UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

LARGE LANGUAGE MODEL INFERENCE WITH LIMITED MEMORY

EDUARDO CID PÉREZ

Director/a
RUBÉN TOUS LIESA (Departamento de Arquitectura de Computadores)

Titulación
Grado en Ingeniería Informática (Computación)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Índice

Índice de figuras

Índice de cuadros

1. Contexto

1.1. Introducción y contextualización

Los Modelos de Lenguaje de Gran Escala (LLMs) han demostrado ser herramientas fundamentales en el ámbito de la inteligencia artificial y el procesamiento del lenguaje natural. Sin embargo, su despliegue en dispositivos con memoria limitada sigue siendo un desafío significativo. Este proyecto se enmarca dentro de la investigación en optimización de inferencia de LLMs, con el objetivo de reducir el consumo de memoria sin comprometer la latencia y la precisión.

En particular, se busca explorar soluciones para permitir que modelos avanzados puedan ejecutarse en hardware con capacidad reducida, como dispositivos IoT, teléfonos móviles o servidores de bajo consumo. Esto permitirá una mayor accesibilidad a la inteligencia artificial avanzada sin depender exclusivamente de grandes infraestructuras en la nube.

Este trabajo de fin de grado será llevado a cabo en el marco de la Facultad de Informática de Barcelona (FIB), en la Universidad Politécnica de Cataluña (UPC). En este proyecto se abordarán varias áreas de las especialidades que la universidad ofrece para este grado, destacando especialmente la de computación. Donde nos centraremos en particular en ciertos temas de asignaturas como Inteligencia Artificial (IA), Sistemas Inteligentes Distribuidos (SID) y Arquitectura de Computadores (AC).

1.2. Conceptos Básicos

1.2.1. LLM

Un **Modelo de Lenguaje de Gran Escala** (LLM, por sus siglas en inglés) es un tipo de inteligencia artificial entrenada para procesar y generar texto de manera similar a cómo lo haría un humano. Funciona analizando grandes cantidades de datos y aprendiendo patrones en el lenguaje.

Una analogía que ayuda a entender el funcionamiento de un LLM es la de un chef que ha leído miles de recetas de cocina y ha practicado con muchas de ellas. Aunque no ha probado cada plato personalmente, puede generar una nueva receta basándose en lo que ha aprendido. De manera similar, un LLM no entiende el lenguaje como un humano, pero puede predecir y generar frases basadas en su entrenamiento.

1.2.2. Edge Computing

El **Edge Computing** es una forma de procesar datos más cerca del lugar donde se generan, en lugar de enviarlos a un servidor central o a la nube. Esto reduce el tiempo de respuesta y el uso de ancho de banda. También ofrece una mejor eficiencia al tener los dispositivos edge un diseño que prioriza el bajo consumo. Adicionalmente, el procesado local de los datos evita que estos viajen por la red, en consecuencia podemos disponer de mayor privacidad.

1.2.3. Inferencia en LLM

La **inferencia en LLM** es el proceso de utilizar un modelo de lenguaje ya entrenado para generar respuestas o completar tareas, como responder preguntas o traducir texto. Es la fase en la que el modelo aplica lo que ha aprendido en su entrenamiento para producir resultados en tiempo real.

La inferencia tiene un coste computacional muy pequeño comparado con la fase de entrenamiento, que suele implicar el uso de grandes volúmenes de datos y clústers de computadores de alto rendimiento ejecutándose durante días o incluso semanas. No obstante, la inferencia debe operar dentro de estrictos límites de latencia para ofrecer respuestas rápidas y necesita que el modelo esté cargado en memoria, lo que puede ser un desafío en dispositivos con recursos limitados.

1.3. Identificación o Formulación del Problema

El problema central que aborda este proyecto es la dificultad de ejecutar LLMs en edge devices como hardware con memoria restringida, como dispositivos IoT, servidores con recursos limitados, computadoras personales, dispositivos móviles, vehículos autónomos o dispositivos médicos portátiles. Las soluciones existentes requieren grandes cantidades de memoria, lo que limita su aplicabilidad en entornos fuera de la nube. La investigación se centrará en analizar y evaluar estrategias para reducir el uso de memoria durante la inferencia de estos modelos.

Se explorarán métodos como la cuantización, knowledge distillation , pruning, knowledge caching y técnicas de inferencia distribuida para encontrar el equilibrio óptimo entre memoria y rendimiento.

1.4. Stakeholders o actores implicados

- **Investigadores en IA**: Interesados en nuevas formas de optimización de LLMs.
- **Empresas tecnológicas**: Buscan reducir costes computacionales y hacer viable la inferencia en hardware limitado.
- **Desarrolladores de software**: Necesitan modelos más eficientes para integrar en sus aplicaciones.
- **Usuarios finales**: Beneficiarios de aplicaciones que requieren procesamiento de lenguaje en dispositivos sin acceso a la nube.
- **Fabricantes de hardware**: Interesados en desarrollar dispositivos optimizados para ejecutar modelos de IA de manera eficiente.
- **Yo**: Persona encargada de llevar a cabo el trabajo y hacer las investigaciones pertinentes para poder realizarlo.
- **Tutor del TFG**: Persona encargada de supervisar el TFG durante su realización y proporcionar el apoyo necesario.

2. Justificación

Con el paso de los años podemos ver que, aunque los LLMs tienen capacidades increíbles, su alto número de parámetros y el enorme coste computacional que tienen produce que su despliegue sea una de las principales fuentes de emisión de carbono en aplicaciones de IA.

Las GPUs modernas como la H100 generan una huella de carbono mucho mayor en comparación con modelos más antiguos. Por ejemplo, la M40 solo emite un tercio del carbono que genera la H100, como se muestra en la Figura 1 [1].

