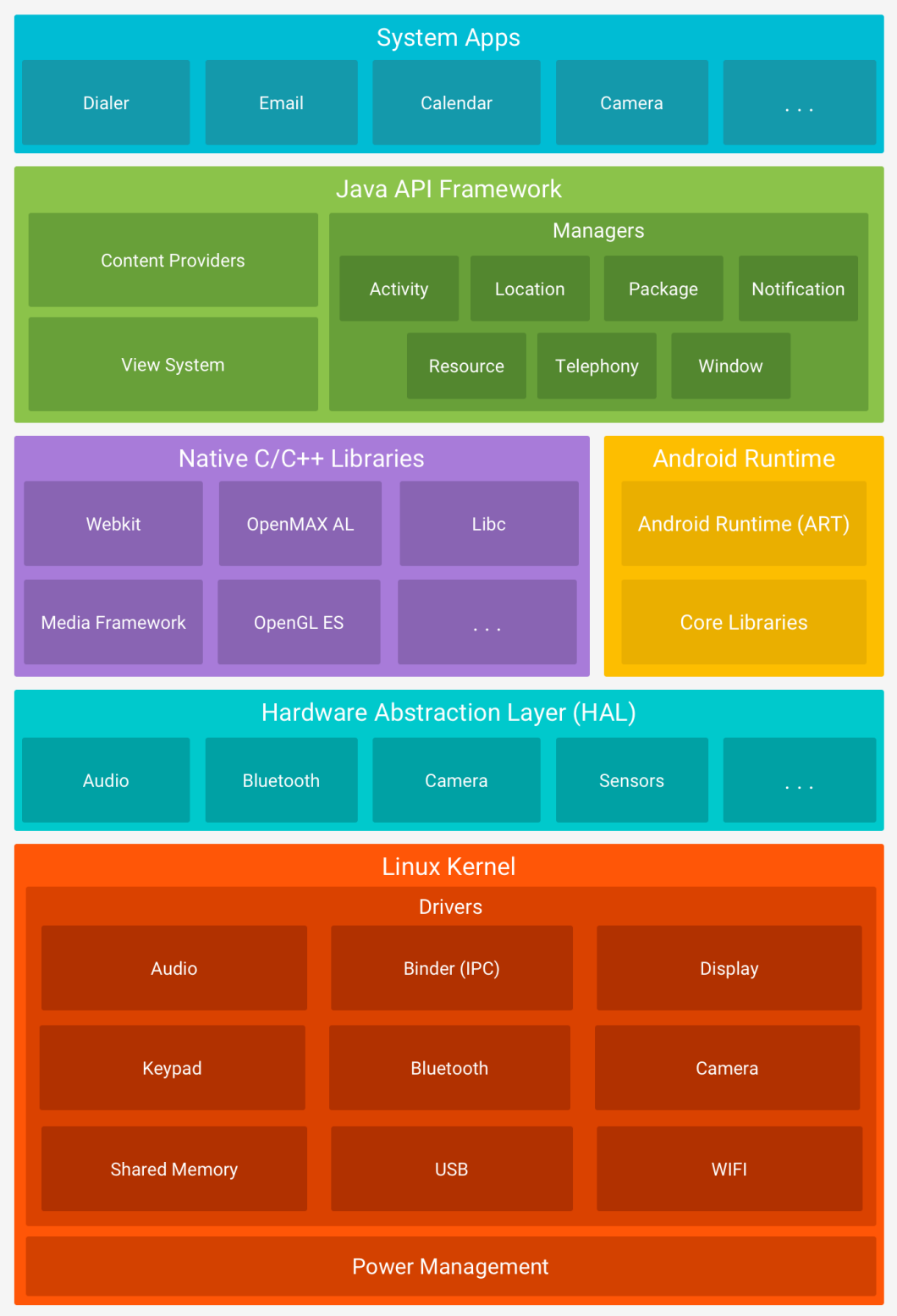
Android incluye una serie de aplicaciones y funciones preinstaladas, como un navegador web, un cliente de correo electrónico y una aplicación de mapas. También cuenta con un rico ecosistema de aplicaciones de terceros, disponibles para su descarga en Google Play Store. Estas aplicaciones pueden desarrollarse utilizando el SDK de Android, un conjunto de herramientas y API para crear aplicaciones nativas de Android.

Además de su amplio uso y popularidad, Android es conocido por su naturaleza altamente personalizable y flexible. Los usuarios pueden personalizar la apariencia y el comportamiento de sus dispositivos cambiando la pantalla de inicio, instalando lanzadores personalizados y utilizando diversos widgets y fondos de pantalla en vivo. Esto permite una experiencia personalizada y única en cada dispositivo.

Realmente, Android es una pila de software de código abierto diseñado principalmente para dispositivos móviles, que en muchas ocasiones, por simplicidad, es tratado de sistema operativo. Una pila de software es un conjunto de componentes independientes de software agrupados en forma de paquete que funcionan conjuntamente para satisfacer determinados servicios. En el caso de Android, puede ser dividida en capas, en función de la ”proximidad” al hardware que lo soporta. (ver Figura 2.1)

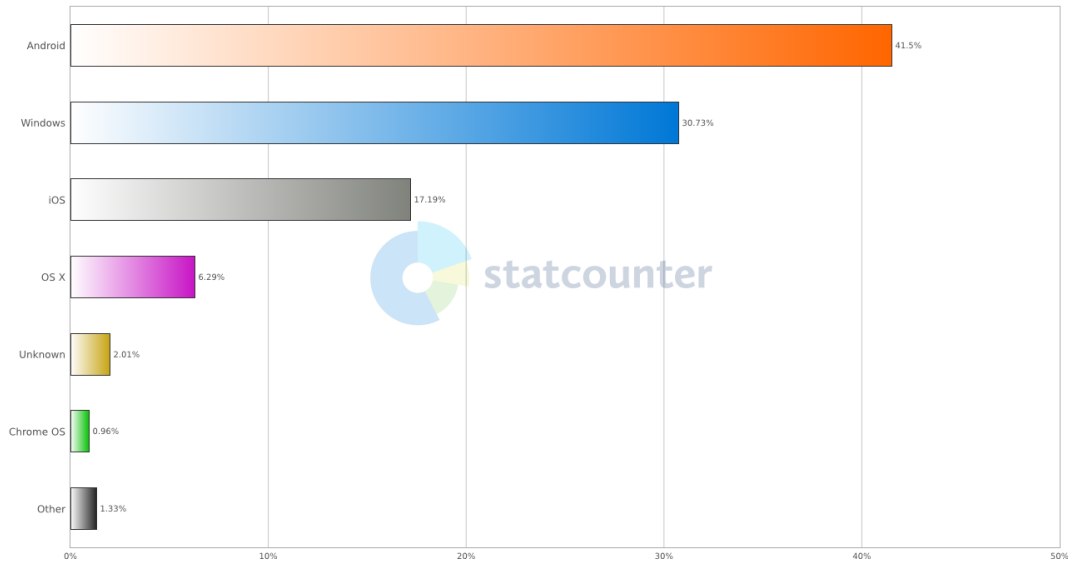
La primera de ellas, la más cercana al hardware, incluye un sistema operativo basado en una adaptación del kernel de Linux, cuya versión depende de cada dispositivo en concreto. La segunda capa está formada por un conjunto de middleware, es decir, software encargado de la comunicación entre el sistema operativo y las aplicaciones que se ejecutan en él. Agrupa, desde una capa de abstracción de hardware HAL, pasando por distintas librerías nativas escritas en C/C++, que se encuentran, al mismo nivel que un entorno de ejecución de aplicaciones conocido como ART. La siguiente capa está compuesta por un framework de aplicación, que proporciona las clases utilizadas para crear aplicaciones Android, así como abstracción para el acceso al hardware y la gestión de la interfaz de usuario y los recursos de la aplicación, a través de un rico conjunto de APIs. También incluye un conjunto de aplicaciones del sistema. Gracias a esta estructura, los desarrolladores pueden diseñar aplicaciones de manera sencilla y universal que interactúen a su gusto con los dispositivos donde se ejecutan, habitualmente móviles y sistemas embebidos.

Figura 2.1: Pila de software de Android [2]



Es importante destacar que este sistema operativo es, según distintas fuentes, el más utilizado en el último año en lo que a uso web se refiere, esto es, excluyendo videoconsolas y sistemas embebidos, ver 2.2. Además, es el sistema operativo móvil más popular del mundo, con una cuota de mercado global superior al 70% [3]. Se utiliza en una gran variedad de dispositivos de distintos fabricantes, como Samsung, Huawei, Xiaomi o Motorola.

Figura 2.2: Cuota de mercado de los sistemas operativos en el mundo en el último año [4]



¿Qué había antes?

El desarrollo de aplicaciones en dispositivos móviles ha ido cambiando a lo largo de las últimas décadas. En sus primeros momentos, se utilizaban lenguajes de bajo nivel (C, C++), muy próximos a la máquina, bajo los cuales era imprescindible tener en cuenta el hardware para el que se diseñaban el software. Esto suponía enormes restricciones de cara a la universalidad del uso del dispositivo móvil, corriente contraria a la propia naturaleza de internet y la tecnología asociada a este.

Años más tarde se produjo un gran salto en el desarrollo de dispositivos móviles y embebidos con la aparición del lenguaje Java y especialmente los MIDlets. Los MIDlets son aplicaciones o programas diseñados en Java para ejecutarse en una Máquina Virtual de Java (JVM son sus siglas en inglés), la cual proporciona una capa de abstracción frente al hardware, superando así, la barrera a la universalidad que otros lenguajes sufrían y permitiendo en un futuro, la aparición de Android.[5]

The Android language

Android soporta una gran diversidad de lenguajes para el desarrollo y ejecución de sus aplicaciones, con la especial restricción de que deben ser interpretables por la JVM, lenguajes como Java o Koltin no tienen ningún problema ya que fueron diseñados con este propósito, no obstante, otros como Go, JavaScript, C++, C o ensamblador, necesitan de herramientas especiales que limitan su uso. [6]

Java

En el momento de la concepción del sistema operativo, Java fue pensado como la columna vertebral de Android, y desde su creación hasta 2019, Java ha sido el lenguaje oficial de desarrollo, por lo que merece un apartado en el que se describan sus puntos más interesantes.

Java es un lenguaje de programación creado en 1995 por la empresa Sun Microsystems. Se trata de un lenguaje de propósito general, orientado a objetos, que utiliza clases para sustentar su funcionamiento. A pesar de tener una sintaxis similar a otros lenguajes como C o C++, Java se caracteriza, al contrario que estos, por ser un lenguaje robusto, simple, dinámico y multiplataforma. Esta última fue la principal característica que popularizó su uso, al estar diseñado para satisfacer la funcionalidad WORA (Escríbelo una vez, ejecútalo en cualquier lugar), muchos proveedores de servicios web basados en arquitecturas cliente-servidor, vieron esta herramienta como una forma de terminar con una de las limitaciones que tenía la web en aquel momento, facilitando y extendiendo las posibilidades del desarrollo y uso de múltiples servicios web. Como resultado, Java se convirtió en uno de los lenguajes de programación más usados en los últimos 20 años. [7]

JVM

Para poder abstraer el código de Java o los lenguajes de programación que se compilan en bytecode Java, del hardware en el que se ejecuta/ejecutará los dispositivos se apoyan sobre una máquina virtual denominada JVM.

La máquina virtual de java o JVM es una máquina de computación abstracta basada en una estructura de pila, que como cualquier maquina real, tiene un conjunto de instrucciones predefinido y es capaz de manipular distintas áreas de memoria en tiempo de ejecución, sin tener la necesidad de asumir ninguna tecnología implementada, hardware o sistema operativo allí donde se ejecute. Esta máquina no tiene dependencia directa con el lenguaje Java, sino con un formato particular de fichero binario, el fichero “class”. Este fichero contiene los denominados bytecodes, que no son más que instrucciones de la máquina virtual de java, una tabla de símbolos y otra información

complementaria. Como se ha comentado anteriormente, al no existir una dependencia estrecha con ningún lenguaje, cualquier lenguaje que se adapte (se pueda compilar) a las restricciones sintácticas y estructurales de la JVM, podrá aprovecharse de esta.

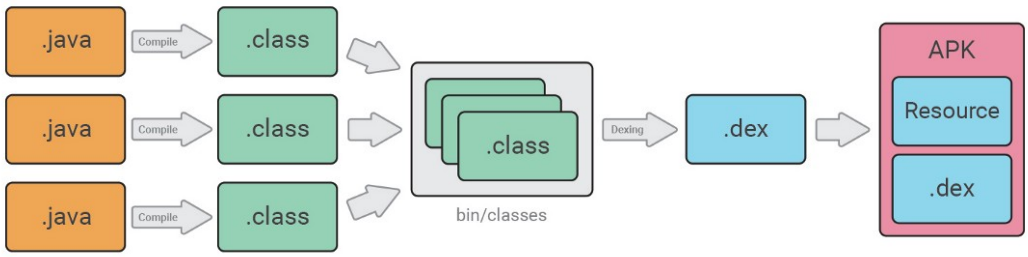
A pesar de todas las ventajas proporcionadas por la JVM, existía una importante restricción y es que estaba sujeta a normativa de propiedad intelectual. No fue hasta 2006 cuando Sun, la empresa propietaria en ese momento de la propiedad intelectual asociada a la tecnología de Java abrió al público su JVM haciéndola gratuita y open source, cuando empezaron a surgir múltiples tecnologías que aprovechaban las ventajas que ofrecían como Android.

Por otra parte, el IDE oficial de desarrollo en Android, Android Studio, introdujo en 2019 su versión 3.0, donde Koltin reemplazaba a Java como el lenguaje preferido para el desarrollo de aplicaciones en el entorno. Hay que aclarar que Koltin es un lenguaje construido sobre la JVM, que es interoperable con los existentes lenguajes en Android y con ART, permitiendo así usar tanto Java como Koltin en las aplicaciones [6].

De la DVM al ART

Durante las primeras versiones de Android, las aplicaciones eran ejecutadas en la Dalvik virtual machine (DVM) y no en una JVM. Esta es máquina virtual utiliza una arquitectura basada en registros, lo cual, proporciona teóricamente ejecuciones más optimizadas para sistemas menos potentes, como lo eran los primeros dispositivos móviles, ya que estas arquitecturas requieren menos instrucciones de la máquina virtual, pero más complejas. Además, en lugar de utilizar múltiples archivos class agrupados en un jar; la DVM combina en uno o más archivos todos los ficheros class, ficheros dex (con diferente estructura e instrucciones de máquina), los cuales se comprimen posteriormente en un fichero ejecutable APK. De esta forma se consigue optimizar espacio de almacenamiento ya que la existencia strings duplicados y otras constantes repetidas en múltiples ficheros class se reduce notablemente.

Figura 2.3: Proceso seguido por los ficheros de código fuente de una APK [8].



No obstante, con la introducción de Android 5.0 ”Lollipop” en 2014, la DVM fue completamente sustituida por una nueva tecnología, el Android Run Time (ART). El ART es un entorno de ejecución de aplicaciones que introduce el concepto de compilación Ahead-of-time (AOT). Este paradigma está basado en el salvado de la compilación total de las aplicaciones o de partes de ella. De esta forma optimiza la ejecución de estas, reduciendo su tiempo de interpretación y compilación cada vez que son ejecutadas, así como el consumo de batería. No obstante, estas mejoras suponen un coste de almacenamiento que han sido aceptables gracias a los avances en memoria disponible de los dispositivos de hoy en día. Para mantener la retrocompatibilidad, el ART utiliza el mismo bytecode que la DVM apoyándose en los ficheros .dek y .apk [9][10].

El paquete android: APK

Un archivo de paquete de aplicaciones Android (APK) es el formato de archivo de paquete utilizado por el sistema operativo Android para la distribución e instalación de aplicaciones móviles. Es similar a un archivo .exe en Windows o a un archivo .sh en Linux.

Un archivo APK contiene todos los archivos necesarios para ejecutar una aplicación Android, incluidos el código, los recursos y los metadatos de la aplicación. La estructura de un archivo APK está diseñada para ser eficiente y compacta, de modo que pueda transferirse e instalarse fácilmente en dispositivos Android. Todo esto es posible ya que realmente, un archivo .apk es en realidad un paquete de compresión zip, que puede descomprimirse fácilmente con herramientas de descompresión. Sus contenidos son los siguientes [11] [6]:

- En primer lugar, uno de los archivos más importantes dentro de un fichero APK, junto con el código de la aplicación, es el archivo de manifiesto de Android. Este archivo se llama ”AndroidManifest.xml” y normalmente se encuentra en la raíz del archivo APK. Se trata de un archivo XML que contiene metadatos sobre la aplicación, como su nombre, versión y permisos necesarios. Destacar que también especifica los permisos necesarios de la aplicación, como la capacidad de acceder a la cámara o a la red del dispositivo, información especialmente interesante cuando se analizan las aplicaciones Android. El sistema operativo Android utiliza este archivo para gestionar e instalar la aplicación en un dispositivo. Proporciona información importante que el sistema utiliza para determinar cómo debe instalarse, ejecutarse y actualizarse la aplicación.
- Un directorio META-INF, que contiene metadatos de la aplicación, como sus firmas, utilizadas para garantizar la integridad del paquete .apk y la seguridad del sistema. Cuando se compila la aplicación por primera vez, se computa la firma de todos los archivos que se

van a empaquetar. Las firmas obtenidas se almacenan en el directorio META-INF. De esta forma, al instalar un paquete .apk en el teléfono, el gestor de aplicaciones aplicará el mismo procedimiento anterior. Si el resultado es diferente al del directorio META-INF, el sistema no instalará la aplicación. Esto permite asegurar que los archivos contenidos en el paquete .apk no serán reemplazados arbitrariamente. Por ejemplo, es prácticamente imposible reemplazar cualquier imagen, fragmento de código o datos de copyright en el paquete .apk descomprimiendo directamente el archivo, reemplazando dicho contenido y volviéndolo a empaquetar. Por lo tanto, esto puede proteger el sistema de alteraciones no deseadas de las aplicaciones, aumentando la seguridad del sistema.

- Otro directorio, assets, que contiene archivos que puede necesitar la aplicación en tiempo de ejecución, estos archivos pueden ser desde elementos multimedia utilizados por la aplicación como imágenes, audio o vídeos; archivos de datos que contengan información de la aplicación como ficheros JSON e incluso archivos de configuración o scripts que utiliza .

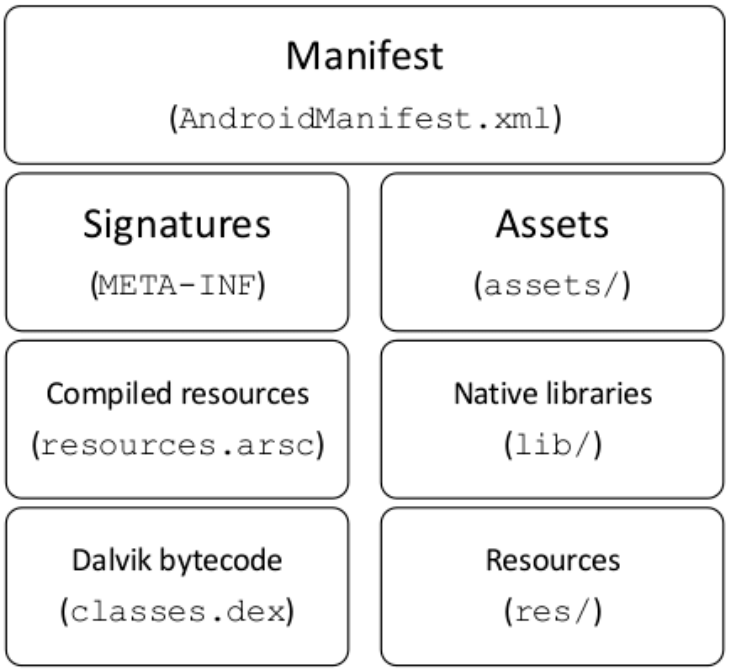
- Un archivo denominado resources.arsc, que contiene todos los recursos que ya han sido compilados por los desarrolladores. Sirve para asegurar que estos recursos estén disponibles cuando la aplicación esté activa. Los archivos XML binarios, por ejemplo, se guardan en este archivo como recursos precompilados.

- Un directorio, lib, el cual está compuesto de bibliotecas nativas que utiliza la aplicación. Una biblioteca nativa es una biblioteca de código compilado específica para una arquitectura de CPU concreta, como ARM o x86. Las apps de Android suelen utilizar bibliotecas nativas para acceder a funciones del sistema o realizar operaciones de cálculo intensivo, como el procesamiento de imágenes o audio.

- Uno o varios ficheros classes.dex, los cuales contienen el bytecode Dalvik de la aplicación, es decir, el código de la aplicación compilado en un formato binario que puede ser ejecutado por un dispositivo Android. Cuando la aplicación es demasiado grande, se crearán diversos ficheros classesX.dex.

- El directorio res, el cual contiene ficheros (o recursos), que la aplicación puede necesitar en tiempo de ejecución. Su contenido puede ser el mismo que el de assets, con la diferencia de que cada fichero del directorio res debe tener un ID precompilado, de forma que puede ser accesible desde la aplicación más fácilmente. La ventaja de assets es que nos permite más libertad a la hora de alojar recursos en la estructura y formato que queramos.

Figura 2.4: Estructura de ficheros de una APK descomprimida [12]



2.2 Malware en Android

El malware para Android suele distribuirse en forma de aplicaciones maliciosas, diseñadas para realizar acciones dañinas en un dispositivo Android. Estas acciones pueden ir desde el robo de información sensible hasta el secuestro de los recursos del dispositivo con fines dañinos. Para protegerse contra estas amenazas, es necesario identificar y analizar las aplicaciones maliciosas para comprender su comportamiento y sus posibles efectos.

El análisis de malware es una habilidad crítica para comprender y protegerse contra la amenaza del software malicioso en los dispositivos. En esta sección, exploraremos los conceptos y técnicas fundamentales del análisis de malware para Android, así como algunas de las herramientas y tecnologías que se utilizan habitualmente en este campo.

Existen dos enfoques a la hora de realizar análisis de malware en cualquier campo, y como no podía ser de otra forma, Android los comparte: el análisis dinámico y el análisis estático, los cuales desarrollaremos a continuación:

Análisis estático en Android

El análisis estático es una técnica común utilizada en el análisis de malware para Android. Consiste en examinar una aplicación sin ejecutar nada de código, con el fin de identificar cualquier posible elemento malicioso. Existen autores que separan el análisis estático de la ingeniería inversa, no obstante en este trabajo los agruparemos por simplicidad. [13]

Para analizar el comportamiento de una determinada muestra se aplican estrategias como: inspeccionar el paquete apk o acceder a los distintos archivos contenidos en el paquete, entre ellos el Android Manifest. Esto permite reunir un conjunto de características relevantes y útiles, como una lista de las llamadas a la API invocadas a lo largo del código o el conjunto de permisos de Android necesarios para desplegar toda la funcionalidad de la muestra. [14]

Otra estrategia habitual de análisis estático de malware consiste en aplicar ingeniería inversa al código de una aplicación para comprender su funcionalidad e identificar cualquier comportamiento malicioso. Para ello se utilizan diversas herramientas y técnicas, como decompiladores, desensambladores y otras herramientas. Al examinar detenidamente el código de la aplicación, es posible identificar indicadores clave de comportamiento malicioso, como la modificación o acceso a elementos sensibles o la presencia de firmas de malware conocidas en forma de strings o hashes. [13]

Una desventaja potencial del análisis estático es que puede no ser capaz de capturar completamente el comportamiento de la aplicación. Dado que el análisis estático consiste en examinar el código de la aplicación sin ejecutarlo, es posible que no pueda detectar todas las acciones maliciosas o efectos potenciales. Por ejemplo, una aplicación puede utilizar técnicas de ofuscación para ocultar su verdadero comportamiento, o puede realizar ciertas acciones sólo cuando se cumplen determinadas condiciones. En tales casos, el análisis estático puede no ser capaz de identificar el comportamiento malicioso de la aplicación.

En general, aunque el análisis estático puede proporcionar información valiosa sobre el código y el comportamiento potencial de una aplicación, no está exento de limitaciones. En algunos casos, puede ser necesario combinar el análisis estático con otras técnicas para analizar a fondo y con precisión el comportamiento de una aplicación.

Análisis dinámico en Android

Por otra parte, otro elemento clave del análisis de malware es el uso del análisis dinámico, que consiste en ejecutar la aplicación en un dispositivo o emulador y observar su comportamiento en tiempo real. El análisis dinámico puede proporcionar información valiosa sobre el comportamien-

to de una aplicación, como los datos a los que accede o las acciones que realiza. Esta información puede utilizarse para identificar comportamientos maliciosos y confirmar los resultados del análisis estático. [15]

Una desventaja del análisis dinámico es que consume mucho tiempo y recursos. Dado que implica ejecutar la aplicación en un dispositivo o emulador y observar su comportamiento en tiempo real, el análisis dinámico puede llevar más tiempo que otras formas de análisis, como el análisis estático. Además, puede requerir el uso de herramientas y tecnologías especializadas, como herramientas forenses o entornos sandbox, que pueden aumentar el coste y la complejidad del proceso de análisis.

Otra desventaja del análisis dinámico es que puede no ser capaz de capturar completamente el comportamiento de la aplicación. Dado que el análisis dinámico sólo observa el comportamiento de la aplicación mientras se ejecuta en un entorno concreto, es posible que no pueda detectar todas las posibles acciones maliciosas o efectos potenciales. Por ejemplo, una aplicación puede realizar acciones maliciosas sólo cuando se cumplen ciertas condiciones como tener conexión inalámbrica a internet o realizar una acción específica como utilizar otra aplicación. En tales casos, el análisis dinámico puede no ser capaz de identificar el comportamiento malicioso de la aplicación.

Además de estas técnicas, también existen muchas herramientas y tecnologías especializadas que se utilizan habitualmente en el análisis de malware para Android. Entre ellas se incluyen herramientas forenses para extraer y analizar datos de dispositivos Android, herramientas de análisis de red para supervisar el tráfico de red y entornos sandbox para ejecutar y observar aplicaciones maliciosas de forma segura.

En general, el análisis de malware para Android es un campo complejo y en constante evolución. Comprendiendo los conceptos y técnicas fundamentales del análisis de malware, así como las herramientas y tecnologías utilizadas en este campo, es posible desarrollar las habilidades y conocimientos necesarios para identificar y protegerse contra el software malicioso en dispositivos Android.

Decompiladores

Los decompiladores son herramientas que se utilizan para realizar ingeniería inversa en distintos tipos de programas, entre ellos, aplicaciones Android, convirtiendo su código compilado (es decir, sus archivos DEX) en su código fuente original. Los analistas de Android suelen utilizar los decompiladores para comprender el comportamiento y la funcionalidad de otras aplicaciones, así

como para identificar posibles vulnerabilidades de seguridad o comportamientos maliciosos en aplicaciones de Android.

Existen diversos decompiladores diferentes que están disponibles gratuitamente para su uso con aplicaciones de Android. Algunos ejemplos populares pueden ser jd-gui [16], que es un descompilador Java independiente, Apktool [17], que es una herramienta de línea de comandos que se puede utilizar para decompilar y reconstruir aplicaciones Android, Dex2jar [18] que tiene como objetivo convertir archivos Dalvik bytecode en archivos compilados Java con extensión jar. Este último programa se suele combinar con otros como CFR [19] o procyon [20] que son decompiladores de Java, que a partir de Java bytecode, como es el caso de los archivos .jar, obtiene el código fuente asociado. Existen herramientas como lazyX [21] que combinan dex2jar con CFR o procyon automatizando un pipeline de decompilación de ficheros DEX a código fuente Java. Otra alternativa es Jadx [22], el cual realiza en un paso la decompilación de DEX a código Java. [14]

Smali

Smali/baksmali es un ensamblador/desensamblador para el formato dex utilizado por las aplicaciones Android. Su propósito es ensamblar o desensamblar los ficheros dex (las aplicaciones Android) a código smali y viceversa, este soporta toda la funcionalidad del formato dex (anotaciones, información de depuración, información de línea, etc.) [23].

Los nombres ”smali” y ”baksmali” son los equivalentes islandeses de ”ensamblador” y ”desensamblador” respectivamente. ¿Por qué en islandés? Porque dalvik debe su nombre a un pueblo pesquero islandés [23].

El propósito de una herramienta como esta, es obtener una representación de bajo nivel del bytecode contenido en los ficheros .dex, que nos permita trabajar con el código de una manera mucho más cómoda y cercana a la máquina, a la vez que legible, sin tener que recurrir a lenguajes de más alto nivel como Java. Multitud de decompiladores como dex2jar o jadx utilizan smali como puente para decompilar aplicaciones. Un ejemplo de código en smali es el proporcionado en 2.1

Listado 2.1: Ejemplo de una clase con un método que imprime Hola mundo en Smali

```
.class public LHelloWorld;

.super Ljava/lang/Object;
```

```
.method public static main([Ljava/lang/String;)V
    .registers 2

    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

    const-string  v1, "Hola mundo!"

    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
       String;)V

    return-void
.end method
```

2.3 ¿Es la inteligencia artificial el futuro?

Como se ha comentado a lo largo del trabajo, el riesgo y el impacto de los ataques de ciberseguridad está aumentando con el paso de los años y lo hace a un ritmo preocupante gracias al aumento en complejidad y frecuencia de ellos.

Algunos profesionales del sector proponen la IA como solución a estos problemas, prometiendo una disminución del impacto de los ataques así como una optimización del capital humano en los equipos de ciberseguridad. Estas optimizaciones se podrían agrupar en tres puntos principales que a su vez son puntos fundamentales de la ciberseguridad:

El primero de ellos es la robustez, robustez entendida como la capacidad de un sistema de comportarse como se espera, incluso cuando se le proporcionan entradas incorrectas. Esta propiedad es uno de los pilares del software testing y se han desarrollado herramientas para automatizar el testeo de programas, así como para solucionar algunos de sus bugs utilizando IA. [24][25]. A su vez está estrechamente relacionada con todos los elementos la triada CIA (confidencialidad, integridad y disponibilidad).

El segundo de ellos es la capacidad de respuesta: la inteligencia artificial nos proporciona herramientas para defendernos de ataques de manera automática, estas soluciones suelen basarse en la aplicación de filtros en múltiples campos, desde el control de tráfico de entrada, el análisis de código fuente en búsqueda de vulnerabilidades, la búsqueda de intrusos en nuestro sistema...[26],

[27], [28]. Además, una de las ventajas que proporciona la inteligencia artificial en este campo es la capacidad de adaptación, estas herramientas son capaces de aprender de los ataques recibidos y ser más seguros con cada uno de ellos, desarrollando estrategias de defensa basadas en engaño como pueden ser los honeypots. [29]

En tercer lugar, lo que podríamos denominar resiliencia; la capacidad de un sistema de recuperarse después de un ataque y ofrecer una estrategia para paliar las futuras ofensivas. Previo a profundizar en la materia, debemos definir el concepto de "Análisis forense digital". Consisten en el uso de métodos científicamente derivados y probados para la preservación, recogida, validación, identificación, análisis, interpretación, documentación y presentación de pruebas digitales derivados de un ataque [30]. Es comúnmente conocido que este análisis es una tarea tediosa y que comprende enormes cantidades de tiempo debido a la cantidad de datos a analizar, desbordando así a los departamentos dedicados a este campo. Sin embargo, el desarrollo de mecanismos de procesado automático de datos mediante inteligencia artificial reduce sustancialmente la cantidad de tiempo y trabajo empleados para entre otros, encontrar los servicios realmente afectados por los ataques y por tanto, mejorando nuestra capacidad para recuperarnos de estos. [31]

Debido a su impacto en estos tres puntos definidos, puntos que son clave para la ciberseguridad, podríamos considerar que la IA ofrece ventajas a nivel técnico y a nivel estratégico. Técnicamente, la IA puede mejorar la seguridad de los sistemas y reducir su vulnerabilidad a amenazas y en consecuencia a ataques. Estratégicamente, la IA puede alterar la dinámica típica en el campo según la cual, los ataques prevalecen sobre las defensas. Por ejemplo, el uso de la IA para mejorar la robustez de los sistemas puede tener un efecto en cadena y disminuir el impacto de los ataques de día cero, que son aquellos que aprovechan las vulnerabilidades de un sistema que no han sido declaradas oficialmente y son desconocidas para los fabricantes/proveedores o no tienen un parche asociado que las resuelva, por lo que pueden ser explotadas por los atacantes.

2.4 Machine learning

Este trabajo se cimienta sobre una arquitectura situada dentro del campo conocido como machine learning, por lo tanto, es necesario presentarlo.

El Machine Learning (ML) o aprendizaje automático, es una rama de la inteligencia artificial cuyo objetivo principal es tratar de emular una de las capacidades que caracterizan a los seres humanos: el aprendizaje automático del entorno. Con aprendizaje se hace referencia a la capacidad de mejorar el desempeño de una tarea por parte de un sistema a través del suministro de datos. Un ejemplo típico podría ser un programa de ordenador que conozca las reglas básicas del ajedrez y

mejore su rendimiento mediante la experiencia obtenida de jugar partidas contra el mismo. Podemos definir tres características clave para definir un problema de aprendizaje: el tipo de tarea, una métrica para medir el rendimiento a mejorar y una fuente de experiencia. [32].

Tipos de algoritmos

Existen formas distintas de que las máquinas aprendan, es por esto que se pueden dividir los enfoques del aprendizaje automático en tres grandes categorías, que corresponden a paradigmas de aprendizaje, según la naturaleza del problema que tratan de resolver y del sistema de retroalimentación usado para aprender:

- El aprendizaje supervisado, en el cual, las máquinas necesitan que el conjunto de datos proporcionados para aprender esté etiquetado, de forma que sea posible inferir una regla que relacione entradas con salidas. La estrategia seguida consiste en dividir el conjunto de datos de forma que la máquina reciba un subconjunto de estos con los que aprender y utilice otro subconjunto para realizar predicciones y de esta manera podamos evaluar su rendimiento. Las tareas más comunes de aplicación de estos algoritmos son la clasificación de entradas y la regresión. Un ejemplo típico de este tipo de aprendizaje puede ser clasificar imágenes que contienen dígitos en función del dígito contenido. [33]
- El aprendizaje no supervisado, en el cual, las maquinas reciben datos sin etiquetar, de forma que tienen que aprender a realizar observaciones reconociendo patrones. Uno de los problemas de estos algoritmos es la evaluación de su rendimiento. Ejemplos de algoritmos de aprendizaje no supervisado pueden ser el clustering o la reducción de dimensiones. [34]
- El aprendizaje reforzado, en el cual, de la misma forma que el aprendizaje no supervisado, no necesita datos etiquetados. Para recoger información, la máquina interactúa activamente con el entorno para recibir recompensas en función de su comportamiento, de esta forma, se solapan la fase de entrenamiento y testeo del modelo. El objetivo del sistema es maximizar su recompensa a lo largo de un curso de acciones e iteraciones con el entorno, aprendiendo a base de prueba y error. Es un ejemplo de algoritmos usados para que las máquinas aprendan a jugar a juegos como el ajedrez. [34], [35]

Deep Learning

Entrando en un nivel más fino de detalle, este trabajo hace uso de una arquitectura basada en una familia de algoritmos particular dentro del campo del aprendizaje automático: el Deep Learning

(DL) o aprendizaje profundo. [36]

Los enfoques tradicionales del Machine Learning, giran en torno al uso de algoritmos que trabajan con un conjunto de características cuya importancia e integrantes son definidos por la persona que decide utilizarlos. Estas son extraídas en un proceso de ingeniería sobre los datos naturales, sin procesar (raw data en inglés), conocido por diversos nombres, entre ellos ingeniería de características (feature extraction en inglés). Este proceso implica obtener representaciones del aprendizaje y el rendimiento de los algoritmos del aprendizaje automático están estrechamente relacionados con la calidad de estas representaciones.

Como hemos repasado, estos métodos se caracterizan por tener limitaciones a la hora de tratar con datos en su formato natural, sin preprocesar o procesar, y es en este contexto, en el que surge el Deep Learning. En este subcampo, es el propio algoritmo el que, mediante una serie de mecanismos, identifica cuales son las características más importantes y se centra en ellas para desempeñar mejor la tarea objetivo. Este enfoque aporta ventajas, ya no solo por reducir el trabajo de los analistas de datos, sino que además, en ocasiones, optimiza sus análisis.

Como se ha mencionado anteriormente, el Deep Learning (DL), facilita o permite el trabajo con un conjunto de datos sobre los que es muy difícil trabajar: los datos sin procesar (o raw data), característicos en campos como el de procesado de imágenes, de lenguaje natural, de procesado de audio o incluso de procesado de código. Los datos sin procesar suponen un especial reto ya que para poder interpretarlos, es necesaria una comprensión casi humana de ellos, resulta que las arquitecturas basadas en DL logran descubrir estructuras intrínsecas a este tipo de datos.

Esta capacidad de trabajar con datos sin procesar supone una de las mayores aportaciones del aprendizaje profundo, logrando interpretaciones de estas características a través de representaciones que se expresan en términos de otras representaciones más simples. Es decir, permite construir conceptos complejos a través de conceptos más simples gracias a las redes de neuronas artificiales.

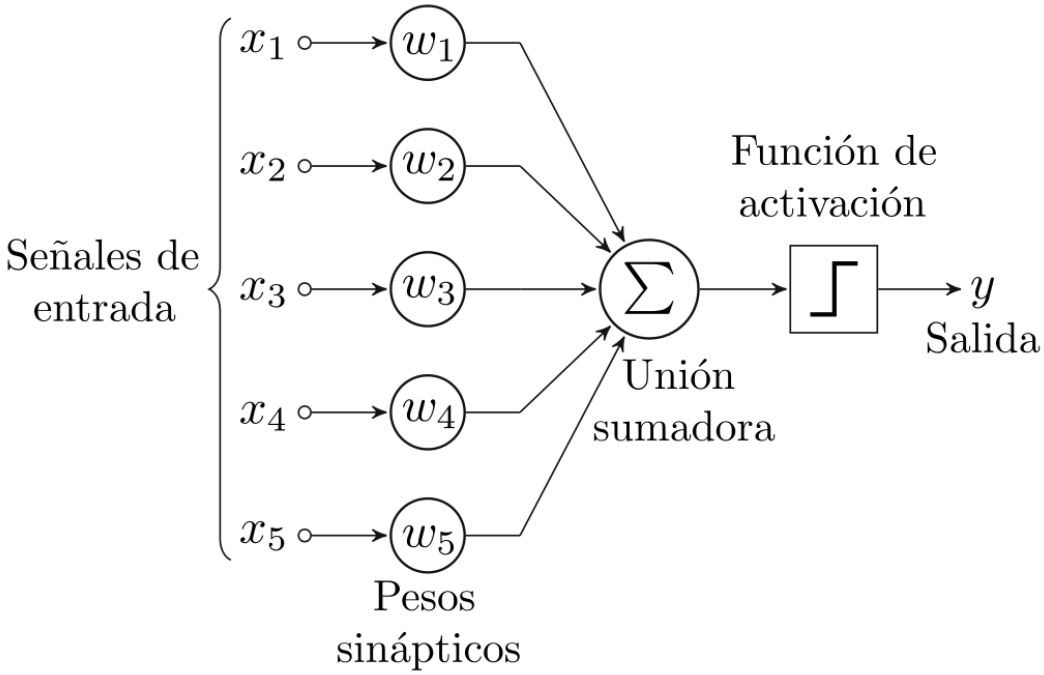
Redes de neuronas

En un intento de emular las capacidades de comprensión humanas, se propuso la utilización de un concepto presente en todo cerebro animal y responsable de los mecanismos de aprendizaje y comprensión de estos: las neuronas. Es a través de la conexión de multitud de unidades simples (neuronas) que nos es posible adquirir y utilizar conocimiento, por lo que se introdujo este concepto a la informática en forma de Redes de neuronas (RRNN) artificiales. [36]

Las redes de neuronas artificiales están compuestas por neuronas artificiales, que son, en grandes

rasgos, funciones matemáticas que trata de emular el comportamiento de las neuronas biológicas. Su comportamiento se basa en el siguiente funcionamiento: reciben una entrada de otra neurona y dependiendo de una función matemática interna, se activa, computando una función interna y transmitiendo datos de salida o permanece desactivada, tratando de emular así el funcionamiento de la sinapsis de las neuronas biológicas. Un ejemplo de neurona simple es el perceptrón [37], el cual, en el momento de su concepción, únicamente podía realizar tareas de clasificación lineal. No obstante, con el paso de los años, se introdujeron conceptos como el perceptrón multicapa o la retro propagación (backpropagation), los cuales permitían trabajar con múltiples capas y funciones de activación más complejas, logrando así, que el diseño de grandes redes compuestas por neuronas más simples como el perceptrón fuesen capaces de resolver problemas no linealmente separables, abriendo así el uso de esta tecnología a múltiples campos. [38]

Figura 2.5: Esquema de un perceptrón con cinco entradas [39].



Transfer learning

Con la introducción de algoritmos en el campo del machine learning que necesitaban disponer de mayores conjuntos de datos de entrenamiento, como el deep learning, y los enormes avances en las capacidades globales de almacenamiento y gestión de datos [40]:la recopilación y utilización de cantidades masivas de datos se estandarizó en la mayoría de campos, dando lugar al surgimiento y la popularización de conceptos como el big data o la minería de datos [41].

En algunos escenarios basados en machine learning, obtener suficientes datos sobre el dominio del problema para entrenar algoritmos puede resultar complejo y costoso, generando así la necesidad de obtener datos del mismo dominio de formas alternativas.

A raíz del contexto y necesidades expuestas anteriormente dan que surge el transfer learning o aprendizaje transferido. Esta técnica se utiliza para mejorar el aprendizaje de un algoritmo en un determinado campo, a partir de la transferencia del conocimiento adquirido en otro campo de un dominio relacionado.

Se pueden trazar paralelismos con la experiencia de aprendizaje de los humanos: considere dos personas que quieren aprender una lengua de procedencia latina como el italiano y solo hablan su lengua materna. La lengua materna de una de ellas es el español, una lengua con la misma procedencia, mientras que la lengua nativa de la otra persona es el inglés, lengua de procedencia sajona. Lo normal es que la persona cuya lengua materna tiene procedencia latina, como el idioma que intenta aprender, tenga mayores facilidades para aprenderlo, ya que en cierta manera, transferirá sus conocimientos anteriores para aprender el italiano. Resumiendo, una persona ha sido capaz de utilizar la información obtenida previamente en una tarea de un dominio concreto y utilizar estos conocimientos de manera beneficiosa a la hora de aprender una tarea nueva de un dominio relacionado. Gracias al big data y data mining, cada vez existen más repositorios públicos con grandes cantidades de datos de múltiples dominios, que pueden ser utilizados para aplicar transfer learning [41].

2.5 Transformers

En esta sección se profundiza en el funcionamiento de la arquitectura sobre la que se basa el trabajo: el modelo Transformer. La introducción de esta arquitectura ha supuesto un salto en multitud de campos diversos, desde el procesado del lenguaje natural, pasando por el procesado de imágenes, vídeos o incluso audio. En nuestro caso, nos interesará el campo del procesado del lenguaje natural y el procesado del lenguaje de máquina.