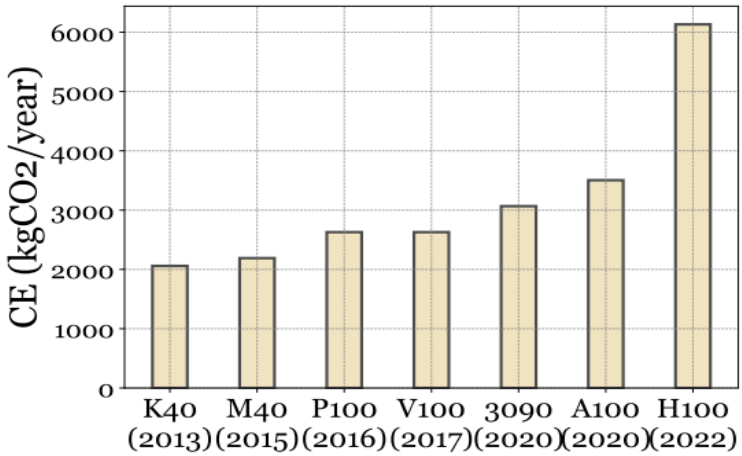

Figura 1: Gráfica que muestra la evolución de emisión de carbono de las distintas GPUs que han ido saliendo con los años[1]

La reducción del uso de memoria en la inferencia de LLMs no solo mejora la eficiencia computacional, sino que también tiene un impacto directo en varios Objetivos de Desarrollo Sostenible (ODS) [2] de la ONU:

- **ODS 9 (Industria, Innovación e Infraestructura)** [3]: La optimización de modelos permite el desarrollo de infraestructuras tecnológicas más eficientes y accesibles, facilitando la adopción de IA en diversos sectores sin la necesidad de hardware costoso.
- **ODS 12 (Producción y Consumo Responsables)** [4]:Al disminuir la demanda de memoria y energía, se reducen los costes ambientales de los centros de datos y el hardware de inferencia, promoviendo el uso sostenible de los recursos tecnológicos.
- **ODS 13 (Acción por el Clima)** [5]: La eficiencia energética en la inferencia de IA contribuye a la reducción de la huella de carbono asociada al entrenamiento y ejecución de modelos de lenguaje, alineándose con la meta de reducir el impacto climático de las tecnologías digitales.
- **ODS 10 (Reducción de las desigualdades)** [6]: Permitir la ejecución de modelos en dispositivos más accesibles democratiza el uso de la IA, beneficiando comunidades con recursos limitados y fomentando la equidad en el acceso a la tecnología.

La investigación propuesta impacta directamente en estos ODS al buscar métodos que hagan posible la ejecución de modelos avanzados en entornos

con limitaciones de memoria y computación, reduciendo la dependencia de grandes infraestructuras y promoviendo una IA más accesible y sostenible.

Esto también nos acerca al concepto de **Edge Computing**, ya que permite procesar la inferencia de los modelos directamente en nuestros dispositivos. Una ventaja clave de este enfoque es el aumento en la privacidad, dado que los datos introducidos en el LLM no necesitan enviarse a la nube, sino que permanecen en el dispositivo del usuario.

En la actualidad, se han realizado numerosas investigaciones que abordan diversas técnicas para optimizar la memoria durante la inferencia en modelos de lenguaje de gran tamaño (LLMs). Sin embargo, comparar estas técnicas entre sí resulta complejo debido a que cada experimento se ha llevado a cabo bajo condiciones distintas. Por lo tanto, el objetivo principal de este trabajo es realizar una comparación exhaustiva de las diferentes soluciones disponibles, con el fin de extraer conclusiones fundamentadas sobre su eficacia y aplicabilidad en entornos con restricciones de memoria.

Podemos encontrar diversos trabajos que combinan técnicas de optimización para modelos de lenguaje a gran escala (**LLMs**). Por ejemplo, QLoRA [7] propone aplicar cuantización de baja precisión (4 bits), seguida de un *fine-tuning* eficiente mediante LoRA, lo que permite adaptar modelos grandes con una huella de memoria muy reducida. Por otro lado, LoRA-Prune [8] investiga una estrategia que combina *pruning* progresivo junto con *fine-tuning* basado en LoRA, aplicados de forma iterativa durante el proceso de entrenamiento. Asimismo, existen repositorios como Awesome-Pruning [9] que recopilan implementaciones y comparativas de distintas técnicas de *pruning* aplicadas a modelos como `BERT`, `GPT-2` o `LLaMA`.

Además, cabe destacar SparseGPT [10], una técnica de *one-shot structured pruning* para LLMs que logra eliminar hasta el 60 % de los parámetros sin necesidad de *fine-tuning*, manteniendo una precisión razonablemente alta. SparseGPT demuestra ser especialmente útil en entornos con recursos limitados, al reducir drásticamente el consumo de memoria y computación.

3. Alcance y Obstáculos

3.1. Alcance: Objetivos y Subobjetivos

- **Objetivo principal**: Analizar y evaluar estrategias de optimización para la inferencia de LLMs en entornos con memoria limitada.
- **Subobjetivos**:
  - Investigar técnicas de optimización existentes y sus impactos en memoria y latencia.
  - Implementar y probar algunas de estas técnicas en modelos de lenguaje preentrenados.
  - Evaluar el balance entre memoria y latencia en función del tipo de optimización utilizada.

3.2. Obstáculos y Riesgos

- **Obstáculos:**
  - Acceso a hardware adecuado para pruebas y benchmarking.
  - Complejidad en la implementación de algunas optimizaciones avanzadas.
  - Equilibrio entre eficiencia, latencia y precisión del modelo.
  - Falta de documentación clara en algunas técnicas emergentes de optimización.
- **Riesgos:**
  - Pérdida de precisión en el modelo debido a la reducción de tamaño.
  - Dependencia de bibliotecas de terceros y software ajeno.
  - Posible sobrecarga computacional en ciertas configuraciones.
  - Pérdida de datos por daños en el equipo.

4. Metodología de trabajo y herramientas de seguimiento

4.1. Metodología

Dado que el desarrollo de este proyecto se basa en la investigación y análisis de distintas técnicas de optimización para la inferencia de Modelos de Lenguaje de Gran Escala (LLMs), las tareas a realizar son mayormente independientes entre sí. Cada técnica debe ser estudiada, implementada y evaluada de manera individual, y la única dependencia existente es el análisis inicial de las diferentes estrategias antes de realizar comparaciones entre ellas.

Debido a esta naturaleza del trabajo, la metodología **Kanban** es la más adecuada. Kanban es un enfoque ágil basado en la gestión visual de tareas mediante un tablero donde se organizan en diferentes estados (por ejemplo, Por hacer, En progreso, Finalizado). Esto permite un seguimiento claro del progreso de cada técnica sin imponer una estructura rígida de planificación.