El modelo fue introducido en 2017 en una publicación académica bajo el título de *Attention is all you need* [1], en el cual, se introduce una perspectiva nueva sobre los enfoques tradicionales existentes en los problemas de transducción y modelado de secuencias, basados en CNNs y RNNs. La arquitectura se cimienta principalmente sobre el concepto de "atención", prescindiendo de las CNNs y RNNs. El artículo se centra en la efectividad de la implementación sobre dos subproblemas concretos dentro de los mencionados anteriormente: el modelado de lenguaje y la traducción automática. [1]

La transducción es un tipo de inferencia o razonamiento, que parte de observaciones concretas y trata de predecir observaciones futuras que son desconocidas. En este escenario, se da un conjunto de entrenamiento (etiquetado) de aprendizaje y un conjunto de test (no etiquetado). La idea de la transducción es realizar predicciones sólo para los puntos de test. Esto contrasta con el aprendizaje inductivo, en el que el objetivo es obtener una función de predicción definida en todo el espacio objetivo X. [42].

Este modelo está principalmente basado en redes de neuronas, alimentadas con una serie de mecanismos que explotan su potencial trabajando con secuencias de entrada cuyo contenido depende del contexto que le precede y le sucede. Ejemplos de estas pueden ser desde secuencias de lenguaje natural (NL), de lenguaje de programación (PL), de los bits que forman una imagen, de los frames de un vídeo... Simplificando: la arquitectura logra *transformar* la secuencia de entrada en una secuencia de salida, definida en función del contexto objetivo que puede ser desde traducir textos, resumir programas, identificar código con comportamiento malicioso o generar imágenes a partir de texto. Su arquitectura se detalla en la Figura 2.6.

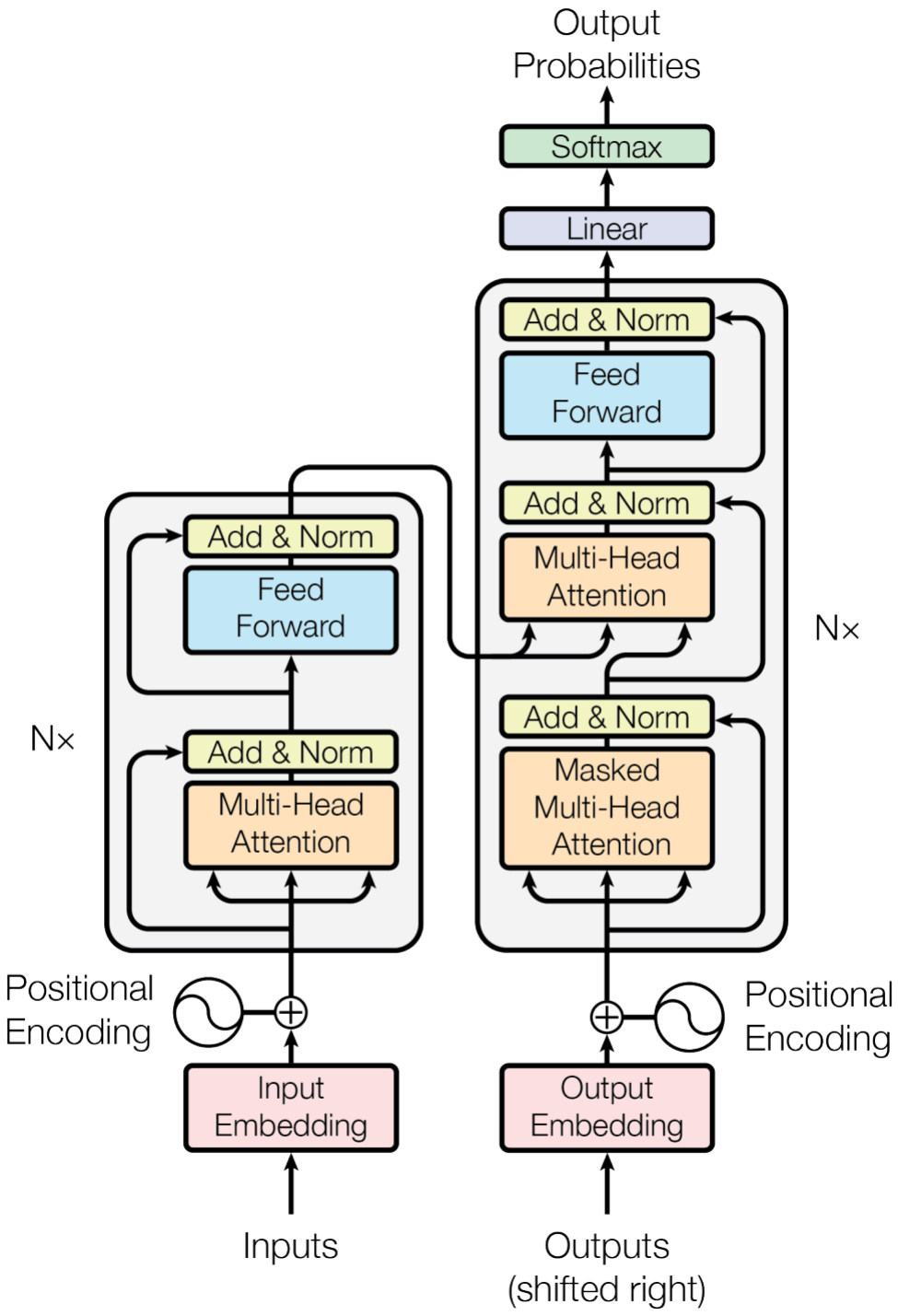


Figura 2.6: El modelo de arquitectura Transformer. [1]

La principal ventaja que aporta este modelo es que supera las limitaciones de las RNNs cuando trabajan con secuencias largas: estas redes neuronales pierden información relativa a los primeros elementos cuando se introducen nuevos elementos, por lo que cuanto más larga sea la secuencia, menos tendrán en cuenta los primeros elementos.

En las siguientes subsecciones se profundizará en las claves del funcionamiento y éxito de esta arquitectura.

Bloque Encoder

El bloque o pila encoder (stack encoder), es una de las dos las estructuras principales del Transformer y genera una codificación de la entrada con la información atencional correspondiente a sus elementos, habiendo **comprendido** la estructura e información de la entrada. Gráficamente, lo podemos ver representado en el bloque coloreado de gris situado a la izquierda en la imagen 2.6. Este bloque está compuesto de una pila de 6 capas idénticas que a su vez están compuestas de dos subcapas: la "Multi-Head Attention", explicada más adelante y conectada a una capa de normalización y la otra, una Feedforward neural network o red neuronal prealimentada aplicada a cada posición de la entrada que consiste en dos transformaciones lineales con una función ReLU:

$$FNN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

Bloque Decoder

El bloque o pila decoder (stack decoder), es la otra estructura principal del Transformer y gracias a la codificación proporcionada por el encoder y al input que se le proporciona, que es la secuencia de salida desplazada hacia la derecha, **genera** la secuencia deseada. Gráficamente, lo podemos ver representado en el bloque coloreado de gris situado a la derecha en la imagen 2.6. Está compuesto de las mismas capas que el encoder, con la diferencia de que dentro de cada una de las 6 capas, tiene una subcapa extra, que incorpora otra "Multi-Head Attention" pero, que recibe como entrada la salida del bloque encoder. Además, el otro bloque "Multi-head attention" enmascara ciertas partes de la entrada, para evitar que se les preste atención y así, junto al desplazamiento de la entrada, asegura que las generaciones o predicciones que realiza solo dependen de información ya conocida (que había sido generada anteriormente).

Representación de la entrada - Tokenizer

Una de las partes fundamentales del Transformer es la representación de las secuencias de entrada. Cuando esta secuencia de entrada está compuesta por palabras, el mecanismo utilizado es el siguiente:

En primer lugar, es necesario convertir las palabras a números para poder aplicar operaciones matemáticas sobre estos. Esto se realiza a través de un mecanismo que facilita el trabajo a las máquinas: one-hot encoding. El one-hot encoding basa conversión o codificación mediante una representación de los elementos a través de un array de ceros salvo el elemento a representar, cuya longitud es la misma que la longitud total del vocabulario ver 2.7.

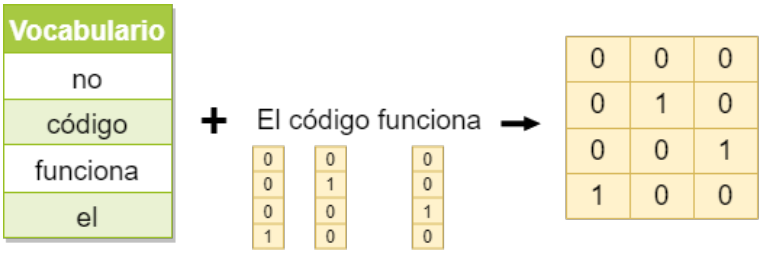


Figura 2.7: Ejemplo de one-hot encoder

Sin embargo, es necesario definir qué compone el vocabulario. Para esto, esta arquitectura se apoya en la construcción del vocabulario utilizando un codificador de pares de bytes, "Byte pair encoder". El funcionamiento del byte pair encoder es el siguiente, cada carácter del conjunto de datos tiene asociada una representación única en forma de byte. Posteriormente, se asocian los pares de caracteres más comunes recibiendo una nueva representación que es incorporada al conjunto de datos, tarea que se repite sucesivamente hasta que represente las secuencias de caracteres más repetidas, formando palabras e incluso expresiones. Este mecanismo de codificación aporta ventajas frente a otros como recoger todas las palabras de diccionarios de un determinado idioma, ya que es capaz de recoger expresiones, palabras nuevas o incluso faltas ortográficas.

La unión del one-hot encoder y del byte pair encoder es utilizada en un proceso vital para estos algoritmos denominada tokenización, a través de la cual transformamos la secuencia de entrada en una secuencia matemática con la que el Transformer pueda operar.

Codificación posicional - Positional encoding

En el apartado anterior se describe como codificar adecuadamente los elementos de la entrada, no obstante, en esta codificación no se tiene en cuenta una parte vital: la posición. Mientras que en

otras arquitecturas como las RNNs, la posición es inherente a la arquitectura, ya que interpreta la entrada secuencialmente, en los Transformers no es así, por tanto, es necesario incluir la posición explícitamente. Esto aporta ventajas interesantes, ya que posibilita la paralelización y de esta forma incrementar la velocidad de entrenamiento además de reducir la perdida de información de los primeros elementos de la entrada.

La solución implementada por los autores implica añadir información posicional a cada elemento de la entrada (una vez codificada por el tokenizador). La posición no es añadida en forma de un simple número, se trata de un vector con *d* dimensiones que contiene la información acerca de una determinada posición de la entrada y es fusionada con el elemento que posiciona. Para añadir esta información se utilizan funciones de seno y coseno en distintas frecuencias:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Figura 2.8: En las funciones, *PE* se refiere a la codificación posicional, *pos* hace referencia a la posición y *i* hace referencia a la dimensión de la codificación posicional corresponde a una sinusoide [1].

La siguiente imagen representa de forma intuitiva como representar los primeros 16 números naturales a través de estas funciones.

0 :	0 0 0 0	8 :	1 0 0 0
1 :	0 0 0 1	9 :	1 0 0 1
2 :	0 0 1 0	10 :	1 0 1 0
3 :	0 0 1 1	11 :	1 0 1 1
4 :	0 1 0 0	12 :	1 1 0 0
5 :	0 1 0 1	13 :	1 1 0 1
6 :	0 1 1 0	14 :	1 1 1 0
7 :	0 1 1 1	15 :	1 1 1 1

Figura 2.9: Podemos observar como las funciones sinusoidales son equivalentes a la alternancia de bits utilizada para representar números en binario. [43]

Atención es todo lo que necesitas

El concepto revolucionario que trajo la arquitectura de Transformer es el de ”Atención”. No obstante, este mecanismo no nació con los Transformers, sino que algunas redes neuronales recu-

rrentes ya los incorporaban aunque no eran iguales. La diferencia se basa en el lugar de aplicación: en los Transformers se introdujo este mecanismo en cada capa de los bloques encoder y decoder, mientras que en las RNN se encontraba fuera de estos.

La implementación utilizada por la arquitectura Transformer es un tipo de atención específico, el ”self-attention”. Simplificando, la idea detrás del self attention es que a la hora de generar la ”vectorización” de un elemento de la entrada en una capa n (denominado embeding), se asignarán unos pesos que indicarán el nivel de “atención” o relevancia que tienen los embeddings resultantes de la capa n-1. De esta forma, al entrenar la red, la representación interna de elementos de entrada se irá adaptando en cada capa para que aprenda a darle la atención adecuada a cada elemento de la entrada dentro del contexto de esa entrada. Por ejemplo, en las frases ”La planta debe recibir luz solar para sobrevivir” y ”Los empleados de la planta están en huelga”, planta tiene significados distintos, los cuales nuestra arquitectura será capaz de comprender gracias a la información que le aportan palabras de su contexto como ”recibir luz solar” en el primer caso y ”empleados” en el segundo.

El mecanismo de atención es implementado mediante el ”scaled dot-product attention”, el cual se basa en una multiplicación matricial entre una matriz de consulta (Q, Query), una matriz clave (K, key) y una matriz de valor (V, value), cuya fórmula se puede ver en 2.10.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Figura 2.10: Fórmula utilizada para computar la atención en la arquitectura Transformer [1]

No obstante, el mecanismo de self-attention no es tan trivial: este mecanismo es implementado en un módulo, dentro del cual se computa, no solo una vez, sino que se divide la entrada y se computa el ”scaled dot-product” en cada subespacio que hemos dividido, de manera paralela. A la salida, los resultados se concatenan y posteriormente se transforman linealmente en las dimensiones esperadas. De esta forma, el modelo puede de manera concurrente prestar atención a la información en distintas posiciones, mejorando la interpretación de las entradas.

De manera esquemática (matizar que K,V y Q son matrices): las claves (K), representan los valores (V) en un espacio vectorial. Por tanto, al aplicar el producto escalar entre la clave y la consulta (Q), estaríamos obteniendo la representación del valor (K_i) más cercano a la consulta, es decir una respuesta a esta, que posteriormente, tras aplicarle la función softmax, nos servirá para señalar matemáticamente el valor correspondiente dentro de la matriz (V). Si volvemos a observar la arquitectura, ver 2.6, podemos advertir como dentro del bloque decoder, en el segundo sub-bloque

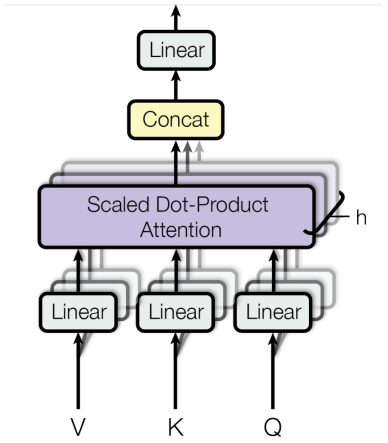


Figura 2.11: Mecanismo de atención - Multi-Head Attention [1]

de ”Multi-Head Attention” el bloque encoder proporciona dos entradas al decoder, estás serían el par clave, valor (K, V); siendo el bloque decoder el que proporciona la consulta (Q). Además, tanto el bloque encoder como el decoder poseen el mismo sub-bloque de atención, obteniendo los tres valores de sus respectivas entradas.

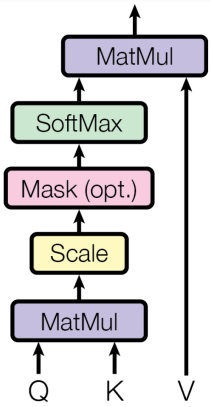


Figura 2.12: Representación gráfica y simplificada de la atención mediante el producto escalar - Scaled Dot-Product Attention [1]

Transformer en pocas palabras

Tras haber descrito las principales estructuras del Transformer, se describe resumidamente su entrenamiento:

Se dispone de una entrada y de la salida producida. La entrada se proporciona al bloque encoder, mientras que al decoder se le proporciona la salida producida desplazada a la derecha. El bloque encoder aplica atención a los distintos elementos de su entrada y posteriormente se le proporciona

la salida al bloque decoder, el cual, concurrentemente ha aplicado atención a la salida producida anteriormente, con posiciones enmascaradas. Después, con la información del encoder, tratará de generar la secuencia deseada, aprendiendo así la importancia y estructura de los elementos de la entrada, además de la importancia y estructura interna de salida que ha sido producida anteriormente, generando la secuencia objetivo con una comprensión mucho mayor.

Tipos de Transformers

Englobamos dentro de *Transformers* a aquellas arquitecturas que utilizan en sus implementaciones de mecanismos derivados del papel [1]. A pesar del gran impacto y relevancia de Transformer, han surgido multitud de variaciones respecto a su implementación que han demostrado tener éxito en diversos campos, siendo algunas más aptas para realizar ciertas tareas.

Una de las variaciones más exitosas es la aplicación del transfer learning al Transformer, incluyéndose dentro del grupo de los considerados Pre-trained Language Models (PTM) que se engloban dentro del campo del NLP. Estas arquitecturas son entrenadas sobre largos corpus de datos para aprender estructuras inherentes a los lenguajes naturales y así poder luego ser entrenados para tareas concretas, optimizando el entrenamiento y mejorando su rendimiento. [44]

No obstante, surgen variaciones de la arquitectura a nivel estructural, las cuales podemos agrupar en tres grupos [45]:

- En primer lugar, los modelos *encoder-only* o solo codificador, basados en la eliminación del bloque decoder, utilizando el bloque encoder como columna vertebral del modelo. En estas arquitecturas, la salida del bloque encoder se utiliza como una representación de la secuencia de entrada, y están estrechamente relacionadas con tareas de NLU como la clasificación de texto. Ejemplos de estas arquitecturas puede ser el famoso BERT [46] o RoBERTa [47], una adaptación del primero.
- En segundo lugar, los modelos *decoder-only* o solo decodificador, basados únicamente en el bloque decoder aunque modificado, eliminando el último bloque de atención que utilizaba la salida del encoder como entrada. Estos modelos se emplean en tareas de NLG como la generación de texto (por ejemplo, escribir el inicio de una historia y que sea completada por el modelo). Es el caso de modelos como la familia GPT [48].
- Por último, los modelos encoder-decoder, también conocidos como modelos sequence-to-sequence, son aquellos que siguen la estructura descrita en este trabajo, preservando tanto el bloque encoder como el decoder, lo cual proporciona capacidad para realizar tareas tanto

de NLU como de NLG como es el caso de la traducción de idiomas o la realización de resúmenes. Ejemplos de este tipo de arquitecturas que son utilizadas hoy en día puede ser BART [49] o T5 [50].

2.6 Aplicaciones de los Transformers

La estructura de estas arquitecturas y de los conceptos en los que se apoyan: desde el transfer learning, pasando por los mecanismos de atención o incluso la paralelización de las entradas, modelan una arquitectura que emula la estructura de aprendizaje y razonamiento humanos a unos niveles que no habían sido vistos anteriormente en el campo de la inteligencia artificial, siendo apoyadas en rendimiento empírico. Es por esto, que los campos de aplicación de los transformers en los que tienen rendimientos a nivel de estado del arte son cada vez más y algunos de ellos serán expuestos a continuación.

Se pueden ver aplicados en campos como: la visión artificial (computer vision), dónde destacan arquitecturas como DALL-E, la cual supuso un salto en el rendimiento de estado del arte en la generación texto a imagen, obteniendo una gran calidad en las imágenes [51]. Otro campo en el que destaca su uso es el campo de audio, dónde destaca la tarea de conversión de voz a texto, con modelos como wav2vec que supera el anterior estado del arte con creces [52]. Por último y no menos importante, ya que fue el propósito inicial de estas arquitecturas: se utiliza en el campo del procesado de lenguaje natural NLP; además, mencionar el campo del procesado de lenguaje de programación. En ambos campos se enmarca este trabajo y que describiremos en las siguientes secciones.

NLP

El Natural Language Processing (NLP) es un campo de la lingüística, estadística y ciencias de la computación que consiste en el análisis, procesado y comprensión del lenguaje natural. La popularización de conceptos como el data minning o el big data ha posibilitado que los investigadores tengan acceso y sean capaces de recoger grandes cantidades de datos de textos, dando lugar a grandes avances en la investigación que van unidos a la incorporación de algoritmos de machine learning en el campo. Desde entonces, el NLP se ha aplicado a multitud de campos con éxito, como es el caso de la traducción automática con traductores como el de Google o DeepL, los asistentes personales como Siri o Cortana o incluso chatbots de asistencia como es el caso de YUBI en la web de la UPM. Destacar el éxito del revolucionario ChatGPT: un modelo chatbot basado en una arquitectura Transformer con capacidad de mantener una conversación, respondiendo a

preguntas y tareas de diversos tipos, entre ellas el entendimiento de código fuente. Lanzado en noviembre de 2022, ha atraído una enorme atención debido a que en su primera fase de preview, ha sido abierto al público en formato web, reuniendo una enorme masa de usuarios en proporción del tiempo desplegado, que no vista anteriormente en prácticamente ningún servicio web, todo gracias a su impresionante rendimiento [53]. [54]

Existen multitud de tareas relacionadas con el NLP, todas están relacionadas con la automatización de tareas de comprensión del lenguaje por parte de los ordenadores pero se pueden clasificar en dos grandes ramas: la comprensión del lenguaje natural y la generación del lenguaje natural. El propósito de la primera, NLU, consiste en la comprensión de las estructuras internas del lenguaje natural como la gramática y el contexto, y la extracción de información relevante para comprender el significado de fragmentos de lenguaje. Por otro lado, el propósito de la segunda NLG consiste en la generación de texto natural que tenga sentido sintáctico y gramatical y por tanto sea comprensible por los seres humanos. [55]

Resumen de textos

Como se ha comentado anteriormente en este trabajo, en los últimos años, la cantidad de información generada, almacenada y por tanto disponible es cada vez mayor. Esto implica que a la hora de formarnos sobre un tema, necesitemos realizar un filtrado exhaustivo entre la cantidad de información disponible, generando la necesidad de desarrollar métodos para facilitar este filtrado. Es en este contexto que surge el resumen de textos automático. Esta tarea consiste en generar un resumen claro y conciso del texto objetivo, reflejando los contenidos principales de este. Existen dos enfoques a la hora de abarcar el problema del resumen automático de texto: la extracción y la abstracción. [56]

- El enfoque basado en la extracción (extractive) se basa en identificar las partes principales del texto, recogerlas y elaborar el resumen a partir de ellas. Esto es, extraer fragmentos del texto y recopilarlos. Este enfoque puede ser muy útil cuando necesitamos filtrar por contenidos del texto, pero no tan útil si queremos un resumen más *humano* (la mayoría de los resúmenes humanos no son extractivos). [56]
- Por otro lado, tenemos el enfoque basado en abstracción (abstractive), el cual se basa comprender en profundidad el lenguaje, para poder extraer las estructuras semánticas principales del texto objetivo y producir o generar un resumen original reconstruyendo frases o partes de este. Frecuentemente, este enfoque se apoya en un primer análisis extractivo para después poder generar el resumen original más fácilmente. Este tipo de resúmenes

son útiles cuando necesitamos un entendimiento más profundo del texto objetivo pero son radicalmente más complejos al requerir un entendimiento más profundo del lenguaje. [57]

2.7 T5 - Text-to-Text Transfer Transformer

La arquitectura Transformer utilizada en este trabajo es el Text-to-Text Transfer Transformer (T5). El T5 fue introducido en 2019 en un papel de Google, que no solo describe la arquitectura, sino que a la vez realiza un estudio sistemático del campo del NLP en relación con los Transformers, seleccionando las mejores técnicas para diseñar un modelo exitoso [50].