Esto proporciona las siguientes ventajas:

- Flexibilidad
- Transparencia y seguimiento
- Permite modificaciones

4.2. Herramientas para el desarrollo

Para la gestión del flujo de trabajo, se utilizará la herramienta **Trello**, donde cada tarea tendrá su propia tarjeta con detalles, y nos permitirá un fácil seguimiento del estado de cada de estas.

Además para poder ir guardando de forma segura los datos y así evitar problemas de pérdida de estos se usará la herramienta **GitHub**. Esta nos permitirá ir subiendo y actualizando las nuevas partes del proyecto.

4.3. Seguimiento

Para el seguimiento del proyecto se irán organizando reuniones presenciales cada dos semanas con el tutor del TFG, donde se discutirá el estado del proyecto y se abordará dudas, ideas o problemas que vayan surgiendo durante el desarrollo del mismo. Además de estas también se mantendrá contacto vía email para dudas que requieran una respuesta más rápidas y/o urgentes.

5. Planificación Temporal

En esta sección se detalla la planificación temporal del proyecto, estableciendo las tareas necesarias para su ejecución. Además, se realizará una estimación de la duración del trabajo, con el objetivo de garantizar un desarrollo estructurado y realista del proyecto. La fecha de inicio del trabajo es el 19 de febrero de 2025, y la fecha de finalización es el 25 de junio de 2025.

Inicialmente, se estima que la duración total del trabajo sea alrededor de las 540, ya que tiene un valor de 18 créditos ECTS (15 la parte del proyecto y 3 la parte de GEP), y tal y como se indica en la normativa de TFG de la FIB, un crédito ECTS de TFG equivale a unas 30 horas [11].

5.1. Descripción de las tareas

5.1.1. Gestión del proyecto

GP1 - Alcance y Contextualización (30 horas)

- **Descripción**: Definición del contexto del proyecto, incluyendo los objetivos, alcance y requisitos. Esta tarea implica la investigación inicial para entender el problema y establecer las bases del proyecto.
- **Dependencias**: Ninguna.

GP2 - Planificación Temporal (25 horas)

- **Descripción**: Creación de un diagrama de Gantt y planificación detallada de las tareas, incluyendo la asignación de tiempos y recursos.
- **Dependencias**: GP1 (Alcance y Contextualización).

GP3 - Presupuesto y Sostenibilidad (15 horas)

- **Descripción**: Estimación de los costes asociados al proyecto y análisis del impacto ambiental, social y económico.
- **Dependencias**: GP2 (Planificación Temporal).

GP4 - Corrección y preparación del documento final (5 horas)

- **Descripción**: Revisión y corrección del documento final del proyecto teniendo en cuenta el feedback de las entregas anteriores.
- **Dependencias**: GP3 (Presupuesto y Sostenibilidad).

GP5 - Reuniones de seguimiento (20 horas)

- **Descripción**: Reuniones periódicas con el tutor para revisar el progreso del proyecto y realizar ajustes en la planificación si es necesario.
- **Dependencias**: Ninguna.

5.1.2. Desarrollo de las Técnicas

Para el desarrollo de las técnicas que usaremos, todas las tareas se dividirán en tres fases: la primera consistirá en la investigación de la técnica, la segunda en su implementación y en la tercera obtendremos los resultados de la aplicación de la técnica para más tarde analizarlos. En cada caso, cada subtarea dependerá de la subtarea anterior, ya que, como es lógico, antes de implementarla será necesario hacer un estudio de la técnica y entenderla, y para obtener los resultados necesitaremos haber aplicado la técnica anteriormente.

DT1 - Desarrollo de la Cuantización

- **DT1.1 - Investigación de la técnica (30 horas)**: Estudio de las técnicas de cuantización para reducir el tamaño de los modelos.
- **DT1.2 - Desarrollo de la técnica (55 horas)**: Implementación de la técnica de cuantización en los modelos seleccionados.
- **DT1.3 - Obtención de Resultados (15 horas)**: Pruebas y evaluación de los resultados obtenidos con la técnica de cuantización.

DT2 - Desarrollo de Static Sparsity

- **DT2.1 - Investigación de la técnica (30 horas)**: Estudio de las técnicas de "static sparsity"para optimizar los modelos.
- **DT2.2 - Desarrollo de la técnica (55 horas)**: Implementación de la técnica de "static sparsity".
- **DT2.3 - Obtención de Resultados (15 horas)**: Pruebas y evaluación de los resultados obtenidos con la técnica de "static sparsity".

DT3 - Desarrollo de Dynamic Sparsity

- **DT3.1 - Investigación de la técnica (30 horas)**: Estudio de las técnicas de "dynamic sparsity"para optimizar los modelos.

- **DT3.2 - Desarrollo de la técnica (55 horas)**: Implementación de la técnica de "dynamic sparsity".
- **DT3.3 - Obtención de Resultados (15 horas)**: Pruebas y evaluación de los resultados obtenidos con la técnica de "dynamic sparsity".

5.1.3. Evaluación de Resultados

ER1 - Análisis de resultados (25 horas)

- **Descripción**: Análisis detallado de los resultados obtenidos con las técnicas implementadas.
- **Dependencias**: DT (Desarrollo de las Técnicas), ya que para poder hacer el análisis de los resultados deberemos haberlos obtenido.

ER2 - Comparación con LLM sin optimizar (15 horas)

- **Descripción**: Comparación de los resultados obtenidos con los modelos optimizados frente a los modelos sin optimizar.
- **Dependencias**: ER1 (Análisis de resultados), necesitaremos haber analizado los resultados para poder compararlos con los datos sin la aplicación de las optimizaciones.

ER3 - Conclusiones (15 horas)

- **Descripción**: Redacción de las conclusiones del proyecto, incluyendo los hallazgos más importantes y las lecciones aprendidas.
- **Dependencias**: ER2 (Comparación con LLM sin optimizar), una vez tengamos la comparación podremos sacar conclusiones.

5.1.4. Documentación

DC1 - Documentación (70 horas)

- **Descripción**: Redacción de la memoria del proyecto, incluyendo la justificación, metodología, resultados y conclusiones.
- **Dependencias**: Durante DT y ER.

DC2 - Defensa TFG (20 horas)

- **Descripción**: Preparación y presentación de la defensa del Trabajo de Fin de Grado (TFG).
- **Dependencias**: DC1 (Documentación), para poder preparar la defensa será necesario haber acabado el tarbajo.

5.2. Tabla de Tareas

Código	Tarea	Tiempo	Dependencias
GP	**Gestión del proyecto**	**Total: 95h**	-
GP1	Alcance y Contextualización	30h	-
GP2	Planificación Temporal	25h	GP1
GP3	Presupuesto y Sostenibilidad	15h	GP2
GP4	Corrección y preparación del documento final	5h	GP3
GP5	Reuniones de seguimiento	20h	-
DT	**Desarrollo de las Técnicas**	**Total: 300h**	-
DT1	Desarrollo de la Cuantización	-	-
DT1.1	Investigación de la técnica	30h	-
DT1.2	Desarrollo de la técnica	55h	DT1.1
DT1.3	Obtención de Resultados	15h	DT1.2
DT2	Desarrollo de Static sparsity	-	-
DT2.1	Investigación de la técnica	30h	-
DT2.2	Desarrollo de la técnica	55h	DT2.1
DT2.3	Obtención de Resultados	15h	DT2.2
DT3	Desarrollo de Dynamic Sparsity	-	-
DT3.1	Investigación de la técnica	30h	-
DT3.2	Desarrollo de la técnica	55h	DT3.1
DT3.3	Obtención de Resultados	15h	DT3.2
ER	**Evaluación de Resultados**	**Total: 55h**	-
ER1	Análisis de resultados	25h	DT
ER2	Comparación con LLM sin optimizar	15h	ER1
ER3	Conclusiones	15h	ER2
DC	**Documentación**	**Total: 90h**	-
DC1	Documentación	70h	Durante DT y ER
DC2	Defensa TFG	20h	DC1

Cuadro 1: Tabla de planificación de tareas

5.3. Diagrama de Gantt

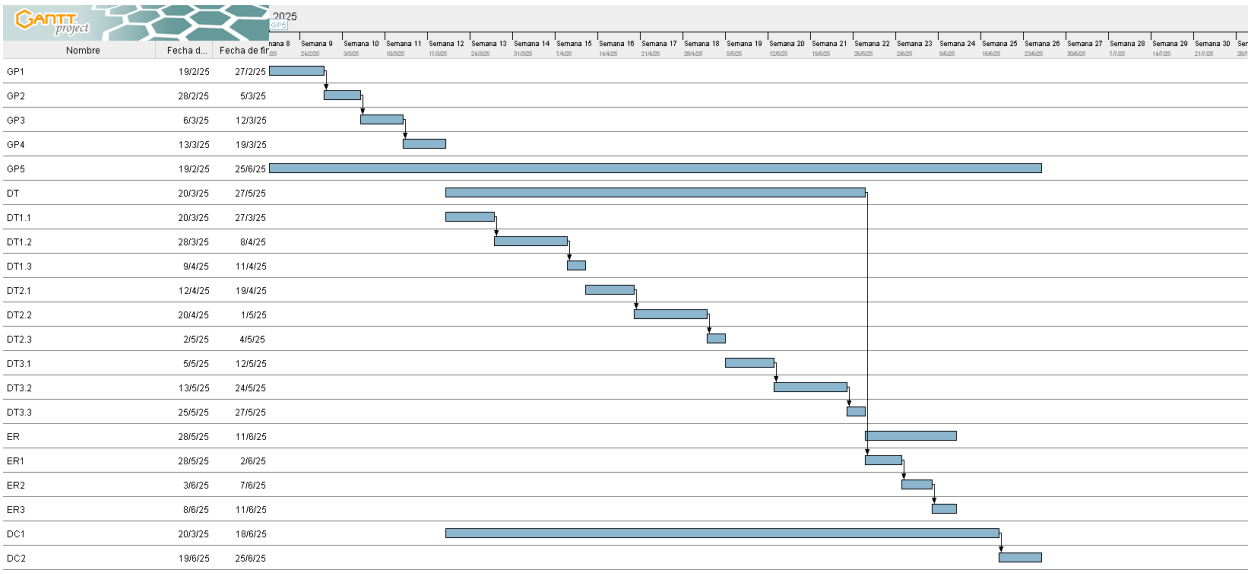

Figura 2: Diagrama de Gantt del proyecto (Elaboración propia).

6. Recursos

En esta sección se detallan los recursos humanos, materiales y herramientas que se utilizarán para el desarrollo del proyecto.

6.1. Recursos Humanos

- **RH1 - Estudiante**: Encargado de la investigación, implementación, pruebas y documentación del proyecto.
- **RH2 - Tutor**: Supervisión del proyecto, revisión del progreso y orientación en las decisiones técnicas.

6.2. Recursos Materiales

- **RM1 - Portátil Acer**:
  - **Especificaciones**: AMD Ryzen 7 7735HS, 16 GB de RAM, 1 TB SSD, NVIDIA RTX 4050.

- **Uso**: Desarrollo local del proyecto, ejecución de pruebas y experimentos de menor escala.

- **RM2 - Cluster de GPUs**:
  - **Especificaciones**: Varias NVIDIA RTX 4090.
  - **Uso**: Ejecución de experimentos en caso de necesidad de más potencia computacional.

- **RM3 - Oficina de Coworking**:
  - **Ubicación**: Espacio de trabajo compartido con acceso a internet de alta velocidad.
  - **Uso**: Redacción de documentación, espacio para trabajar en un entorno adecuado.

6.3. Herramientas de Software

- **SW1 - GanttProject**:
  - **Uso**: Creación y gestión del diagrama de Gantt para la planificación temporal del proyecto.

- **SW2 - Visual Studio Code (VSCode)**:
  - **Uso**: Entorno de desarrollo integrado (IDE) para la programación y edición de código.

- **SW3 - GitHub**:
  - **Uso**: Control de versiones y almacenamiento del código fuente del proyecto.

- **SW4 - Python**:
  - **Uso**: Lenguaje de programación principal para la implementación de las técnicas de optimización.