El T5 se engloba dentro de la familia de Transformers encoder-decoder, ya que tras realizar un estudio de las fortalezas y debilidades de este tipo de Transformers, se concluye que son los óptimos para realizar tareas de secuencia a secuencia, en las cuales se centra la aplicación de este modelo.

La implementación es muy similar a la introducida en el primer Transformer [1] aunque con claras diferencias respecto al tamaño (número de parámetros) y alguna en cuanto a implementación. Esta arquitectura descarta el uso de funciones sinusoidales para representar la codificación posicional de la entrada, en pos de la utilización de un mecanismo que cada vez es más común en arquitecturas Transformers, basado en la codificación posicional relativa, según la cual, en lugar de utilizar una posición fija utilizando la fórmula de 2.8. Por otra parte, se utiliza una versión simplificada de la capa de normalización en la que se suprime el bias y se sitúa esta capa de normalización, fuera del camino residual (residual path).

Como el propio nombre indica, Text-to-Text Transfer Transformer, el T5 hace uso del transfer learning que ha demostrado tener éxito en el NLP. En concreto, el modelo es pre-entrenado en un corpus enorme de datos sin etiquetar denominado Colossal Clean Crawled Corpus (C4) que consiste en texto limpiado a partir de una serie de heurísticas de un volcado de la web Cromon Crawl. Para realizar el pre-entrenamiento en este corpus, se realiza un estudio de los objetivos más populares y útiles, concluyendo que el insertado de ruido en forma de corrupción de fragmentos de la entrada y su posterior identificación es el objetivo que ofrece mejores resultados en términos de eficiencia y eficacia (denoising objetive). Por otra parte, una de las novedades que incluye este papel, es la conversión de todas las tareas de *fine-tunning* a tareas de texto a texto. Esto es, en algunas arquitecturas como BERT [46], se ajusta el modelo después de haber sido entrenado para la realización de tareas como puede ser la clasificación de texto, donde la salida es una etiqueta. En este caso, T5 no trata la salida como una etiqueta. sino como texto natural. Para ello, se añade un prefijo de texto a las secuencias de entrada que identifique la tarea a realizar. Por ejemplo, para traducir del inglés al español la frase ”I am good”, se le proporcionaría al modelo la entrada

”translate English to Spanish: I am good” y la secuencia objetivo sería ”Estoy bien”.

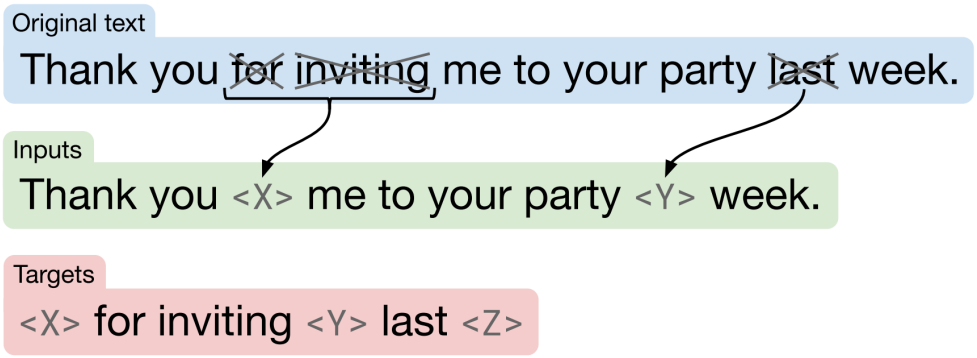


Figura 2.13: Un ejemplo del objetivo utilizado en el pre-entrenamiento del T5, en el cual los distintos fragmentos tachados han sido elegidos aleatoriamente y se han ocultado al modelo sustituyéndose por tokens únicos que han de ser predichos por el modelo. [50]

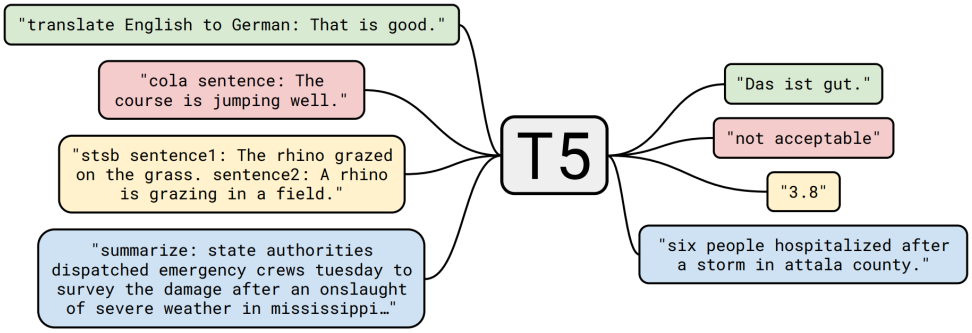


Figura 2.14: Un ejemplo de las distintas tareas que es capaz de realizar el T5. [50]

2.8 Transformers para código

En esta sección se describirá una de las aplicaciones de los transformers: el procesado del código fuente. Además, se describirán algunas de las arquitecturas más relevantes de Transformers para código para este proyecto.

Procesado del código fuente

A pesar de que las máquinas pueden compilar código fuente, no son capaces de entender aspectos ”humanos” de este como podría ser el funcionamiento, comportamiento, complejidad o maliciosidad. Es en este contexto que surge el campo del procesado de lenguaje de programación. Aparece como una intersección entre el análisis de código fuente y el machine learning y aunque podría estar englobado dentro del NLP debido a la inclusión de lenguaje natural en este, el código fuente

es un lenguaje que sigue unas reglas sintácticas estrictas que divergen de las comunes al lenguaje natural.

Mientras arquitecturas anteriores como las RNNs se centraban en relaciones locales entre los elementos de entrada, los Transformers son capaces de procesar la entrada en paralelo y entender las relaciones globales de todos los elementos, logrando representaciones más acertadas del código. [58]

En ocasiones, en algunas arquitecturas, el código fuente es tratado como lenguaje natural y es proporcionado como entrada única, no obstante, como se ha mencionado anteriormente, por la naturaleza estructural del código, estos modelos tendrán una peor comprensión del código, por lo que se plantean alternativas para codificar esta estructura sintáctica como es el caso de los árboles estructurales de código.

AST

El Abstract Syntax Tree (AST) o árbol de sintaxis abstracta es una representación de la estructura sintáctica del código fuente en forma de árbol, según el cual, cada nodo representa un constructo sintáctico contenido en la fuente (bifurcaciones de código, identificadores, asignaciones de variables...). Se denominan abstractos, ya que no representan exactamente todo el texto, sino que evitan ciertos elementos como la puntuación o delimitadores, por lo que suponen una abstracción respecto al texto que representan. El AST [59]

Otras formas de representar la naturaleza sintáctica del código han sido propuestas como es el caso de las redes neuronales basadas en arboles (TNNs) que, utilizan ASTs como entrada o las redes de neuronas basadas en AST (ASTNN). No obstante, el AST es considerada una de las estrategias más efectivas de sinterización del código, usada extensamente en otros campos como los compiladores, por lo que no se tratarán el resto de las alternativas. [59]

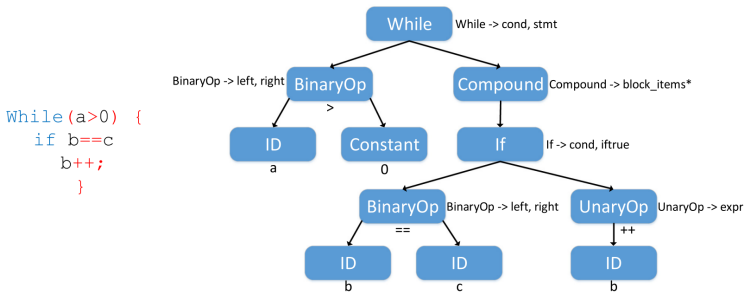


Figura 2.15: Un ejemplo de fragmento de código en C y su representación correspondiente en AST [60]

Se ha demostrado que los Transformers, al ser una arquitectura enfocada para procesado de secuencias como es el caso del NLP: tienen éxito en tareas relacionadas con el procesado de código fuente siempre y cuando se adapte de forma adecuada, consiguiendo rendimiento de estado del arte en múltiples tareas de este campo. [58]

Estudio comparativo de Transformers para código

Para seleccionar un modelo Transformer capaz de comprender y trabajar con código fuente adecuadamente, se ha realizado un estudio comparativo de los distintos modelos publicados, ya que el procesado de código mediante Transformers es un campo sobre el que no existen tantas publicaciones en comparación con los Transformers aplicados a NLP. Destacar que existen modelos en producción como es el caso de Codex [61] en el famoso copilot de GitHub: un asistente de desarrollo que genera automáticamente código en función del contexto del usuario; que desgraciadamente, ni han hecho público su modelo ni han publicado su implementación, por lo que no pueden ser objeto de estudio.

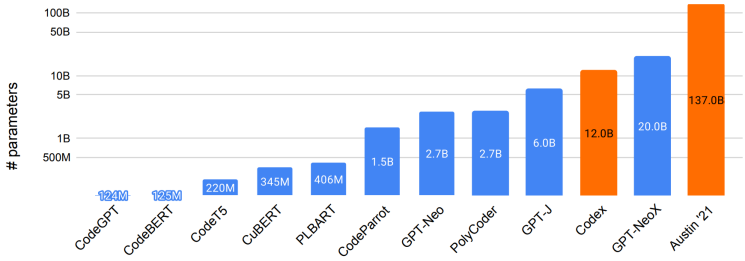


Figura 2.16: Algunos Transformers de procesado de código fuente, dónde se muestran sus tamaños (eje Y), y su disponibilidad, siendo los azules los modelos públicos (open source) y los naranjas los mantenidos en privado (not open source) [62]

CuBERT

Uno de los primeros modelos Transformer para procesar código es CuBERT [63], Este modelo, es entrenado en un corpus de código en Python sobre el modelo BERT, al cual se le proporciona el código fuente sin tratamiento, empleando sus famosos objetivos de entrenamiento: modelado de lenguaje enmascarado (MLM) y la predicción de la siguiente frase (NSP).

Al dedicarse exclusivamente al lenguaje Python y no tratar de forma especial el código fuente, descartamos profundizar en el funcionamiento del modelo y utilizarlo.

CodeBERT

Este modelo, CodeBERT [64], junto con CuBERT, son considerados los primeros Transformers en código fuente. No obstante, CodeBERT es entrenado sobre un corpus de datos mucho más grande, que comprende hasta 6 lenguajes de programación distintos: Go, Java, JavaScript, php. Python y Ruby [65]. Además, este se implementa sobre la arquitectura RoBERTa [47], recibiendo dos tipos distintos de entrada: entradas bimodales, compuestas por lenguaje natural seguido de código fuente separados por un token delimitador y entradas unimodales, formadas únicamente por código fuente o por lenguaje natural. Los objetivos utilizados para el entrenamiento son, el modelado de lenguaje enmascarado (MLM) para las secuencias bimodales y la detección de tokens reemplazados (RTD) para tanto las secuencias bimodales como para las unimodales.

Este modelo se ajusta para la realización de cuatro distintas tareas:

- Selección de código dentro de una colección que más se aproxime a una descripción en lenguaje natural.
- Predicción de elementos enmascarados entre una serie de ejemplos: en secuencias de lenguaje natural solo se pueden enmascarar las palabras *max, maximize, min, minimize, less, greater* y en código fuente *max, min*. El propósito de esta tarea es que el modelo aprenda a completar código.
- Generación de documentación a partir de código: para poder realizar esta tarea, se acoplan distintos bloques encoder pre-entrenados de otras arquitecturas.
- Generación de resúmenes a partir de código en un lenguaje desconocido para la arquitectura, C#.

GraphCodeBERT

GraphCodeBERT [66] es entrenado en el mismo dataset que CodeBERT, el CodeSearchNet [65]. No obstante, GraphCodeBERT se implementa sobre BERT, y recibe como entradas lenguaje natural, código fuente y una representación semántica del código. Este modelo denota el problema de distintas arquitecturas que realizan tareas relacionadas con código fuente que tratan el código como si fuera lenguaje natural, ignorando sus propiedades estructurales. Por tanto, incorpora como representación del código su data-flow, el cual representa el flujo de los datos en las operaciones que ocurren en un programa. Descarta el uso del AST, ya que lo considera demasiado complejo y detallado.

El entrenamiento de la arquitectura se basa en tres objetivos. El primero de ellos es el modelado de lenguaje enmascarado característico (MLM) de BERT, el segundo de ellos consiste en elegir aleatoriamente nodos del data flow, y enmascarar las aristas que conectan directamente estos nodos, con el objetivo de que el modelo identifique estas aristas, aprendiendo así el flujo de las variables en el código. El último consiste en identificar el elemento del código asociado a un nodo del data flow, entendiendo así la relación entre el dataflow y el código fuente.

Posteriormente, el modelo se ajusta para realizar cuatro tareas:

- Selección del código dentro de unas opciones que más se aproxime a una descripción en lenguaje natural.
- Una tarea de clasificación correspondiente a la detección de código clonado entre dos fragmentos.
- Una tarea de traducción de código desde un lenguaje conocido a uno desconocido (Java a C#) y al contrario (C# a Java).
- Solución de bugs o errores en un trozo de código, en el cual se presentan fragmentos de código tanto con bugs como sin ellos para que sean reparados por el programa.

Además, se realiza un estudio comparativo acerca de la utilización del data flow frente al AST, Podemos observar en 2.17 como representar la naturaleza estructural del código tiene sentido a mayor sea la longitud del código y cómo la introducción del AST tiene peor rendimiento que el uso del dataflow.

Code Transformer

El CodeTransformer [67] subraya la importancia de, como se comprobó en GraphCodeBERT, entrenar los modelos con contexto (referido al código fuente) y con la estructura (referido a estructuras del tipo AST). Para ello, proponen un modelo implementado sobre XLNet [68], una arquitectura formada únicamente por el bloque encoder y cuya característica más destacable es que carece de límite de tamaño en la secuencia de entrada. El CodeTransformer recibe como entradas código fuente y el cálculo de distancias por pares en el AST (como las longitudes de los caminos más cortos entre dos nodos i, j entre otras distancias explicadas en el papel).

Está entrenada sobre partes del dataset CodeSearchNet [65], en concreto las correspondientes a los lenguajes Python, JavaScript, Ruby y Go [69]. Además, se apoya en el data set Java-small, basado en

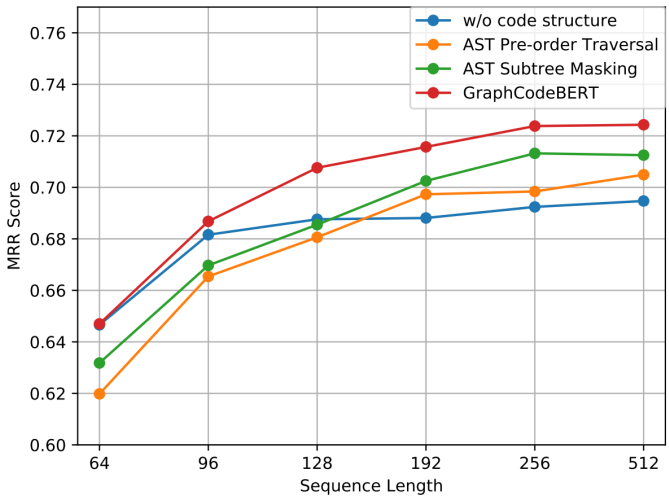


Figura 2.17: Representa el rango reciproco medio MRR en la validación del dataset de Ruby para la tarea de selección de código frente a la longitud de la entrada [66].

resumen de código, junto con java-mendium y java-large [70]. Este modelo se entrena únicamente con el objetivo del resumen de código, dividiendo los datasets en entrenamiento, validación y test.

El CodeTransformer resultó ser el primer modelo entrenado para resumir código en múltiples lenguajes de programación, logrando demostrar de nuevo la importancia de representar la naturaleza estructural del código y en este caso el potencial de representarlo mediante distancias entre distintos nodos del AST.

PLBART

El modelo PLBART [71] está implementado sobre el modelo Transfomer BART, el cual se basa en una arquitectura encoder-decoder, permitiendo realizar tareas tanto de NLU como de NLG. PLBART ha sido entrenado sobre datatasets con código en Java y Python. Uno de ellos es un subconjunto de los repositorios de GitHub de estos lenguajes extraídos de Google BigQuery siguiendo la pipeline descrita en [72]. El otro consiste en posts con lenguaje natural y código fuente extraídos de Stack Overflow en el dump de septiembre de 2020 de stackchange.

Para el entrenamiento del modelo, se le proporcionan entradas bimodales compuestas de lenguaje natural seguido de código fuente con tres objetivos que utiliza la arquitectura BART [49]: dos basados en el enmascaramiento de fragmentos de la entrada: uno enmascara elementos en relación 1:1 con la máscara que el modelo debe predecir (el MLM propuesto en la arquitectura BERT) mientras que el otro enmascara elementos en relación N:1 con la máscara y posteriormente el modelo

debe predecir cuantos elementos han sido enmascarados. El otro está basado en la eliminación de fragmentos de la entrada en la que el modelo debe decidir que posiciones han sido eliminadas.

Posteriormente, se ajusta PLBART para la realización de cuatro tareas:

- Tareas de clasificación de secuencias, en las que el modelo se enfrenta al típico problema de clasificación de ML como la detección de código clonado, y la detección de código vulnerable (código en C/C++)
- Traducción de código de un lenguaje conocido a otro desconocido, Java a C# y viceversa.
- Otra de ellas es el resumen de fragmentos de código en lenguaje natural, para lo que se incorporan 4 lenguajes de programación no vistos anteriormente por el modelo: PHP, Ruby, Go y Javascript.
- Por último, la generación de código, que es lo opuesto al resumen de código: generar código a partir de un fragmento de lenguaje natural.

PLBART destaca por ser uno de los primeros Transformes de código fuente que utiliza una arquitectura encoder-decoder, en concreto BART, mostrando su gran potencial para la realización de tareas de entendimiento y generación del lenguaje.

CoTexT

La arquitectura CoTexT [73] se implementa sobre el modelo T5, descrito anteriormente en 2.7, concretamente con un checkpoint de esta arquitectura que ha sido entrenada en lenguaje natural.

Para su entrenamiento, más allá de partir del checkpoint, se entrena sobre dos datasets. El primero de ellos, CodeSearchNet, con los 6 lenguajes de programación ya mencionados y con datos extraídos de repositorios de Google BigQuery en Java y Python, mencionados también anteriormente en 2.8.

En el entrenamiento, se proporcionan datos al modelo de forma bimodal, lenguaje natural seguido de código fuente y de forma unimodal, únicamente código fuente. Fragmentos de estas entradas son enmascaradas de manera aleatoria y el modelo trata de predecir los fragmentos.

Para ajustarlo se emplean las tareas de resumen de código a lenguaje natural, generación de código a partir de lenguaje natural, reparación de código con errores y clasificación de código en función de si tiene defectos (es vulnerable) o no.

CodeT5

En esta sección se presenta el modelo utilizado en este trabajo, el CodeT5 [27], que como el nombre indica, se cimienta sobre la arquitectura Transformer T5. Posteriormente se explicarán los detalles que han dado lugar a la selección de la arquitectura frente a las demás.

CodeT5 ha sido entrenado sobre el dataset completo de CodeSearchNet (Java, Javascript, Ruby, Python, PHP y GO) y además, se añaden dos datasets extraídos de Google BigQuery, con repositorios de GitHub con código en C y C#. Se le proporcionan como entrada tanto datos bimodales con lenguaje natural seguido de código fuente como datos unimodales con únicamente código fuente. A lo largo del papel y de este trabajo, se ha remarcado la importancia de tratar el código de forma distinta al lenguaje natural, por sus naturaleza estructural. Para eso, CodeT5 no lo incorpora en las entradas sino que se incorpora en los objetivos de entrenamiento del modelo :

- El primer objetivo y el que resulta novedoso es, el ”Identifier tagging” (Fig 2.18 b), el cual es el que se aprovecha del AST: convierte el segmento de código en su AST asociado y extrae los tipos de cada nodo representado en cada caso si son identificador o no (si contienen información relevante o no) y se le proporciona al modelo.
- En segundo lugar, la estrategia ”Masked Span Prediction” (Fig 2.18 a), la cual es la misma que se utiliza en el modelo T5, conocida como ”Sentence Piece”, y trata de añadir ruido aleatorio a todo el código, sin diferenciar entre variables o instrucciones, y que el modelo las descifre.
- En tercer lugar, el ”Masked identifier Prediction” (Fig 2.18 c), el cual, aleatoriamente oculta todas las ocurrencias de un identificador y fuerza a predecir las localizaciones de este en el código, lo cual produce un aprendizaje similar a la deofuscación de código.
- Por último, la estrategia ”Bimodal Dual Generation” (Fig 2.18 d) aprovecha los identificadores asignados por el desarrollador en el código. Cuando escriben programas, los desarrolladores tienden a emplear comentarios con identificadores informativos para hacer el código más comprensible, de modo que estos comentarios suelen contener información interesante del código (por ejemplo, el identificador ”binarySearch” de la Figura 2.18 a) indica directamente su funcionalidad). Por lo que se intenta obtener el código asociado a estos comentarios y viceversa, enseñando al modelo la relación entre el lenguaje natural y el código fuente. La forma que tiene [27] de saber el lenguaje objetivo es gracias a la introducción de un elemento que identifique (por ejemplo, para referir Java será <java> y para referir al inglés <en>).

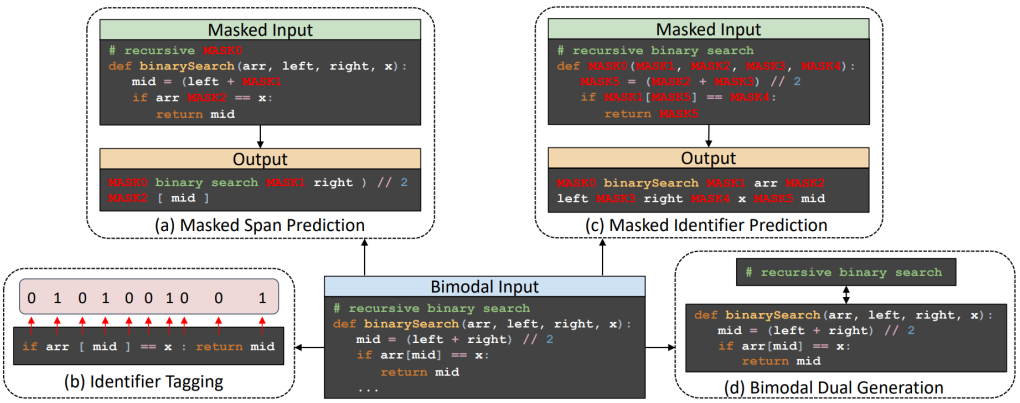


Figura 2.18: Tareas de preentrenamiento de CodeT5. Primero se entrena alternativamente la predicción de tramos, la predicción de identificadores y el etiquetado de identificadores en datos unimodales y bimodales, y luego se aprovechan los datos bimodales para el entrenamiento de doble generación. [27]

El CodeT5 ha sido diseñado para, en su versión entrenada y publicada, ser capaz de realizar distintas tareas relacionadas con comprensión y generación de código.

- En primer lugar, es capaz de realizar resumen de funciones de código a lenguaje natural, en concreto al inglés (generación de texto).
- Generación de código a partir de una descripción textual en inglés.
- Traducción de código de un lenguaje a otro (generación de código).
- Corrección de código con fallos.
- Detección de defectos, cuyo objetivo es clasificar código como vulnerable o no.
- Detección de código clonado, que trata de medir la similitud de dos fragmentos de código y predecir si tienen la misma funcionalidad.

Resultados

En esta sección, recogemos los resultados de las distintas arquitecturas que realizan resumen de código a texto natural, al ser la aplicación que daremos al modelo Transformer que incorporemos en nuestro trabajo.

Tras analizar las características de las distintas arquitecturas Transformer para código que hemos encontrado más interesantes y sus resultados en las distintas tareas, centrando nuestra atención especialmente en el resumen de código.

Methods	Ruby	JavaScript	Go	Python	Java	PHP	Overall
RoBERTa	11.17	11.90	17.72	18.14	16.47	24.02	16.57
CodeBERT	12.16	14.90	18.07	19.06	17.65	25.16	17.83
DOBF	-	-	-	18.24	19.05	-	-
PLBART	14.11	15.56	18.91	19.30	18.45	23.58	18.32
CodeT5-small	14.87	15.32	19.25	20.04	19.92	25.46	19.14
+dual-gen	15.30	15.61	19.74	19.94	19.78	26.48	19.48
+multi-task	15.50	15.52	19.62	20.10	19.59	25.69	19.37
CodeT5-base	15.24	16.16	19.56	20.01	20.31	26.03	19.55
+dual-gen	**15.73**	16.00	19.71	20.11	20.41	**26.53**	19.75
+multi-task	15.69	**16.24**	**19.76**	**20.36**	**20.46**	26.09	**19.77**

Figura 2.19: Puntuaciones en la métrica BLEU-4 para resumen de código. La columna ”Overall” muestra las puntuaciones medias de seis lenguajes de programación. [27]

Finalmente, se ha seleccionado el modelo CodeT5 [27], como modelo base para este trabajo. Para ello nos hemos basado en diversos criterios, el primero de ellos ha sido priorizar los modelos que habían sido entrenados en el lenguaje Java, ya que es el lenguaje del código fuente que analizaremos. Por otra parte, resultaba interesante que nuestro modelo se basase en una arquitectura encoder-decoder, ya que para la tarea de resumen de secuencias, es necesario realizar tareas tanto de entendimiento (NLU), como de generación (NLG). No obstante, no podíamos ignorar el hecho de que un mal diseño de arquitectura encoder-decoder, en la que no se tuviesen en cuenta propiedades como la naturaleza estructural del código, o la elección de objetivos de entrenamiento del modelo poco óptimos, o la utilización de datos de entrenamiento de mala calidad; afectaría notablemente al desempleo de nuestro objetivo, por lo que utilizamos como referencia puntuaciones en las distintas tareas recogidas en los papeles asociados a las distintas arquitecturas como la figura 2.19. En las cuales hemos observado que CodeT5 logra rendimiento de estado del arte en diversas tareas, entre ellas el resumen de código.

3. Metodología

En este capítulo se detalla el proceso seguido para la elaboración de la herramienta de análisis de aplicaciones Android basada en una pipeline de machine learning, compuesta por Transformers.

3.1 Herramientas utilizadas

Para la elaboración de esta a herramienta ha sido necesario apoyarse en diversas herramientas y tecnologías que serán descritas a continuación.

Antes que nada, destacar que para la ejecución de la herramienta sobre aplicaciones complejas, se ha utilizado un servidor con las siguientes características: un procesador Intel i9, 64 GB de RAM, 1 TB de SSD (unos 700 GB de espacio libre), 1 Nvidia RTX 3080 y el sistema operativo Ubuntu 20.04. Para aplicaciones más pequeñas se ha ejecutado en el ordenador personal del alumno.

Python

En primer lugar, el lenguaje utilizado para desarrollar la herramienta es Python. Python es un lenguaje de programación cuya popularidad no ha dejado de crecer en los últimos años. Es conocido por su sencillez y versatilidad, lo que lo convierte en una buena opción para una amplia gama de tareas y aplicaciones como el desarrollo web, análisis de datos, inteligencia artificial, machine learning o ciencia de datos. Algunas de las razones por las que ha sido seleccionado para este proyecto son las siguientes:

- Tiene una sintaxis sencilla y legible, por lo que es una buena opción para desarrollar modelos y herramientas de aprendizaje automático.

- Tiene una serie de potentes librerías para el aprendizaje automático, como TensorFlow y PyTorch, que podrían utilizarse para entrenar y desplegar los modelos basados en Transformer para el resumen del código.
- Es un lenguaje versátil que puede utilizarse para una amplia gama de aplicaciones, lo que lo convierte en una buena opción para poder añadir características a la herramienta de manera sencilla como puede ser una interfaz web.
- Tiene una comunidad grande y activa, lo que significa que hay una gran cantidad de recursos en línea y apoyo disponible para aquellos que trabajan con el lenguaje.

En general, el uso de Python en esta herramienta agilizaría el proceso de desarrollo y facilitaría el acceso a una serie de herramientas y bibliotecas útiles que harían más sencillo el desarrollo.

Un ejemplo de estás es **PyTorch**, biblioteca de código abierto utilizada en aprendizaje automático, en esta herramienta nos apoyaremos para realizar todas las operaciones relacionadas con Transformers. Otra de ellas es **transformers**, librería popular en el ámbito del NLP, respaldada por HugginFace que incluye una gran variedad de modelos pre-entrenados que utilizaremos en nuestra herramienta como es el caso de T5 y CodeT5.

Por otra parte, utilizaremos la librería **Dash**, un framework para construir aplicaciones web con Python de forma sencilla y clara, cuyo objetivo principal es la representación de datos de forma personalizada. La herramienta será accesible vía web dónde se podrá interactuar con ella y visualizar los distintos análisis ejecutados.

JADX

Jadx es un popular descompilador de código abierto que se utiliza habitualmente para realizar ingeniería inversa de aplicaciones Android. Es capaz de descompilar el bytecode Dalvik compilado en un archivo APK y generar el código fuente Java correspondiente. Esto permitirá a la herramienta extraer los métodos Java del archivo APK e introducirlos en el modelo Transformer para su resumen.

Jadx es conocido por su capacidad para descompilar código complejo y fuertemente ofuscado, que es común en muchas aplicaciones de Android. Esto significa que la herramienta será capaz de proporcionar resúmenes precisos y completos incluso de las aplicaciones más complejas.

Además, Jadx es de código abierto y se mantiene activamente, lo que significa que se actualiza y mejora constantemente para soportar las últimas versiones de Android y manejar código cada vez más complejo. Esto lo convierte en una opción fiable y robusta para su uso en una herramienta de análisis de aplicaciones Android.

3.2 Diseño y desarrollo

El diseño y desarrollo de la aplicación se puede dividirse en módulos diferentes. La separación del proyecto en estos módulos permite un enfoque más modular y flexible para la construcción de la aplicación, y hace que sea más fácil de gestionar, comprender y mantener el proyecto en el tiempo.

- El primero de ellos corresponde con el módulo de interfaz gráfica. Este módulo se encarga del diseño e implementación de la interfaz de usuario de la aplicación. Permite a los usuarios seleccionar archivos y cargarlos, visualizando posteriormente los resultados de los análisis realizados por CodeT5 y T5. Este módulo es implementado utilizando el framework Dash, que permite crear rápida y fácilmente aplicaciones web interactivas con Python.
- El segundo módulo es el de análisis: Este módulo se encarga de implementar la funcionalidad de la aplicación, referida al análisis de aplicaciones Android utilizando jadx, codeT5 y T5. Este módulo integra estas herramientas y librerías, y proporcionar el código necesario para extraer los métodos y clases Java de la app descompilada, y genera resúmenes utilizando codeT5 y T5.

Es importante destacar la actividad asociada a la investigación bibliográfica. Mientras esta no puede ser considerada un módulo de la aplicación, es una de las bases de esta y abarca todas las tareas relacionadas con la recopilación, organización y análisis de la información necesaria para poder implementar la herramienta. Se puede dividir en partes según el tema tratado, desde la extracción del código de las aplicaciones Android, pasando por las técnicas de procesamiento de lenguaje para analizar las aplicaciones Android (tanto de lenguaje natural como de código fuente). Y finalmente con el desarrollo de la interfaz de usuario de la herramienta.

Fases del desarrollo

Una vez definidas los módulos más importantes de la aplicación, podemos entrar a desglosar las distintas tareas o fases que se han realizado o seguido en función de los distintos módulos, para el correcto desarrollo de este trabajo.

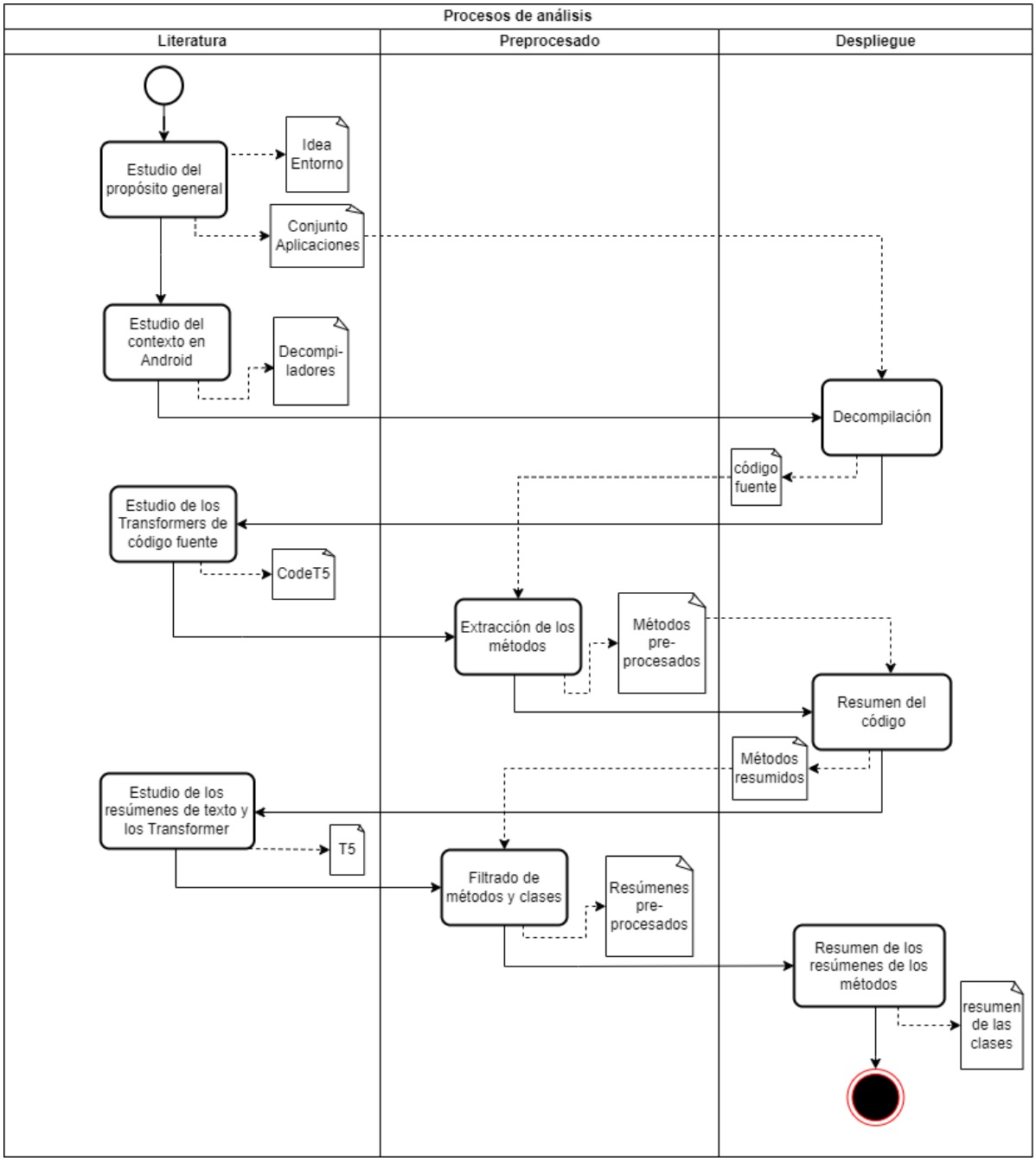


Figura 3.1: Diagrama que representa las fases que se han realizado para la implementación del módulo de análisis

En primer lugar, las fases correspondientes al módulo de análisis, ver 3.1:

1. Estudio de bibliografía relacionada con el análisis de malware en Android mediante herramientas de aprendizaje automático para la definición del propósito general del proyecto.

2. Selección del entorno y lenguaje de programación: una vez definido el propósito general del trabajo, se define un entorno de desarrollo así como el lenguaje de programación utilizado, Python.

3. Búsqueda y análisis de información relacionada con el funcionamiento básico de Android y con las distintas herramientas existentes de decompilación de aplicaciones para extraer el código Java de estas.

4. Despliegue de las herramientas seleccionadas sobre varias aplicaciones muestra para poder comprender su funcionamiento y calidad, seleccionando así la más apta para nuestra tarea. Finalmente, se opta por Jadx.

5. Estudio de la literatura relacionada con los distintos modelos de procesado de lenguaje. Primeramente, se comprende el funcionamiento y partes básicas de la arquitectura Transformer[1] para posteriormente realizar un análisis de los modelos publicados sobre código fuente y seleccionar así el utilizado en nuestra herramienta, CodeT5.

6. Despliegue de una pipeline que combine la salida del decompilador con CodeT5 para poder probar la arquitectura sobre las aplicaciones muestra. Esta fase incluye el ajuste de la salida de Jadx de manera que se pueda hacer uso óptimo de CodeT5.

7. Estudio del campo de resúmenes de texto mediante el uso de arquitecturas Transformer. Tras haber comprendido las distintas ventajas de cada modelo, se dispone a seleccionar el más adecuado para nuestra aplicación, T5.

8. Adaptación de la pipeline anteriormente mencionada para incluir el resumen del resumen de métodos generado por CodeT5. Se prueba su funcionamiento con las aplicaciones muestra y se realiza un procesado sobre la salida de CodeT5 para que sea una entrada apta y óptima para el modelo T5.

A continuación detallaré algunas tareas de adaptación de las distintas entras o salidas de los modelos.

Conjunto de datos utilizado

A lo largo del desarrollo de la herramienta se ha ido probando su funcionamiento con diversas aplicaciones para asegurar el correcto funcionamiento de esta. Para ello hemos utilizado tanto aplicaciones que explotan vulnerabilidades, malware, como aplicaciones con un funcionamiento benigno, benignware.

Como conjunto de malware nos hemos basado en un dataset relativamente pequeño, CICAndMal2017 [74], formado por muestras de distintas familias de malware que pueden ser clasificadas en cuatro categorías:

- En primer lugar, muestras de Adware, se tratan de aplicaciones cuyo comportamiento malicioso se basa en introducir publicidad en el dispositivo instalado.
- En segundo lugar, muestras de Ransomware, las cuales son aplicaciones basadas en encriptar la información del teléfono para luego pedir un rescate para su desencriptado. Versiones modernas de este malware incluyen él envió de una copia de los datos al atacante.
- En tercer lugar, muestras de Scareware, una forma de malware que a través de ingeniería social trata de manipular a los usuarios para que adquieran software no deseado, ya sea de pago o gratuito, que puede contener más malware.
- Por último, muestras de SMS malware, conocido también como smishing, es un tipo de malware que se aprovecha del servicio del SMS del dispositivo para o bien introducir malware, filtrar información del dispositivo al atacante o plantear una estrategia de phising.

Este dataset ha sido seleccionado ya que contiene un conjunto pequeño de muestras y en nuestra herramienta únicamente necesitamos una pequeña muestra para realizar comprobaciones de su funcionamiento.

Por otra parte, como hemos comentado anteriormente, hemos utilizado benignware extraído de distintas páginas de internet como APKMirror o apkpure para probar su funcionamiento en aplicaciones conocidas de mayor tamaño, como pude ser *playstore*, *spotify*...

Jadx

La herramienta para extraer el código Java de una app, Jadx, tiene diversas opciones que aprovecharemos para aumentar la calidad de los métodos proporcionados al modelo. En primer lugar, eliminaremos los comentarios automáticos con la flag *–comments-level none*. Por otra parte, marcaremos la opción *–deobf* para que active la deofuscación de código, la cual sirve para tratar de revertir la posible ofuscación de código que se utiliza para hacer menos legible el código. De esta forma los nombres de métodos, clases y variables tendrán mayor sentido, facilitando su entendimiento para CodeT5.

Por último, se marca la flag *–log-level info* para que nos indique detalles básicos de la decompilación, como puede ser el número de clases o métodos y los errores que han podido ocurrir, los cuales serán mostrados en el front end.

Procesado de métodos

El modelo CodeT5 ha sido entrenado con funciones de código, por lo que extraemos los métodos individuales de cada clase antes de proporcionar el código a CodeT5. Esto puede ayudar a CodeT5 a comprender la estructura y organización del código de forma más eficaz, y permitirle proporcionar resúmenes más precisos de los métodos y clases individuales.

La extracción es realizada mediante el control de expresiones regulares, teniendo en cuenta la siguiente estructura: inicio de una línea con texto, seguido de apertura y cierre de paréntesis (con texto o no en su interior), seguido de una apertura de corchetes e inmediatamente un salto de línea (cabecera del método), seguido de texto (cuerpo del método) contando con la posibilidad del uso de corchetes y finalizando con un salto de línea seguido de un corchete de cierre delimitando el final de la función, esto es, la estructura típica de los métodos de Java generados por Jadx.

Por otra parte, se ha detectado una mayor calidad en los resúmenes de los métodos en los que se eliminan los saltos de línea, por lo que antes de ser proporcionados a CodeT5 se eliminarán.

Además, una de las limitaciones del Transformer en el que se basa CodeT5, T5, es que el número de elementos de entrada es limitado, lo cual provoca que en ocasiones, se produzcan errores a la hora de proporcionar los métodos al modelo. Estos errores pueden ser provocados por la declaración de variables demasiado largas o por sentencias return demasiado complejas, entre otros. Es por esto, que existe un filtro basado en la longitud de línea, de tal manera que cuando la longitud sea superior a 200 caracteres, se trunque la línea a los primeros 25 y se añada un comentario repetido tres veces al final de la línea *"#Long declaration*. Además, se añadirá por cada línea truncada un comentario en el código fuente del método con *This line has been shortened keeping the first 25 characters* para que los usuarios de la herramienta sepan que se ha recortado. El valor de 150 caracteres ha sido elegido de forma arbitraria, basándose en la guía de Java de Google [75] que recomienda un límite de 100 caracteres por línea.