- **SW5 - Overleaf**:
  - **Uso**: Plataforma para la redacción de la memoria del proyecto en LaTeX.

7. Gestión de Riesgo

En esta sección se identifican los principales riesgos asociados al proyecto, así como las soluciones o medidas ya implementadas para mitigarlos.

7.1. Riesgos Identificados y Soluciones

- **R1 - Coste computacional alto** (Riesgo Bajo):
  - **Descripción**: El uso intensivo de recursos computacionales, especialmente en experimentos a gran escala, podría generar un coste elevado.
  - **Solución**: Para mitigar este riesgo, se dispone de un clúster con varias GPUs NVIDIA RTX 4090, que permite ejecutar los experimentos de manera eficiente sin incurrir en costes adicionales.
  - **Posibilidad**: Media/Alta.
- **R2 - Complicaciones en la implementación de técnicas** (Riesgo Medio):
  - **Descripción**: Algunas técnicas de optimización (como cuantización, "sparsity.º PagedAttention) pueden ser complejas de implementar, lo que podría retrasar el proyecto.
  - **Solución**: Se ha asignado un margen de error en los tiempos de desarrollo para cada técnica, lo que permite abordar posibles complicaciones sin afectar el cronograma general del proyecto.
  - **Posibilidad**: Media.
- **R3 - Errores en el código o en los experimentos** (Riesgo Bajo):
  - **Descripción**: Errores en el código o en la configuración de los experimentos podrían generar resultados incorrectos o retrasos.
  - **Solución**: Se utilizará GitHub para el control de versiones, lo que permitirá revertir cambios problemáticos y mantener un historial de todas las modificaciones.
  - **Posibilidad**: Media.
- **R4 - Pérdida de datos** (Riesgo Alto):
  - **Descripción**: Podría ocurrir la pérdida de datos debido a fallos en el hardware o errores humanos.

- **Solución**: Se realizarán copias de seguridad periódicas (diarias) del código y los datos en GitHub y en discos en la nube. Además, se documentarán los pasos clave para poder reproducir los experimentos en caso de pérdida de datos.
- **Posibilidad**: Muy baja.

- **R5 - Falta de disponibilidad del clúster** (Riesgo Bajo/Medio):
  - **Descripción**: El clúster podría no estar disponible en ciertos momentos debido a mantenimiento o uso por otros proyectos.
  - **Solución**: Se planificarán los experimentos con anticipación y se reservará tiempo en el clúster. Además, se ejecutarán pruebas preliminares en el portátil local para reducir la dependencia del clúster.
  - **Posibilidad**: Media.

7.2. Plan de Contingencia

En caso de que alguno de los riesgos anteriores se materialice y las soluciones implementadas no sean suficientes, se seguirán las siguientes acciones:

- **Revisión de la planificación**: Si se detecta un retraso significativo, se revisará la planificación para redistribuir las tareas y priorizar las más críticas.
- **Comunicación con el tutor**: En caso de problemas graves, se informará al tutor para buscar soluciones conjuntas.

8. Gestión Económica

Como en todos los proyectos, la parte de estimación de los costes es una parte fundamental. Para ello vamos a realizar un estudio teniendo en cuenta los distintos factores que afectan al presupuesto del proyecto. Para llevar a cabo esto estudiaremos los costes de personal, los generales, los de contingencia y los imprevistos.

8.1. Presupuesto

8.1.1. Coste de personal

Primero de todo estimaremos los distintos costes de personal. Para ello hemos definido tres tipos de roles dependiendo de la tarea a realizar; para estimar el sueldo hemos seleccionado la media del rango de sueldos que indica la página web donde lo hemos consultado [12] [13] [14]. Luego hemos tenido en cuenta que el coste de la seguridad social(SS) es sobre un 30 por ciento.

Rol	Coste por hora	Seguridad Social	Coste Total/h
Jefe de proyecto	24.04€/h	7.32€/h	31.26€/h
Desarrollador	15,87€/h	4,73€/h	20.60€/h
Analista	16,83€/h	5.05€/h	21.88€/h

Cuadro 2: Tabla de Roles y Salarios

8.1.2. Costes por actividad

Una vez tenemos definidos los costes del personal para cada rol del proyecto podemos calcular el coste de las actividades que tendremos que realizar en este. Para ellos usaremos esta tabla donde asignaremos cada actividad a un rol para con ello poder estimar su coste:

Código	Tarea	Tiempo	Rol
GP1	Alcance y Contextualización	30h	JP
GP2	Planificación Temporal	25h	JP
GP3	Presupuesto y Sostenibilidad	15h	JP
GP4	Corrección y preparación del documento final	5h	JP
GP5	Reuniones de seguimiento	20h	JP
DT1.1	Investigación de la técnica	30h	D
DT1.2	Desarrollo de la técnica	55h	D
DT1.3	Obtención de Resultados	15h	D
DT2.1	Investigación de la técnica	30h	D
DT2.2	Desarrollo de la técnica	55h	D
DT2.3	Obtención de Resultados	15h	D
DT3.1	Investigación de la técnica	30h	D
DT3.2	Desarrollo de la técnica	55h	D
DT3.3	Obtención de Resultados	15h	D
ER1	Análisis de resultados	25h	A
ER2	Comparación con LLM sin optimizar	15h	A
ER3	Conclusiones	15h	A
DC1	Documentación	70h	JP
DC2	Defensa TFG	20h	JP

Cuadro 3: Tabla de tareas asignada a cada rol

De lo anterior resulta que el desarrollador (D) trabajará un total de 300 horas, el analista (A) trabajará 55 horas, y el jefe de proyecto (JP) lo hará 185 horas.

Si nos ajustamos a las estimaciones de los salarios que hemos visto en la tabla 1, el coste del desarrollador será de 300*20.60=6180€, el coste del analista será de 55*21.88=1203.4€ y el del jefe de proyecto será de 31.26*185=5783.1€, es decir, un total de costes por actividad todo nos da un total de **13166.5€**.

8.1.3. Costes genéricos

Los costes genéricos de un proyecto son aquellas categorías de gasto que se aplican de manera general a cualquier tipo de proyecto, independientemente de su naturaleza específica.

Nombre	Coste
Portátil (amortizado)	27.5€
Oficina Coworking (BCN)	520€
Total	547.5€

Cuadro 4: Tabla de costes genéricos

En este caso no tenemos ningún coste de software dado que todos los que usaremos son gratuitos.

8.1.4. Contingencia

La contingencia en un presupuesto de proyecto es una reserva financiera destinada a cubrir imprevistos que puedan surgir durante su ejecución. Esta provisión ofrece margen de maniobra y estabilidad frente a posibles incrementos en los costos o retrasos, evitando la necesidad de modificar el presupuesto global. Para este proyecto, se asignará un 7 % del costo total como contingencia, garantizando la cobertura de cualquier eventualidad no planificada.

Nombre	Coste	Contingencia
Costes por actividad	13166.5€	921.66€
Costes genéricos	547.5€	38.33€
Total coste + contigencia	-	**14673.99€**

Cuadro 5: Tabla de contingencia

8.1.5. Imprevistos

Finalmente, calcularemos el coste de los imprevistos que pueden ir surgiendo durante el desarrollo del proyecto; para ello, tendremos en cuenta la posibilidad de que estos sucedan y el coste que tendrán.

- **Fallo dispositivo**: En caso de que el portátil falle y haya que comprar uno nuevo el coste de este sería sobre los 1000€, pero dado que es bastante nuevo la posiblidad de que ocurra estaria entre el 5 y 10 %, por lo que lo dejaremos en un 7.5 %.
- **Necesidad de horas adicionales**: En caso que por problemas de planificación o de que una tarea resulte ser más complicada de lo que se prevé se podrían necesitar más horas. De todos modos esto es poco

probable ya que las horas han sido asignadas con un margen de error para evitar esto. De todos modos podríamos suponer que para alguna tarea se puedan necesitar 20h más, probablemente del desarrollador, que es donde podemos encontrar los errores, por lo que el coste sería de 412€, pero con una posiblidad del 5 %.

Nombre	Coste
Fallo dispositivo	75€
Necesidad de horas adicionales	20.6€
Total	**95.6€**

Cuadro 6: Tabla de costes imprevistos

8.1.6. Coste final estimado

Una vez hemos visto los distintos costes que hay en el proyecto, podemos hacer la suma de todos para poder establecer el coste final de este:

Nombre	Coste
Costes + Contingencias	14573.99€
Imprevistos	95.6€
Total	**14669.59€**

Cuadro 7: Tabla de costes finales

8.2. Control de gestión

Tras establecer el presupuesto inicial, se implementan medidas de control para prevenir desviaciones y se definen indicadores cuantitativos que faciliten su seguimiento.Al completar una tarea, se actualizará el presupuesto con las horas efectivamente empleadas y se comparará con las estimaciones previas.

Para gestionar posibles imprevistos, también se registrarán los gastos adicionales generados en cada tarea y se contrastarán con la reserva destinada a imprevistos y contingencias. Esto permitirá identificar desviaciones de manera rápida y evaluar si es necesario ajustar el alcance del proyecto o incrementar el presupuesto.

A continuación, se presentan los descriptores numéricos para el control:

1. **Desviación coste personal por tarea:**

$$(coste\_estimado - coste\_real) * horas\_reales \tag{1}$$

Esta métrica evalúa la diferencia entre el coste estimado y el coste real de una tarea, ponderada por el número de horas reales trabajadas.

2. **Desviación realización tareas:**

$$(horas\_estimadas - horas\_reales) * coste\_real \tag{2}$$

Representa la diferencia entre las horas estimadas y las reales, multiplicadas por el coste real.

3. **Desviación total en la realización de tareas:**

$$coste\_estimado\_total - coste\_real\_total \tag{3}$$

Mide la diferencia entre el coste total estimado y el coste total real de todas las tareas.

4. **Desviación total de recursos:**

$$coste\_estimado\_total - coste\_real\_total \tag{4}$$

Evalúa si los recursos empleados fueron mayores o menores que los previstos.

5. **Desviación total coste de imprevistos:**

$$coste\_estimado\_imprevistos - coste\_real\_imprevistos \tag{5}$$

Compara el presupuesto asignado a imprevistos con el gasto real en imprevistos.

6. **Desviación total de horas:**

$$horas\_estimadas - horas\_reales \tag{6}$$

Mide la diferencia entre el tiempo estimado y el tiempo realmente empleado en el proyecto.

9. Sostenibilidad

9.1. Reflexión

A lo largo de la carrera de Ingeniería Informática, la sostenibilidad ha sido un concepto tratado de manera superficial en la mayoría de las asignaturas, con algunas excepciones que han profundizado más en el tema. Sin embargo, no ha sido hasta este momento cuando he tomado plena conciencia de su importancia y de cuánto me queda por aprender en este ámbito.

La realización de la encuesta sobre sosteniblidad me ha permitido reflexionar sobre mi nivel de conocimiento en la materia, haciéndome consciente de que es más limitado de lo que imaginaba. A partir de este análisis, he comprendido que la sostenibilidad no se limita únicamente al impacto ambiental, sino que también abarca dimensiones sociales y económicas. Estos factores resultan fundamentales en la planificación y ejecución de cualquier proyecto, independientemente de su tamaño o alcance.

Este proceso de reflexión ha despertado en mí un mayor interés por profundizar en el estudio de la sostenibilidad y comprender cómo integrarla de manera efectiva en la gestión de proyectos dentro del ámbito de la ingeniería informática.

9.2. Dimensión Económica

El proyecto cuenta con una evaluación detallada de los costes, tanto en recursos materiales como humanos. Se han considerado los costes del personal involucrado, con una asignación clara de horas y salarios por roles, así como los gastos generales asociados al desarrollo, como el uso de un espacio de coworking y la amortización del equipo informático. Además, se han previsto costes adicionales derivados de posibles imprevistos, como fallos de hardware o la necesidad de horas extras de trabajo, asegurando un margen de contingencia del 7 %.

En cuanto a la viabilidad del proyecto en un entorno competitivo, su rentabilidad dependerá de la eficiencia obtenida en la reducción del consumo de recursos computacionales. Si las técnicas de optimización permiten ejecutar modelos de lenguaje en hardware más económico sin afectar significativamente al rendimiento, la solución podría ser altamente atractiva para empresas y usuarios con limitaciones de infraestructura.