Otra de las causas de estos errores puede ser que el método sea demasiado largo, por lo que se ha decidido truncar la entrada a los métodos cuya longitud sea mayor de 2000 caracteres, lo cual equivale aproximadamente a los 512 tokens de límite de CodeT5, ya que no es posible saber con exactitud cuántos tokens tendrá cada entrada. Tampoco es viable dividir el método cuando exceda los 2000 caracteres ya que el modelo será incapaz de comprender los métodos divididos. Cuando

se trunque un método se añadirá a la cabecera el siguiente texto *This method has been shortened to the first 2000 characters* para que el usuario que analiza la aplicación lo tenga en cuenta.

Toda la información derivada de los métodos de la aplicación como su ruta (que incluye su clase), su código fuente y su resumen son exportados a un fichero csv con el nombre de la aplicación android analizada.

CodeT5

La configuración del modelo CodeT5 utilizado para el resumen del código fuente de los métodos es el recomendado en la propia página de Github de CodeT5 [76], en el que se utiliza como Tokenizer una versión pre-entrenada por ellos del RobertaTokenizer y hemos elegido como framework PyTorch, así como fijado la longitud máxima de los resúmenes en 20 tokens para evitar perder información en los métodos más largos.

Resumen del código fuente

En primer lugar, tras resumir el código fuente de los métodos se analizó la calidad de los mismos, observando que en algunos casos, CodeT5 producía salidas erróneas con palabras o secuencias de palabras repetidas en bucle. Es por esto que decidimos limpiar los métodos eliminando las apariciones de palabras consecutivas antes de exportarlos en el fichero csv. No obstante, no fue posible eliminar la repetición de secuencias de palabras de mayor longitud en los resúmenes

Se han desarrollado estadísticas de los resúmenes de los métodos generados por CodeT5 agrupando por clase (usando la ruta del método como id) de las aplicaciones de muestra para comprender los datos obtenidos y poder decidir estrategias para realizar un preprocesado óptimo. Con estas estadísticas podemos apreciar como un gran porcentaje de las clases (entendidas como la agrupación de los resúmenes de los métodos que las componen) no necesitan resumirse ya que tienen longitudes menores a dos líneas (250 caracteres aproximadamente). Esta estadística es coherente con los principios de diseño de software limpio, según los cuales, la longitud de los métodos debe ser corta, de unas 20 líneas sería ya considerado largo [77].

Por otra parte, se observa un problema: la existencia de clases con tantos métodos que el total de sus resúmenes supera los 512 tokens de entrada del modelo T5. Se trataron distintas soluciones para abordar este problema, desde el uso de modelos NLP de mayor entrada como BART [49] (1024 tokens) o LongT5 [78] (16.384 tokens). El problema de BART es que no termina de solucionar el problema, ya que seguirían existiendo clases que sobrepasan este límite de tokens. Sin embargo, LongT5 sí que solucionaría el problema, su uso tiene un coste a tener él cuenta: el lar-

go crecimiento en memoria al tratarse de un modelo muy grande. Finalmente, concluimos que al existir un número muy bajo de clases que exceden los 512 tokens (menos del 5% en la mayoría de las aplicaciones de muestra), no merece la pena asumir este coste. En su lugar, las clases serán divididas en n partes iguales cuyo tamaño será inferior al límite en tokens para posteriormente ser resumidas por T5 y estos resúmenes concatenados en un resumen final.

Las estadísticas generadas hacen referencia al número de métodos por clase, la longitud de los resúmenes de estos métodos, así como la longitud de su tokenización y el número de carpetas en el que están almacenadas las clases desde el directorio padre *sources*.

Tanto las estadísticas generadas como los resúmenes de las clases han sido almacenados en distintos ficheros csv.

T5

Para poder utilizar T5 en nuestra herramienta, nos hemos apoyado en la implementación que proporciona HugginFace en su librería transformer del método pipeline, en el cual se especifica la tarea objetivo, en este caso resumir (summarization) y el modelo y tokenizer utilizados, en este caso se ha optado por el t5-large al proporcionar mejores resultados que el base y no tener un tamaño desmedido. El framework seleccionado es, de nuevo, PyTorch.

En segundo lugar, las fases correspondientes al módulo de interfaz de usuario:

1. Análisis de los frameworks de Python para desarrollo de interfaces de usuario y posterior selección de Dash para desarrollar el front-end.
2. Implementación de la interfaz gráfica de usuario de nuestra herramienta, con la correspondiente adaptación del código para el correcto funcionamiento de esta.
3. Testeo del funcionamiento de nuestra herramienta con distintos tipos de aplicaciones Android.

Front-end

Finalmente se decide trabajar con el framework Dash debido a varios factores. En primer lugar, Dash está construido sobre la biblioteca Plotly, que proporciona una amplia gama de herramientas para entre otros, la visualización de datos y la gestión de elementos interactivos. Además, Dash utiliza una sintaxis declarativa que facilita la creación de visualizaciones complejas sin tener que

entender profundamente conceptos relacionados con la creación de elementos dinámicos, reduciendo las líneas de código.

Además, se ha optado por apoyarnos en la librería dash-bootstrap-components [79]. Dash-bootstrap-components es una librería de componentes para Dash, diseñada para integrarse con el framework CSS Bootstrap. Bootstrap es un popular marco de front-end que proporciona un conjunto de estilos CSS y clases que se pueden utilizar para crear rápida y fácilmente interfaces de usuario con un aspecto coherente. Utilizando dash-bootstrap-components, puedes aprovechar las ventajas de Bootstrap en Dash, y crear interfaces de usuario fáciles de personalizar. La biblioteca incluye una serie de componentes, incluyendo botones, formularios, tarjetas, y más, que se pueden utilizar para construir aplicaciones interactivas y sensibles con Dash.

Aplicación web

La interfaz de usuario está dividida en tres componentes principales que se integran a su vez con la lógica de análisis proporcionando así toda la funcionalidad de la herramienta. Destacar la existencia de otro componente de menor relevancia, *About*, en el cual se realiza una breve descripción del proyecto, autor y se indexan sitios útiles de análisis de malware en Android.

A continuación se describe el funcionamiento de cada una:

Home

El componente Home es un componente básico del front-end de la herramienta. Es responsable de proporcionar al usuario la posibilidad de subir a la web una aplicación Android para que sea analizada por la herramienta. Esto implica tener un elemento del tipo *dcc.Upload*, que proporciona un área de arrastrar y soltar, así como un selector de archivos, que permite al usuario seleccionar la aplicación que se va a analizar.

Una vez seleccionada la aplicación, el componente Home se encarga de gestionar el proceso de almacenamiento. Esto implica la validación del fichero para garantizar que se trata de una aplicación Android válida (.apk), enviando un mensaje de error en caso de que no sea posible validar el fichero. Por último, aparece un botón en pantalla para continuar con el análisis, lanzando el siguiente componente ver 3.2.

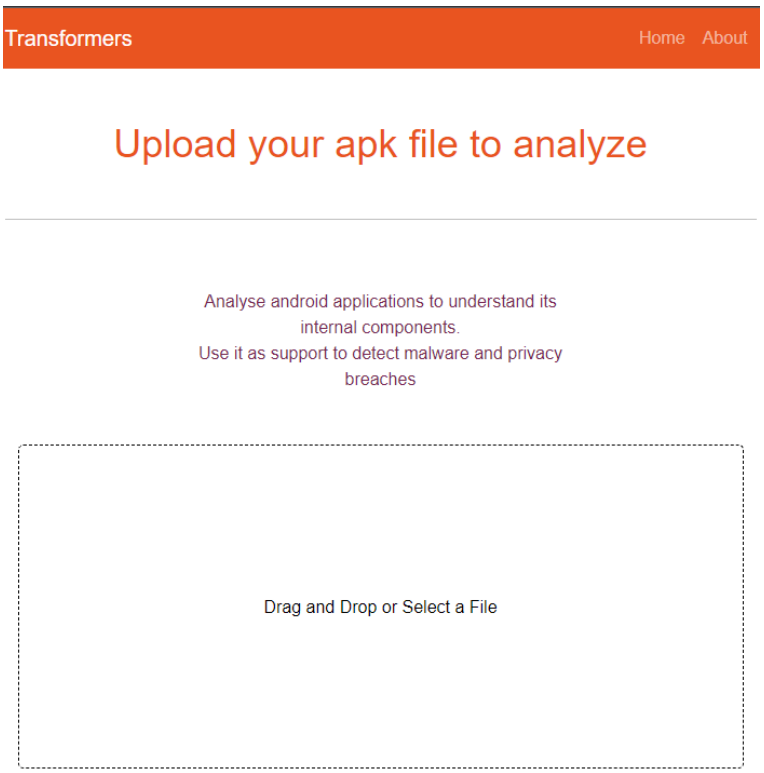


Figura 3.2: Interfaz gráfica del componente home de la herramienta

Analyzer

El componente analizador es un componente clave del front-end de su herramienta. Es responsable de utilizar la lógica de análisis del back-end que computa la decompilación y los resúmenes de las aplicaciones Android utilizando CodeT5 y luego T5.

Para esto, se ofrece un botón que al pulsarlo, inicia una llamada al módulo del back-end encargado de realizar la decompilación de la aplicación con Jadx, denominado *Decompile*, almacenando la salida de Jadx en un fichero, la cual es mostrada a continuación, con el propósito de visualizar estadísticas y errores. A continuación, se le ofrece al usuario la posibilidad de iniciar el análisis con un botón, tras presionarlo, se llama al módulo del back-end encargado de realizar los resúmenes del código fuente de los métodos de la aplicación, llamado *CodeT5*, para lo cual se crea un hilo iniciando un subproceso. Este módulo almacena el número de métodos sobre el que itera en un fichero, gracias a lo cual es posible mostrar por pantalla el progreso de la aplicación en una barra de progreso (componente dbc.Progress). Una vez finalizada su ejecución, este componente crea otro hilo, el cual llama a otro módulo del back-end, encargado de la realización de los resúmenes por clases, denominado *T5*, el cual, de la misma forma que el anterior, almacena el número de clases sobre las que itera en un fichero, posibilitando la existencia de otra barra de progreso que

muestre por pantalla el avance de este módulo ver 3.3.

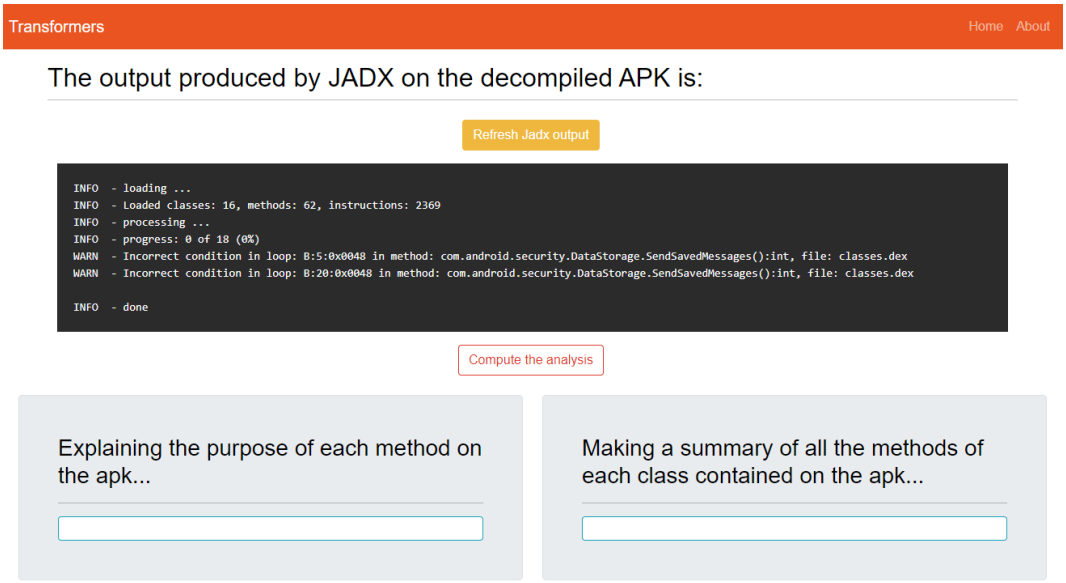


Figura 3.3: Interfaz gráfica del componente analyzer de la herramienta

Una vez terminan los procesos del back-end que generan los resúmenes, se lanza un modal con un botón para continuar al componente con los resultados.

Results

El componente de results es una parte del front-end que se encarga de mostrar los resultados de los resúmenes computados por la herramienta. Estos resultados son presentados a través de una estructura con dos ventanas, gracias al componente dbc.Tabs.

En la primera de ellas, *Classes of the App*, se podrá visualizar en formato de tabla el csv generado por el módulo T5 que contiene los resúmenes de las clases junto con su nombre y path, pudiendo seleccionar una y copiar su contenido ver 3.4. En la otra ventana, *Query a class*, se podrá realizar una consulta sobre una clase, preferiblemente copiada de la ventana mencionada anteriormente, y visualizar los detalles de esta como su resumen, los métodos que la componen y los resúmenes computados de estos por CodeT5 ver 3.5.

La siguiente figura muestra un diagrama de secuencia que representa los pasos fundamentales seguidos por la web y por tanto la herramienta, para que un usuario obtenga el análisis completo de su aplicación.

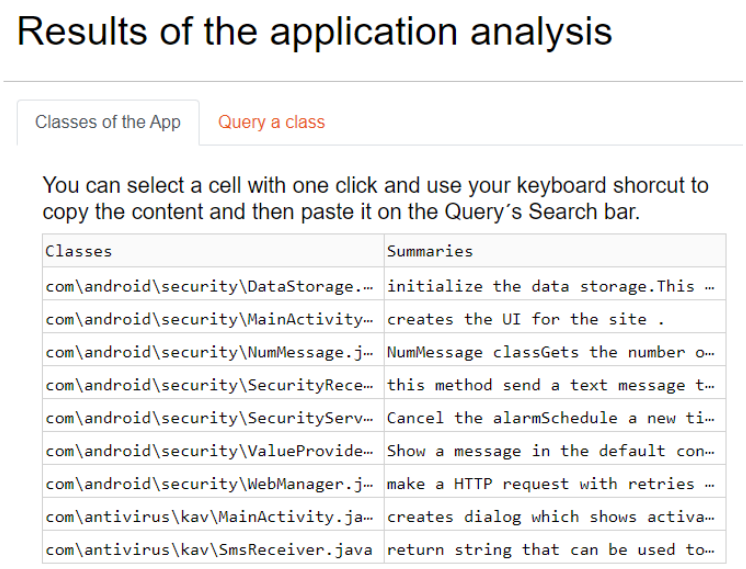


Figura 3.4: Interfaz gráfica que muestra las clases resumidas en el componente results de la herramienta

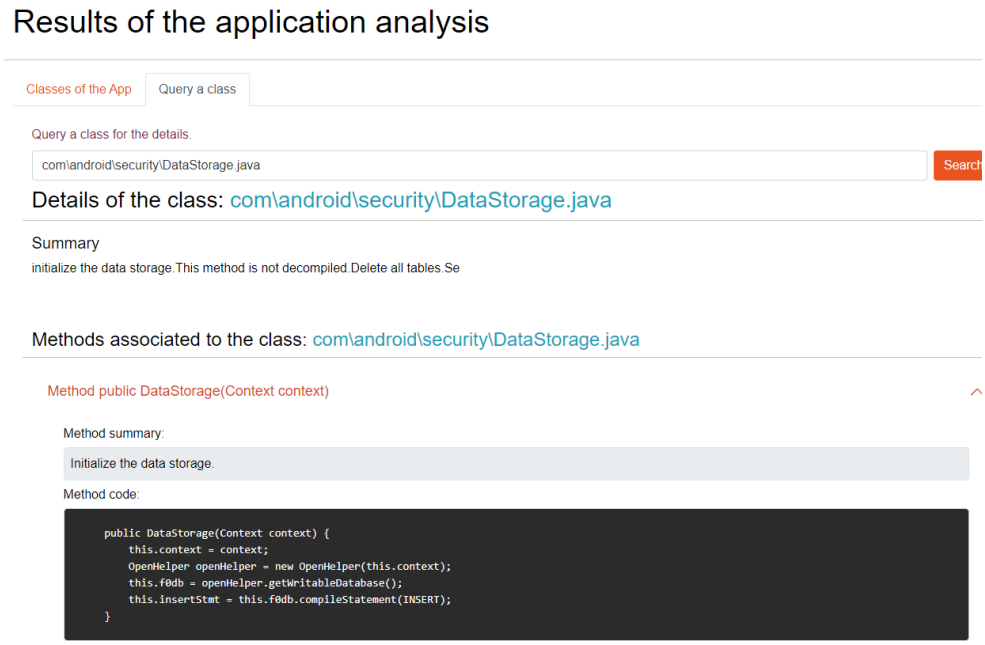


Figura 3.5: Interfaz gráfica que muestra la clase objetivo y sus métodos resumidos en el componente results de la herramienta

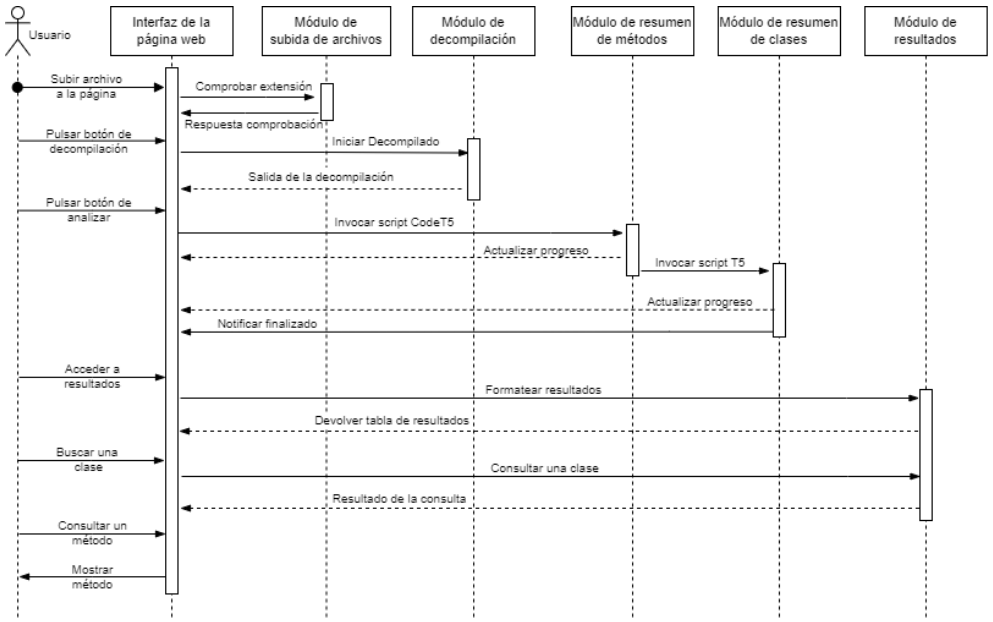


Figura 3.6: Diagrama de secuencia de una ejecución normal de la herramienta en la web

4. Experimentación y análisis de resultados

En esta sección se mostrará un análisis de una ejecución normal de la aplicación así como distintas pruebas que se han realizado sobre las aplicaciones de muestra mencionadas anteriormente y las limitaciones que han ido apareciendo que hemos superado mediante distintas estrategias de preprocesado.

4.1 Análisis de una aplicación: SMSsniffer

Para realizar este análisis hemos escogido una aplicación de tamaño reducido por simplicidad, en concreto una de SMS malware [74], de la familia de SMSsniffer. La aplicación tiene el hash SHA-256: 99621de457d2ff5d192cd7b27f64f3c7ad64aab2e60ad22610076850aaa2828c.

Una vez analizada, podemos ver, en el módulo de resultados, como la aplicación tiene 7 clases y se muestran sus resúmenes correspondientes, como podría ser el de la clase:

"com\android\security\SecurityService.java"

"Cancel the alarmSchedule a new time period for security.Invoked".

Si consultamos la clase en la herramienta podemos ver que tiene 6 métodos y los resúmenes de cada uno de ellos. Un ejemplo puede ser la imagen 4.1.