Es posible que un proyecto similar pudiera llevarse a cabo en menos tiem-

po si los desarrolladores contaran con un conocimiento experto previo en las técnicas utilizadas. Sin embargo, los salarios en este sector presentan una alta variabilidad, lo que dificulta estimar si la reducción del tiempo de desarrollo supondría una reducción significativa del coste final. A pesar de ello, la distribución del tiempo entre tareas ha sido proporcional a su importancia, evitando dedicar esfuerzos innecesarios a partes del proyecto que pudieran haber sido reutilizadas de soluciones existentes. Además, el ahorro en memoria y eficiencia computacional que se logre con las técnicas implementadas podría traducirse en menores costos operativos a largo plazo, ya que se requeriría menos inversión en hardware especializado y se reduciría el gasto en consumo energético.

9.3. Dimensión Social

El proyecto se desarrolla en España, en el entorno académico de la Universitat Politècnica de Catalunya (UPC), dentro del ámbito de la investigación en inteligencia artificial. Dada la creciente importancia del edge computing en el procesamiento eficiente de datos, esta iniciativa contribuye a mejorar la accesibilidad a modelos avanzados de IA sin depender de infraestructuras computacionales costosas.

Existe una necesidad real de optimizar la inferencia de modelos de lenguaje, ya que actualmente requieren hardware especializado con un alto consumo energético y costes elevados. Este proyecto permitirá ejecutar estos modelos en dispositivos más accesibles, lo que beneficiará a un mayor número de usuarios y facilitará la adopción de la IA en sectores donde actualmente su uso es limitado.

La principal mejora en la calidad de vida de los usuarios será la posibilidad de utilizar modelos de lenguaje de gran tamaño sin necesidad de recurrir a servidores de alto rendimiento, reduciendo así costes y dependencias de infraestructura. El impacto social de esta solución también se extiende a la educación y a pequeñas empresas, que podrán acceder a tecnología avanzada sin inversiones significativas. Esto podría abrir nuevas oportunidades en sectores como la salud, la educación y la asistencia personal, donde el acceso a modelos avanzados de IA es actualmente limitado por los requerimientos de hardware.

Por otro lado, no se identifican colectivos directamente perjudicados por el desarrollo del TFG. Sin embargo, la optimización de estos modelos podría reducir la demanda de hardware de alto rendimiento en ciertos sectores, lo que podría afectar indirectamente a fabricantes y proveedores de estos equipos.

No obstante, esto podría incentivar el desarrollo de hardware más eficiente y asequible, beneficiando en última instancia a una mayor diversidad de usuarios y promoviendo una industria más sostenible.

9.4. Dimensión Ambiental

Los recursos necesarios para el desarrollo del proyecto incluyen el uso de un portátil y acceso a internet, dentro del entorno de un espacio de coworking. El consumo energético de estos recursos será el principal impacto ambiental durante la fase de desarrollo.

Se estima que el consumo energético del portátil durante las horas de trabajo representa una huella de carbono relativamente baja en comparación con el uso de servidores de alto rendimiento. Un portátil típico consume aproximadamente 22W en uso activo, lo que supone un consumo total estimado de 132 kWh si se considera una dedicación de 600 horas al proyecto.

Sin la existencia de este proyecto, la ejecución de modelos de lenguaje de gran tamaño seguiría dependiendo en gran medida de tarjetas gráficas de alto rendimiento, que tienen un consumo energético significativamente mayor. Este proyecto permitiría reducir la necesidad de recurrir a infraestructuras costosas y energéticamente intensivas, disminuyendo así la huella ecológica global de la inferencia en modelos de IA.

Las técnicas y estrategias de optimización empleadas en este TFG pueden ser reutilizadas en otros proyectos, maximizando el impacto positivo del trabajo realizado. En términos de reciclabilidad, al tratarse de un proyecto de software, no genera residuos físicos significativos, y las metodologías utilizadas pueden ser aplicadas en futuros desarrollos sin generar impacto ambiental adicional. Además, al hacer posible la inferencia en hardware menos potente, se reduce la necesidad de producción de nuevos dispositivos especializados, contribuyendo a la reducción de desechos electrónicos y promoviendo un ciclo de vida más largo para los dispositivos existentes.

Finalmente, con la implantación del proyecto se espera una reducción en la huella ecológica de la ejecución de modelos de lenguaje, ya que permitirá prescindir de hardware más costoso y menos eficiente en términos de consumo energético. En comparación con los métodos actuales, la optimización de la inferencia en dispositivos más modestos supone un avance hacia un uso más sostenible de la inteligencia artificial. Esto no solo beneficiará a empresas y usuarios individuales, sino que también contribuirá a la reducción global del consumo energético asociado al entrenamiento e inferencia de modelos de IA,

alineándose con las iniciativas de sostenibilidad en el ámbito tecnológico.

10. Consideraciones Legales

Dado que este proyecto se centra en la investigación y experimentación con técnicas de optimización para la inferencia de LLMs, no se enmarca directamente dentro de los ámbitos clasificados como "de alto riesgo" según el Reglamento Europeo de Inteligencia Artificial aprobado en 2024. Aun así, se han tenido en cuenta los principios generales que establece esta normativa[15].

En particular:

- No se utilizan datos personales ni información sensible durante el entrenamiento o evaluación de los modelos, por lo que no se vulnera el Reglamento General de Protección de Datos (GDPR)[16].
- Todos los *datasets* empleados son de acceso público y con licencias abiertas que permiten su uso para fines académicos y de investigación, cumpliendo así con la normativa sobre propiedad intelectual y derechos de autor.
- No se hace uso de los modelos con fines comerciales, ni en aplicaciones críticas (sanidad, justicia, vigilancia, etc.), lo cual sitúa el proyecto fuera de los marcos regulativos más estrictos definidos por la legislación europea.

En resumen, el proyecto cumple con la normativa vigente en materia de inteligencia artificial y protección de datos dentro del contexto de la Unión Europea.

11. Conocimientos Previos

11.1. Cómo funciona un transformador

Para trabajar con las técnicas que aplicaremos usaremos el LLM Llama de Meta (Facebook), que está basado en la arquitectura de Transformadores introducida originalmente en el artículo "Attention is All You Need"[17]. Al haber tenido que aprender de cero sobre este tema durante el desarrollo del TFG me ha parecido interesante hacer una explicación simplificada de cómo funcionan los transformadores.