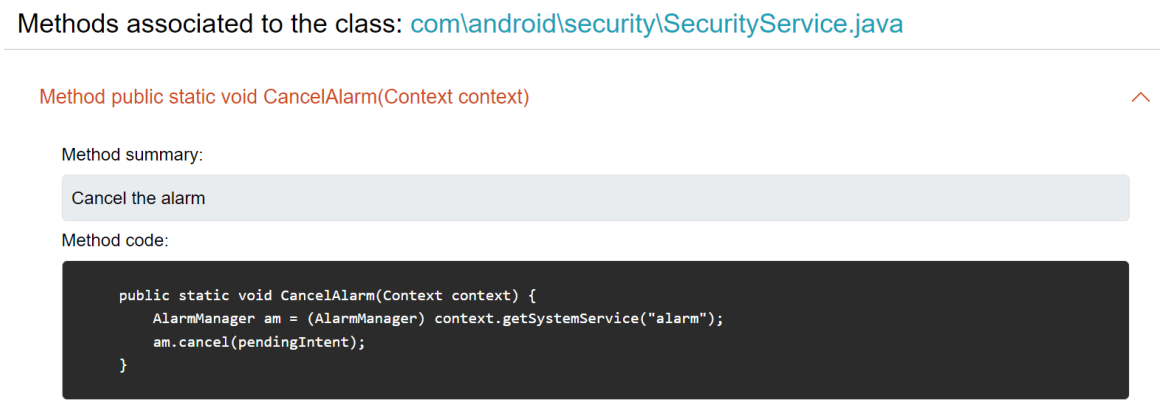


Figura 4.1: Primer método de la clase SecurityService.java de la apk analizada

4.2 Sobre los resúmenes de CodeT5

Tratado del formateo

Como hemos comentado anteriormente en el apartado 3.2, proporcionar los métodos con indetación correcta, esto es, con saltos de línea, provoca que CodeT5 tenga un peor rendimiento en términos generales a la hora de resumir su funcionamiento. Esto puede ser visualizado, entre otros, en el siguiente ejemplo, extraído después de analizar todos los métodos existentes en la aplicación de Spotify (spotify.apk):

/androidx/transition/C1125i0.java	public static void m60355c(View view, Matrix matrix) { f5065a.mo60337c(view, matrix); }	Create a new matrix that stores the matrix in the view.	----------
/androidx/transition/Visibility.java	C1097a(View view, int i, boolean z) { this.f4995a = view; this.f4996b = i; this.f4997c = (ViewGroup) view.getParent(); this.f4998f = z; m60413a(true); }	Create a class entry for C1097a.	----------
/androidx/transition/Transition.java	public Animator mo49080a(ViewGroup viewGroup, C1157x c1157x, C1157x c1157x2) { return null; }	Create a mo49080a animation.	Create a new object with a custom view group and a custom custom animation.
/androidx/transition/C1159z.java	public C1159z(View view, View view2, int i, int i2, float f, float f2) { this.f5136b = view; this.f5135a = view2; this.f5137c = i - Math.round(view.getTranslationX()); this.f5138f = i2 - Math.round(this.f5136b.getTranslationY()); this.f5142l = f; this.f5143m = f2; int[] iArr = (int[]) this.f5135a.getTag(C1135m.transition_position); this.f5139i = iArr; if (iArr != null) { this.f5135a.setTag(C1135m.transition_position, null); } }	PUBLIC METHODS This constructor initializes the Earth - Householder objects.	----------
/androidx/transition/TransitionSet.java	C1095a(TransitionSet transitionSet, Transition transition) { this.f4991a = transition; }	DESCRIPTION C1095a - Class name.	----------
/androidx/transition/TransitionSet.java	@Override public Transition mo60443a(View view) { for (int i = 0; i < this.f4986M.size(); i++) { this.f4986M.get(i).mo60443a(view); } this.f4964j.add(view); return this; }	Add a view to the list of tables that make up the VISA.	Add a view to the sequence of transition identified by tag - type sequence sequence sequence sequence sequence
/androidx/transition/TransitionSet.java	@Override public void mo60442a(ViewGroup viewGroup) { super.mo60442a(viewGroup); int size = this.f4986M.size(); for (int i = 0; i < size; i++) { this.f4986M.get(i).mo60442a(viewGroup); } }	Add the EKB to the list of EKBs for the given view group.	Add the NCBI - A sequence of all the sequence sequence sequence sequence sequence sequence sequence

Figura 4.2: La primera columna corresponde a la clase del método, la segunda columna contiene el código fuente del método, la tercera columna contiene el resumen del método habiendo eliminado los saltos de línea y la cuarta el resumen del método sin alterar.

Como puede observarse en 4.2, los resúmenes de la tercera columna, la cual es la que utilizamos en nuestra herramienta, contiene resúmenes de mayor calidad. Además, aprovechamos esta tabla

para comentar un hecho recurrente al resumir métodos con CodeT5: la repetición de palabras.

```
False    146923
True       2694
Name: Total_Repetitive_Clean, dtype: int64
False    136836
True      12781
Name: Total_Repetitive, dtype: int64
```

Figura 4.3: Muestra el total de resúmenes de métodos de la aplicación analizada que tienen palabras repetidas consecutivamente, representado por True. Las primeras cifras corresponden con los métodos que han sido preprocesados (Total Repetitive Clean) mientras que las segundas cifras los que no han sido (Total Repetitive).

Para poder confirmar que no hemos elegido los métodos de la tabla arbitrariamente, hemos calculado el número de resúmenes que repiten palabras de forma consecutiva en la tercera columna y en la cuarta columna ver 4.3. Obteniendo una clara diferencia que nos hace decantarnos por utilizar los resúmenes de la tercera columna, es decir, eliminar los saltos de línea.

/androidx/appcompat/view/menu/MenuItemC0287j.java	@Override public Drawable getIcon() { return this.f1621d.getIcon(); }	Returns the icon of the icon icon of the icon icon of the icon icon of the icon	Gets the icon of the
/p000/C21013j4.java	public long m19149b() { View view = this.f73001a.get(); if (view != null) { return view.animate().getDuration(); } return 0L; }	Returns the duration of the fade in in milliseconds.	This method returns the duration of the action in milliseconds.
/androidx/drawerlayout/widget/DrawerLayout.java	public LayoutParams(int i, int i2) { super(i, i2); this.f3157a = 0; }	Sets the layout layout parameters.	LayoutParams method.
/p000/o4a.java	@Override public float getInterpolation(float f) { double d = f; Double.isNaN(d); double pow = Math.pow(2.0d, (-10.0d) * d); Double.isNaN(d); return (float) ((Math.sin(((d - 0.1875d) * 6.283185307179586d) / 0.75d) * pow) + 1.0d); }	Calculate the interpolation factor for a logarithmic logarithmic logarithmic logarithmic	This method returns the interpolation of the tag.

Figura 4.4: Muestra los resúmenes en los que es interesante eliminar las palabras repetitivas, sigue la misma estructura que 4.2.

Tratado de elementos repetitivos

Por otra parte, estas cifras nos hacen plantearnos la eliminación de este tipo de eventos en resúmenes, ya que no tienen ningún tipo de coherencia. Es cierto, que en algunas ocasiones la calidad de los resúmenes ya es deficiente de por sí, no obstante, la aparición de estas palabras no es necesaria y en ocasiones mejora la calidad del resumen ver 4.4.

Tratado de líneas demasiado largas

Otro de los preprocesados aplicados a los métodos, como se describe en 3.2 es la eliminación de líneas demasiado largas, las cuales suponen un problema para CodeT5 al poder propiciar proble-

mas tanto a nivel de comprensión de código como en longitud de entrada aceptadas. Ejemplos de estas líneas son las siguientes:

```
int m58627a = byd.m58627a(this.f74696D0.widthPixels, 3, m6172900
    ().getDimensionPixelSize(m79.radio_cover_cat_height), byd.
    m58582b(64.0f, m6172900()), m6172900().
    getDimensionPixelOffset(m79.radio_cover_items_cat_right_gap),
     1.0f);
this.f63503f = new C19558ff(String.format("%s$%d", this.f9294b,
    7002), String.format("%s$%d", this.f9294b, 7002), String.
    format("%s$%d", this.f9294b, 7002), String.format("%s$%d",
    this.f9294b, 7002));
byte[] icon = ((abstractC24039ug instanceof AbstractC19800gg) ||
    (abstractC24039ug instanceof AbstractC19339eg) || (
    abstractC24039ug instanceof AbstractC20256ig)) ? ((
    AbstractC19800gg) abstractC24039ug).getIcon() : null;
Intent intent2 = new Intent("android.intent.action.VIEW", Uri.
    parse("https://play.google.com/store/apps/details?id=com.waze
    &referrer=utm_source%3Dpartner%26utm_medium%3Ddirect%26
    utm_campaign%3Dspotify"));
return new dy5(this.f64258a.get(), this.f64259b.get(), this.
    f64260c.get(), this.f64261d.get(), this.f64262e.get(), this.
    f64263f.get(), this.f64264g.get(), this.f64265h.get(), this.
    f64266i.get());
return m18942a(this.f73336a.get(), this.f73337b.get(), this.
    f73338c.get(), this.f73339d.get(), this.f73340e.get(),
this.f73341f.get(), this.f73342g.get(), this.f73343h.get(), this.
    f73344i.get());
```

Tratado de métodos demasiado largos

Otro de los problemas que está relacionado con el anterior es la existencia de métodos largos, los cuales serán recortados como se detalla en 3.2. Ejemplo de estos métodos puede ser 4.1:

Listado 4.1: Ejemplo de método que será largo recortado

```
final String[] sorter(String s, MyFont font2, int txt_w, boolean
    width_en, boolean clr_spc) {
        char ch;
        int s_length = s.length();
        String[] buf = new String[s_length];
        int ww = 0;
        int w = 0;
        int ksim = 0;
        int n_str = 0;
        int str_s = 0;
        int str_e = 1;
        int str_s_e = 0;
        int i = 0;
        while (i < s_length + 1) {
            if (i >= s_length) {
                ch = '\n';
            } else {
                ch = s.charAt(i);
            }
            int sim = ch;
            int wm = width_en ? font2.charWidth(ch) : 5;
            if (ch != '\n' && ch != '\r') {
                w += wm;
                ww += wm;
            }
            switch (sim) {
                case 10:
                case Constants.RMS_store_recordID_SubscribtionMode:
                    str_e = i;
                    w = txt_w + 1;
                    ww = 0;
                    if (i < s_length) {
                        int i2 = i + 1;
                        if (i2 < s_length) {
                            char cho = ch;
                            char ch2 = s.charAt(i2);
```

```
                            if ((ch2 == '\n' || ch2 == '\r') && ch2 !=
                                cho) {
                                i2++;
                            }
                        }
                        str_s_e = i2;
                        i = i2 - 1;
                    } else {
                        str_s_e = s_length;
                    }
                    ch = '\n';
                    break;
                case 32:
                    str_e = i + 1;
                    ww = 0;
                    break;
            }
            if ((w <= txt_w || !width_en) && ch != '\n') {
                ksim++;
            } else {
                if (str_e == str_s && ksim > 0) {
                    str_e = str_s + ksim;
                    ww = 0;
                    if (i < s_length && i + 1 > str_e) {
                        ww = font2.stringWidth(s.substring(str_e, i + 1))
                            ;
                    }
>>>------- EL RESTO DEL MÉTODO HA SIDO RECORTADO --------------------
                }
                if (str_e > str_s) {
                    buf[n_str] = s.substring(str_s, str_e);
                } else {
                    buf[n_str] = "";
                }
                str_s = str_e;
                if (str_s < str_s_e) {
```

```
                    str_s = str_s_e;
                }
                str_e = str_s;
                w = ww;
                ksim = (i - str_e) + 1;
                if (clr_spc && buf[n_str] != null && buf[n_str].length()
                    > 0) {
                    while (buf[n_str].length() > 0 && buf[n_str].charAt(
                        buf[n_str].length() - 1) == ' ') {
                        buf[n_str] = buf[n_str].substring(0, buf[n_str].
                            length() - 1);
                    }
                }
                n_str++;
            }
            i++;
        }
        int kvo_str = n_str;
        String[] text = new String[kvo_str];
        System.arraycopy(buf, 0, text, 0, kvo_str);
        return text;
    }
```

4.3 Sobre los resúmenes de T5

Tratado del tamaño de los resúmenes

Existen dos tratamientos en función del tamaño de los resúmenes:

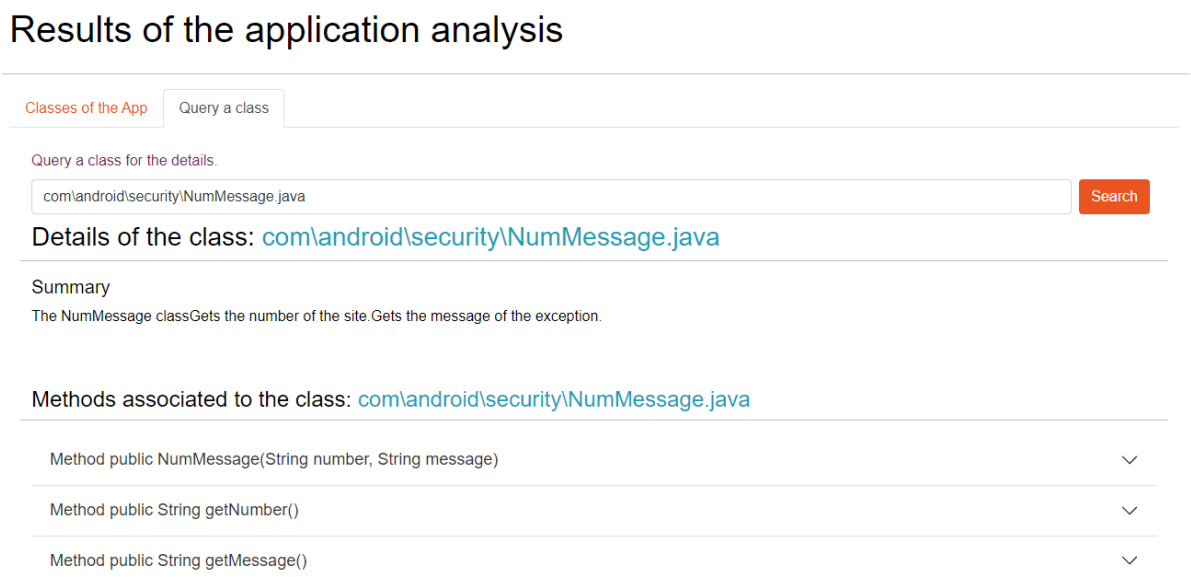


Figura 4.5: Como se puede observar, esta clase tiene 3 métodos cuyos resúmenes están incluidos directamente en "Summary", dónde se diferencian al comenzar cada uno con .

Cuando son de un tamaño inferior a dos líneas, como puede ser el ejemplo 4.5, no se introduce en T5 y simplemente se muestra ese resumen.

Listado 4.2: Resumen de la clase Registrator.Java

```
['The Registrator class.Check security.Returns the language of the
   current thread.Returns the LMC string.Returns the language code minus
    the language code of the current language.Checks if the LastModified
    field is valid.Checks if a SimCard exists.Sets the country code.
   Returns the countries list.Returns the country info.Returns the
   activation key for this session.This method return the ActivationKey.
   Returns the subscription key for the given subscription type.This
   method returns the S', 'ubscriptionKey for the given type and sms
   tail.Request a serial.Request a serial number.Request a subscription
   of the given type.This method requests a subscription of the
```

```
specified type to the specified smsTail.Checks if the given serial is
 a valid serial for the current registration type.Check Serial.This
method is used to test the ChSn of the RMS store.This method is used
to test the code of the RMS module.Returns true if the class is
registered.This method returns true if the us', 'er is registered in
the system.Get config mode.getConfigStatus - returns config status.
OTHERWISE METHODS FOR MEDIA ACCESSParse the string val.Returns the
test resource.Get the test case.Returns true if the site is
subscribed to the site.This method check if the current user is
subscribed to the current user.Returns the Subscribtion Mode of the
current language.Creates a new SubscribtionMode.Returns true if the
plugin is activated.Returns the key for the bonus type.Returns a
random k', 'ey for the bonus.This method returns the key of the bonus
.Request a bonus.Request a bonus.Request a bonus.This method request
a bonus.Checks if the given code is a bonus.This method send an SMS.
Returns the game link.Returns the game link title.Returns the title
for the more games.Returns the link to the more games page.Sends an
SMS to the user who is a friend.Send a friend sms.Gets the number of
times a user attempts to send a friend.Returns the expiration date of
 the resource.Retur', 'ns the registration type of the resource.Set
the registration type.Returns the activation key size.Returns the key
 size in bytes for the Bonus.Returns the current subscription key
size.Gets the key size of the subscription.Returns the version of the
 current registrator.Get the version of the Config.Returns the SMS
code version.Returns the number of seconds since the demo mode.Sets
the duration in minutes for demo mode.Returns the error code of the
exception.Returns the error text.']
```

No obstante, cuando el tamaño es mayor a 512 tokens, se divide en partes, como es el caso de 4.2, en el que se divide en 4 partes y se introduce en el modelo T5, obteniendo el resumen 4.6

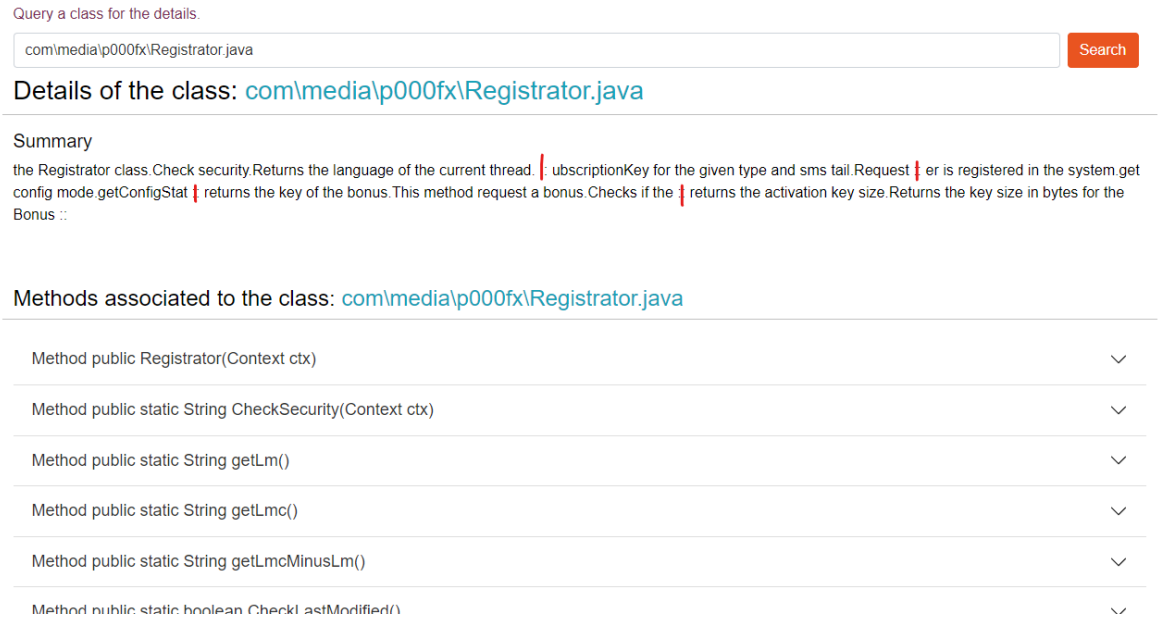


Figura 4.6: Se muestra el resumen obtenido tras haber recortado 4.2 en 4 partes y haberse introducido estas en el modelo T5.

5. Impacto social y aspectos éticos y profesionales

El desarrollo y uso de una herramienta para analizar aplicaciones Android puede tener importantes implicaciones sociales, éticas y profesionales. En este capítulo, exploraremos estas implicaciones con más detalle, y discutiremos cómo el diseño y el uso de la herramienta pueden optimizarse para minimizar cualquier consecuencia negativa.

En primer lugar, analizaremos las posibles repercusiones sociales de la herramienta, incluido su potencial para mejorar la calidad, seguridad y transparencia de las aplicaciones Android. Sin embargo, también consideraremos las posibles implicaciones sociales y éticas negativas de la herramienta, como su potencial para violar los derechos de propiedad intelectual o vulnerar la privacidad de los usuarios.

A continuación, examinaremos las implicaciones medioambientales y computacionales de la herramienta. Hablaremos de la huella de carbono de la herramienta y del impacto potencial de su uso en el rendimiento del dispositivo. También estudiaremos cómo pueden abordarse estas implicaciones en el diseño y uso de la herramienta.

Por último, exploraremos la cuestión de la responsabilidad profesional en relación con la herramienta. Consideraremos las obligaciones éticas de los desarrolladores de la herramienta y el papel de las organizaciones profesionales y las normas del sector en la promoción del uso responsable de la herramienta.

Una posible repercusión social de la herramienta de análisis de aplicaciones Android es su potencial para mejorar la calidad, seguridad y transparencia del ecosistema de aplicaciones. Al proporcionar a desarrolladores, investigadores y otras partes interesadas una mejor comprensión del

comportamiento y la funcionalidad de las aplicaciones, la herramienta podría ayudar a identificar y abordar posibles problemas, como vulnerabilidades de seguridad o contenidos inapropiados. Reduciendo así la cantidad de malware que podría afectar a los usuarios de Android, una gran masa de la sociedad.

Además, la herramienta podría utilizarse para ayudar a identificar aplicaciones que infrinjan las políticas o normas de la playstore. Por ejemplo, si se descubre que una aplicación contiene contenido malicioso o incurre en prácticas engañosas, la herramienta podría utilizarse para resumir de forma rápida y precisa el comportamiento de la aplicación, lo que permitiría una aplicación más rápida y eficaz de las políticas de la tienda de aplicaciones. Evitando así que usuarios que realizan un uso seguro del teléfono sean atacados. De esta manera, la herramienta podría contribuir a aumentar la confianza de los consumidores en las aplicaciones Android. Esto podría conducir a un aumento de las descargas y el uso de aplicaciones, impulsando la economía del desarrollo de apps y proporcionando beneficios económicos a los desarrolladores de aplicaciones y otras partes interesadas.

Por otra parte, las empresas y organizaciones podrían utilizar la herramienta para mejorar sus propios procesos de desarrollo y gestión de aplicaciones, principalmente a aquellas que utilizan metodologías ágiles. Al ofrecer un resumen rápido y preciso del comportamiento y la funcionalidad de las aplicaciones, la herramienta podría ayudar a las empresas a identificar posibles problemas y oportunidades, y a optimizar sus estrategias de desarrollo y gestión de aplicaciones. Esto podría mejorar el rendimiento de las aplicaciones y la satisfacción de los usuarios, lo que reportaría beneficios a las empresas, como mayores ingresos y fidelidad de los clientes.

En general, la capacidad de la herramienta para resumir con rapidez y precisión el comportamiento y la funcionalidad de las aplicaciones puede mejorar la calidad, la seguridad y la transparencia del ecosistema de aplicaciones. En última instancia, esto podría conducir a un ecosistema de aplicaciones mejor, más seguro y transparente para desarrolladores, usuarios y Stakeholders.

Sin embargo, también hay que tener en cuenta las posibles implicaciones sociales y éticas negativas de la herramienta. Por ejemplo, podría utilizarse para aplicar ingeniería inversa a aplicaciones patentadas, lo que podría vulnerar los derechos de propiedad intelectual. Además, podría utilizarse para analizar aplicaciones sin el conocimiento o consentimiento de sus creadores o usuarios, lo que plantearía problemas de privacidad.

Para resolver estos problemas, se podría plantear incluir en la aplicación de controles estrictos sobre el acceso a la herramienta, y proporcionar una orientación clara sobre el uso ético. Por ejemplo, se podría exigir a los usuarios que obtuvieran el permiso de los creadores de aplicaciones (en el

caso de las más conocidas) antes de utilizarla para analizar sus aplicaciones. Esto ayudaría a garantizar que los creadores de las aplicaciones son conscientes del análisis de sus aplicaciones y dan su consentimiento. Además, se podrían implantar controles estrictos de acceso a la herramienta para evitar usos no autorizados o poco éticos.

Por otra parte, más allá de las implicaciones sociales comentadas anteriormente, el desarrollo y uso de una herramienta basada en Transformer tiene potenciales implicaciones medioambientales, ya que es ampliamente conocida la significante huella de carbono que los modelos de aprendizaje profundo producen.

El entrenamiento y la ejecución de modelos de aprendizaje profundo pueden requerir importantes recursos computacionales, lo que puede dar lugar a un aumento de las emisiones de gases de efecto invernadero, contribuyendo al cambio climático y derivando en repercusiones negativas en el medio ambiente.

Para mitigar este impacto, se podría considerar el uso de fuentes de energía renovables para alimentar los procesos computacionales implicados. Por ejemplo, podrían utilizar energía solar para hacer funcionar los modelos de Transformer utilizados en la herramienta, reduciendo así su huella de carbono.

Además, más allá del diseño realizado, que ha tratado de que sea lo más eficiente posible, minimizando su coste computacional y reduciendo utilización de modelos cuando no es necesario podría plantearse la revisión y la implementación de algoritmos y estructuras de datos eficientes para minimizar las demandas computacionales de la herramienta.

En general, las posibles repercusiones sociales de esta herramienta de análisis de aplicaciones para Android son significativas y tienen implicaciones tanto positivas como negativas. Trataremos de garantizar que tenga un impacto positivo en la sociedad y se utilice de forma ética y responsable.

6. Conclusiones y trabajo futuro

Tras el desarrollo de este proyecto y haber expuesto los resultados obtenidos, podemos afirmar que la herramienta propuesta para el análisis de aplicaciones Android a través de una pipeline de machine learning con Transformers, es eficaz para resumir automáticamente la funcionalidad de las aplicaciones Android, pero la calidad de los resúmenes producidos no es la ideal.

En cuanto al nivel técnico, la herramienta fue capaz de decompilar con éxito las aplicaciones y de aprovechar eficazmente la potencia de la arquitectura Transformer para procesar el código Java y generar resúmenes. El uso de CodeT5 para extraer los resúmenes de los métodos Java y de T5 para resumirlos por clases permitió un proceso de análisis ágil y eficaz. Sin embargo, los resúmenes generados por la herramienta no siempre son de alta calidad y no siempre capturaban con precisión la funcionalidad de los métodos Java.

Una de las limitaciones detectadas a la hora de resumir los métodos tiene que ver con el decompilado y con las propiedades del Transformer CodeT5: el no poder trazar relaciones entre los distintos elementos de la aplicación.

CodeT5 es una arquitectura que carece de memoria al realizar tareas como es en este caso, el resumen de código fuente. Por otra parte, el decompilado de aplicaciones tiene todavía margen de mejora, principalmente, las tareas de deofuscado de código: por mucho que la aplicación pase por este proceso, en multitud de casos, este no es capaz de rescatar con precisión los distintos nombres de métodos, clases, variables o funciones. Estas dos características combinadas impiden que sea posible trazar relaciones precisas a la hora de analizar métodos que interactúan con recursos externos a ellos. Por ejemplo, analizando un método, X, que llama a otro, Y, sería posible entender el propósito de la llamada, en el caso de que su nombre tuviese relación con su funcionalidad, como podría ser si Y se denominase *obtenerCaminoOptimo()*. Sin embargo, si su nombre fuese *c0m23()* y no se tiene información contextual de la funcionalidad de este método (derivada de un análisis), ya que CodeT5 carece de memoria en este aspecto, es prácticamente imposible que deduzca el

propósito de la invocación de Y, afectando por tanto a la calidad del resumen del método objetivo, X.

Otra de las limitaciones derivada dependencia de la arquitectura Transformer, es que esta es conocida por requerir una gran cantidad de recursos computacionales, y en concreto de tarjetas gráficas. Esto puede limitar la eficacia de la herramienta en dispositivos con recursos limitados, como ordenadores sin tarjetas gráficas o smartphones. Además, el campo del procesado de código fuente necesita más recursos, investigación, desarrollo y menos competitividad empresarial. Como hemos visto en 2.16, existen modelos eficaces ya desplegados en producción cuya implementación no ha sido publicada, lo cual limita el avance científico del campo que nos permitiría mejorar la calidad de los modelos de código, incorporando características como una breve memoria (como la vista en ChatGPT [53]) que nos permita trazar relaciones entre los métodos. O mejoras en el diseño de las arquitecturas que permitan comprender mejor las estructuras inherentes al código.

En general, la herramienta propuesta se muestra prometedora como instrumento para analizar y resumir aplicaciones Android. Tiene el potencial de proporcionar a desarrolladores y usuarios información valiosa sobre la funcionalidad de las aplicaciones Android, y podría perfeccionarse y mejorarse en futuros trabajos.

6.1 Trabajo futuro

Hay varias direcciones que el trabajo futuro podría tomar con el fin de mejorar la herramienta propuesta. Una posible dirección es centrarse en mejorar la calidad de los resúmenes generados por la herramienta. Esto podría implicar la revisión de nueva literatura para encontrar arquitecturas que se adapten mejor a la tarea de resumen.

Una posible dirección podría ser centrarse en el de-ofuscado de las clases de Java generadas, buscando nuevas herramientas en la literatura, como podría ser la aplicación de un modelo Transformer para el de-ofuscado de código, lo cual implicaría una mayor comprensión del código por parte del modelo CodeT5 y por lo tanto mejores resúmenes de los métodos y de las clases.

Otra posible dirección para el trabajo futuro es hacer que la herramienta sea más eficiente y eficaz para su uso en dispositivos con recursos limitados, utilizando arquitecturas más pequeñas como es el caso de los modelos destilados (modelos que han seguido el proceso de transferencia de conocimientos de un modelo grande a otro más pequeño.). Otro camino podría ser obviar el análisis de clases con funcionalidad base en Android, elaborando para ello algún tipo de estrategia , por ejemplo, apoyarse en técnicas de hashing sensible a la distancia.

Por otra parte, se podría mejorar la aplicación web añadiendo funcionalidades de otras herramientas de análisis de aplicaciones Android como podría ser VirusTotal.

En general, existe un potencial significativo para un mayor desarrollo y mejora de la herramienta propuesta para el análisis de aplicaciones Android utilizando la arquitectura Transformer. Centrándose en la mejora de la calidad y la eficiencia de los resúmenes generados, ampliando las capacidades de la herramienta y abordando los desafíos potenciales, puede ser posible crear una herramienta potente y eficaz para analizar aplicaciones Android.

Bibliografía

[1] A. Vaswani, N. Shazeer, N. Parmar et al., «Attention is all you need», *Advances in neural information processing systems*, vol. 30, 2017.

[2] Android, *Arquitectura de la plataforma*, 2022. dirección: https://developer.android.com/guide/platform?hl=es-419.

[3] Statcounter, *Mobile Operating System Market Share Worldwide*, 2022. dirección: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[4] Statcounter, *Cuota de mercado de los sistemas operativos en el mundo en el último año*, 2022. dirección: https://gs.statcounter.com/os-market-share#monthly-202109-202208-bar.

[5] Oracle, *The Java Virtual Machine Specification*, 2022. dirección: https://docs.oracle.com/javase/specs/jvms/se7/jvms7.pdf.

[6] R. Meier, *Professional Android 4 application development*. John Wiley & Sons, 2012.

[7] TIOBE, *Ranking de lenguajes de programacion usados en los ultimos 20 años*, 2022. dirección: https://www.tiobe.com/tiobe-index/.

[8] H. Verma, *JVM vs DVM*, 2019. dirección: https://towardsdatascience.com/jvm-vs-dvm-b257229d18a2.

[9] Android, *Configuración de ART*, 2022. dirección: https://source.android.com/devices/tech/dalvik/configure.html#how_art_works.

[10] Android, *Tiempo de ejecución de Android (ART) y Dalvik*, 2022. dirección: https://source.android.com/devices/tech/dalvik.

[11] Android, *Descripción general sobre los recursos de las apps*, 2022. dirección: https://developer.android.com/guide/topics/resources/providing-resources.

[12] E. P. Ratazzi, «Understanding and improving security of the Android operating system», Tesis doct., Syracuse University, 2016.

[13] K. Dunham, S. Hartman, M. Quintans, J. A. Morales y T. Strazzere, *Android malware and analysis*. CRC Press, 2014.

[14] A. Martín, R. Lara-Cabrera y D. Camacho, «Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset», *Information Fusion*, vol. 52, págs. 128-142, 2019.

[15] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh y L. Cavallaro, «The evolution of android malware and android analysis techniques», *ACM Computing Surveys (CSUR)*, vol. 49, nº 4, págs. 1-41, 2017.

[16] E. Dupuy, *jd-gui*, 2022. dirección: http://java-decompiler.github.io/.

[17] C. T. Ryszard Wiśniewski, *Apktool*, 2022. dirección: https://ibotpeaches.github.io/Apktool/.

[18] pxb1988, *dex2jar*, 2022. dirección: https://github.com/pxb1988/dex2jar.

[19] L. Benfield, *CFR - another java decompiler*, 2022. dirección: http://www.benf.org/other/cfr/.

[20] M. Strobel, *procyon*, 2022. dirección: https://github.com/mstrobel/procyon.

[21] erev0s, *lazyX*, 2022. dirección: https://github.com/erev0s/lazyX.

[22] skylot, *JADX*, 2022. dirección: https://github.com/skylot/jadx.

[23] J. Freke, *smali*, 2022. dirección: https://github.com/JesusFreke/smali.

[24] T. M. King, J. Arbon, D. Santiago, D. Adamo, W. Chin y R. Shanmugam, «AI for Testing Today and Tomorrow: Industry Perspectives», págs. 81-88, 2019. DOI: 10.1109/AITest.2019.000-3.

[25] P. R. Srivastava y T.-h. Kim, «Application of genetic algorithm in software testing», *International Journal of software Engineering and its Applications*, vol. 3, nº 4, págs. 87-96, 2009.

[26] D. Appelt, C. D. Nguyen, A. Panichella y L. C. Briand, «A Machine-Learning-Driven Evolutionary Approach for Testing Web Application Firewalls», *IEEE Transactions on Reliability*, vol. 67, nº 3, págs. 733-757, 2018. DOI: 10.1109/TR.2018.2805763.

[27] Y. Wang, W. Wang, S. Joty y S. C. Hoi, «Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation», *arXiv preprint arXiv:2109.00859*, 2021.

[28] M. Alkasassbeh y M. Almseidin, «Machine Learning Methods for Network Intrusion Detection», *CoRR*, vol. abs/1809.02610, 2018. dirección: http://arxiv.org/abs/1809.02610.

[29] D. Fraunholz, M. Zimmermann y H. D. Schotten, «An adaptive honeypot configuration, deployment and maintenance strategy», págs. 53-57, 2017. DOI: 10.23919/ICACT.2017.7890056.

[30] D. Jeong, «Artificial Intelligence Security Threat, Crime, and Forensics: Taxonomy and Open Issues», *IEEE Access*, vol. 8, págs. 184 560-184 574, 2020. DOI: 10.1109/ACCESS.2020.3029280.

[31] D. Jeong, «Artificial Intelligence Security Threat, Crime, and Forensics: Taxonomy and Open Issues», *IEEE Access*, vol. 8, págs. 184 560-184 574, 2020. DOI: 10.1109/ACCESS.2020.3029280.

[32] T. M. Mitchell, *Machine Learning*. New York, NY: McGraw-Hill, 1997.

[33] L. Deng, «The mnist database of handwritten digit images for machine learning research», *IEEE Signal Processing Magazine*, vol. 29, nº 6, págs. 141-142, 2012.

[34] M. Mohri, A. Rostamizadeh y A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012, ISBN: 026201825X.

[35] Google, *Different approaches to Machine Learning*, 2022. dirección: https://storage.googleapis.com/gweb-news-initiative-training.appspot.com/upload/Lesson3_1.pdf.

[36] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[37] F. Rosenblatt, «The perceptron: a probabilistic model for information storage and organization in the brain.», *Psychological review*, vol. 65, nº 6, pág. 386, 1958.

[38] A. Krogh, «What are artificial neural networks?», *Nature biotechnology*, vol. 26, nº 2, págs. 195-197, 2008.

[39] A. Cartas, *Esquema de un perceptrón con cinco entradas*, 2022. dirección: https://commons.wikimedia.org/w/index.php?curid=41534843.

[40] M. Hilbert y P. López, «The World’s Technological Capacity to Store, Communicate, and Compute Information», *Science*, vol. 332, nº 6025, págs. 60-65, 2011. DOI: 10.1126/science.1200970. eprint: https://www.science.org/doi/pdf/10.1126/science.1200970. dirección: https://www.science.org/doi/abs/10.1126/science.1200970.

[41] K. Weiss, T. M. Khoshgoftaar y D. Wang, «A survey of transfer learning», *Journal of Big data*, vol. 3, nº 1, págs. 1-40, 2016.

[42] A. Gammerman, V. Vovk y V. Vapnik, «Learning by transduction», *arXiv preprint arXiv:1301.7375*, 2013.

[43] A. Kazemnejad, *Transformer Architecture: The Positional Encoding*, 2019. dirección: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[44] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai y X. Huang, «Pre-trained models for natural language processing: A survey», *Science China Technological Sciences*, vol. 63, nº 10, págs. 1872-1897, 2020.

[45] T. Lin, Y. Wang, X. Liu y X. Qiu, «A survey of transformers», *AI Open*, vol. 3, págs. 111-132, 2022, ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2022.10.001. dirección: https://www.sciencedirect.com/science/article/pii/S2666651022000146.

[46] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «Bert: Pre-training of deep bidirectional transformers for language understanding», *arXiv preprint arXiv:1810.04805*, 2018.

[47] Y. Liu, M. Ott, N. Goyal et al., «Roberta: A robustly optimized bert pretraining approach», *arXiv preprint arXiv:1907.11692*, 2019.

[48] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever et al., «Improving language understanding by generative pre-training», 2018.

[49] M. Lewis, Y. Liu, N. Goyal et al., «Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension», *arXiv preprint arXiv:1910.13461*, 2019.

[50] C. Raffel, N. Shazeer, A. Roberts et al., «Exploring the limits of transfer learning with a unified text-to-text transformer.», *J. Mach. Learn. Res.*, vol. 21, nº 140, págs. 1-67, 2020.

[51] A. Ramesh, M. Pavlov, G. Goh et al., «Zero-Shot Text-to-Image Generation», 2021. DOI: 10.48550/ARXIV.2102.12092. dirección: https://arxiv.org/abs/2102.12092.

[52] A. Baevski, H. Zhou, A. Mohamed y M. Auli, «wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations», *CoRR*, vol. abs/2006.11477, 2020. arXiv: 2006.11477. dirección: https://arxiv.org/abs/2006.11477.

[53] OpenAI, *ChatGPT: Optimizing Language Models for Dialogue*, 2022. dirección: https://openai.com/blog/chatgpt/.

[54] C. Manning y H. Schutze, *Foundations of statistical natural language processing*. MIT press, 1999.

[55] Y. Kang, Z. Cai, C.-W. Tan, Q. Huang y H. Liu, «Natural language processing (NLP) in management research: A literature review», *Journal of Management Analytics*, vol. 7, nº 2, págs. 139-172, 2020.

[56] M. Allahyari, S. A. Pouriyeh, M. Assefi et al., «Text Summarization Techniques: A Brief Survey», *CoRR*, vol. abs/1707.02268, 2017. dirección: http://arxiv.org/abs/1707.02268.

[57] L. Abualigah, M. Q. Bashabsheh, H. Alabool y M. Shehab, «Text Summarization: A Brief Review», en *Recent Advances in NLP: The Case of Arabic Language*, M. Abd Elaziz, M. A. A. Al-qaness, A. A. Ewees y A. Dahou, eds. Cham: Springer International Publishing, 2020, págs. 1-15, ISBN: 978-3-030-34614-0. DOI: 10.1007/978-3-030-34614-0_1. dirección: https://doi.org/10.1007/978-3-030-34614-0_1.

[58] N. Chirkova y S. Troshin, «Empirical study of transformers for source code», en *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, págs. 703-715.

[59] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang y X. Liu, «A novel neural source code representation based on abstract syntax tree», en *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, págs. 783-794.

[60] W. Wang, G. Li, S. Shen, X. Xia y Z. Jin, «Modular tree network for source code representation learning», *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, nº 4, págs. 1-23, 2020.

[61] M. Chen, J. Tworek, H. Jun et al., «Evaluating large language models trained on code», *arXiv preprint arXiv:2107.03374*, 2021.

[62] F. F. Xu, U. Alon, G. Neubig y V. J. Hellendoorn, «A systematic evaluation of large language models of code», en *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 2022, págs. 1-10.

[63] A. Kanade, P. Maniatis, G. Balakrishnan y K. Shi, «Pre-trained contextual embedding of source code», 2019.

[64] Z. Feng, D. Guo, D. Tang et al., «Codebert: A pre-trained model for programming and natural languages», *arXiv preprint arXiv:2002.08155*, 2020.

[65] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis y M. Brockschmidt, «Codesearchnet challenge: Evaluating the state of semantic code search», *arXiv preprint arXiv:1909.09436*, 2019.

[66] D. Guo, S. Ren, S. Lu et al., «Graphcodebert: Pre-training code representations with data flow», *arXiv preprint arXiv:2009.08366*, 2020.

[67] D. Zügner, T. Kirschstein, M. Catasta, J. Leskovec y S. Günnemann, «Language-agnostic representation learning of source code from structure and context», *arXiv preprint arXiv:2103.11318*, 2021.

[68] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov y Q. V. Le, «Xlnet: Generalized autoregressive pretraining for language understanding», *Advances in neural information processing systems*, vol. 32, 2019.

[69] M. Allamanis, H. Peng y C. Sutton, «A convolutional attention network for extreme summarization of source code», en *International conference on machine learning*, PMLR, 2016, págs. 2091-2100.

[70] U. Alon, S. Brody, O. Levy y E. Yahav, «code2seq: Generating sequences from structured representations of code», *arXiv preprint arXiv:1808.01400*, 2018.

[71] W. U. Ahmad, S. Chakraborty, B. Ray y K.-W. Chang, «Unified pre-training for program understanding and generation», *arXiv preprint arXiv:2103.06333*, 2021.

[72] M.-A. Lachaux, B. Roziere, L. Chanussot y G. Lample, «Unsupervised translation of programming languages», *arXiv preprint arXiv:2006.03511*, 2020.

[73] L. Phan, H. Tran, D. Le et al., «Cotext: Multi-task learning with code-text transformer», *arXiv preprint arXiv:2105.08645*, 2021.

[74] A. H. Lashkari, A. F. A. Kadir, L. Taheri y A. A. Ghorbani, «Toward developing a systematic approach to generate benchmark android malware datasets and classification», en *2018 International Carnahan Conference on Security Technology (ICCST)*, IEEE, 2018, págs. 1-7.

[75] Google, *Google Java Style Guide*, 2022. dirección: https://google.github.io/styleguide/javaguide.html#s4.4-column-limit.

[76] Salesforce, *Github: Salesforce CodeT5*, 2022. dirección: https://github.com/salesforce/CodeT5.

[77] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1ª ed. USA: Prentice Hall PTR, 2008, ISBN: 0132350882.

[78] M. Guo, J. Ainslie, D. Uthus et al., «Longt5: Efficient text-to-text transformer for long sequences», *arXiv preprint arXiv:2112.07916*, 2021.

[79] facultyai, *Dash Bootstrap Components*, 2022. dirección: https://github.com/facultyai/dash-bootstrap-components.

Acronyms

AOT Ahead-of-time. 11

API Application Programming Interface. 6

APK Android Application Package. 10

ART Android Run Time. 6, 10, 11

AST Abstract Syntax Tree. 35, 37, 38

ASTNN AST-based Neural Network. 35

C4 Colossal Clean Crawled Corpus. 33

CNN Red neuronal convolucional. 23

DL Deep Learning. 19, 20

DVM Dalvik virtual machine. 10, 11

FNN Feedforward neural network. 25

HAL Hardware Application Layer. 6

IA Artificial intelligence. 17, 18

IDE Integrated Development Environment. 10

JVM Java virtual machine. 8–10

ML Machine Learning. 18, 20, 40

MLM Masked Language Modeling. 39

MRR Mean Reciprocal Rank. iii, 39

NLG Natural Language Generation. 30–32, 39, 43

NLP Natural Language Processing. 30–34, 36, 45, 51

NLU Natural Language Understanding. 30–32, 39, 43

PTM Pre-trained Language Models. 30

RNN Red neuronal recurrente. 23, 25, 27, 35

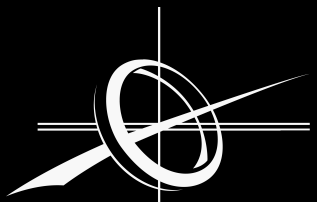
RRNN Redes de neuronas. 20

T5 Text-to-Text Transfer Transformer. ii, 33, 34

TNN Abstract syntax tree. 35

UPM Universidad Politécnica de Madrid. 31

WORA Write once, run anywhere. 9


Universidad
Politécnica
de Madrid
ETSI SISTEMAS
INFORMÁTICOS