11.1.1. Transformadores

A partir de la idea de estos transformadores se pueden construir distintos tipos de modelo como a partir de un audio generar su transcripción, o viceversa, a partir de un texto generar un audio, o también a partir de un texto generar una imagen. También pueden ser usados para a partir de un texto generar una imagen, traducir textos o generarlos.

Aunque la función principal con la que se ideó este transformer era la traducción de texto de un idioma a otro nos centraremos en la generación de texto, ya que es en lo que este proyecto utilizaremos y haremos pruebas sobre ello.

La idea es que dado un conjunto de palabras, intenta predecir qué palabra puede ser la siguiente. Esta predicción toma la forma de una función de probabilidad. Podemos ver un ejemplo simple para entenderlo mejor en la siguiente figura[3]:

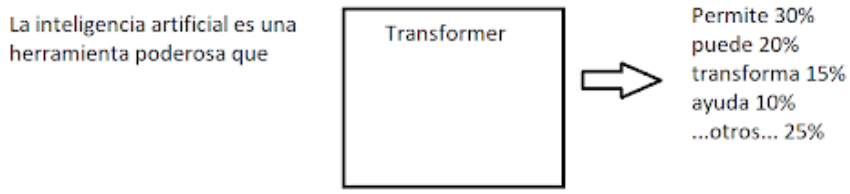

Figura 3: Imagen funcionamiento básico Transformadores (Elaboración propia)

Este proceso se compone de varios pasos, y para comprender cómo funciona realmente, los iremos analizando uno por uno. De esta forma, podremos

entender con claridad qué sucede en cada etapa y cómo contribuye al resultado final.

Además, también analizaremos de dónde proviene el número de parámetros que tienen los modelos, ya que este número influye directamente en el tamaño del modelo. Como ejemplo, utilizaremos GPT-3 para observar, en cada fase, cuántos parámetros intervienen y de dónde salen exactamente. GPT-3 en total cuenta con 175 mil millones de parámetros.

11.1.2. Word embedding (y tokenización)

Lo primero que hace el modelo es procesar el texto de entrada dividiéndolo en unidades más pequeñas llamadas tokens. Aunque para simplificar a menudo se dice que estas unidades son "palabras", en realidad suelen ser subpalabras, caracteres o incluso combinaciones frecuentes de letras. Esta tokenización depende del método utilizado por el modelo.

Ejemplo: La frase "transformadores automáticos"podría dividirse en los siguientes tokens:

- "transformadores"→ "transform", "adores"
- "utomáticos"→ "auto", "máticos"

Esta división permite que el modelo maneje eficientemente palabras nuevas o raras, y reduce el tamaño del vocabulario total que necesita aprender.

El modelo tiene un vocabulario predefinido que contiene todos los tokens que puede reconocer. Cada uno de estos tokens está asociado a un vector numérico de tamaño fijo, conocido como su vector de embedding.

- Estos vectores no son aleatorios. Se entrenan junto con el modelo y están organizados de forma que tokens con significados similares tengan vectores cercanos en el espacio vectorial.
- Así, palabras como "gato" y "felino" estarán representadas por vectores relativamente cercanos, mientras que "gato" y "avión" estarán más alejados.

Representación matemática

Si el vocabulario tiene tamaño V y los vectores tienen dimensión d, el *embedding* es una matriz $E \in \mathbb{R}^{V \times d}$.

Cada token se representa como una fila de esa matriz.

Ejemplo: Supongamos que el modelo recibe el texto:

"El gato maúlla"

1. **Tokenización:** "El", "gato", "maúlla" → Tokens $[101, 457, 892]$ (números arbitrarios de ejemplo).
2. **Embedding:** Se buscan los vectores correspondientes en la tabla de embeddings:
   - Token 101 → vector $\vec{e}_1$
   - Token 457 → vector $\vec{e}_2$
   - Token 892 → vector $\vec{e}_3$

Como resultado, obtenemos una secuencia de vectores:

$$[\vec{e}_1, \vec{e}_2, \vec{e}_3] \in \mathbb{R}^{3\times d}$$

Esta secuencia vectorial es la que se pasa a las siguientes capas del modelo Transformer, donde se aplicarán los siguientes mecanismos. Pero antes de ir a estos, hablaremos un poco más sobre el *word embedding*.

Significado semántico de los embeddings

Como se ha mencionado antes, los vectores de embedding acaban adquiriendo cierto significado semántico durante el entrenamiento. Es decir, los vectores de palabras con significados similares tienden a tener direcciones parecidas en el espacio vectorial.

Ejemplo ilustrativo:

- $\text{Vector(hombre)} - \text{Vector(mujer)} \approx \text{Vector(padre)} - \text{Vector(madre)}$

Esto significa que, incluso si no conocemos la palabra "madre", podemos aproximarla de esta forma:

$$\text{Vector(madre)} \approx \text{Vector(padre)} - (\text{Vector(hombre)} - \text{Vector(mujer)})$$

Esta propiedad es una de las razones por las que los embeddings son tan poderosos en tareas de procesamiento de lenguaje natural.

Parámetros hasta el momento

Los primeros parámetros que calcularemos salen de esta fase, en este caso es la Embedding matrix, W_e. El tamaño del vocabulario de GPT-3 es de 50,257 tokens, y la dimensión de los vectores de cada token es de 12,288, por lo que el total de parámetros de la matriz W_e es de 12,288×60,257=617,558,016 parámetros.

Matriz	**n parámetros (fórmula)**	**Resultado**
Embedding	$12,288 \times 50,257$	617,558,016

Cuadro 8: Número de parámetros para la matriz de embedding (elaboración propia)

El patrón de atención

Es la forma en que el modelo decide qué palabras mirar cuándo está procesando una palabra. Este proceso está dividido en varias cabezas de atención.

Una cabeza de atención es una parte del modelo que ayuda a cada palabra (o token) a decidir a qué otras palabras debe prestar más atención para entender mejor el significado de la frase.

En lugar de tratar todas las palabras por igual, esta cabeza aprende a dar más importancia a las palabras que son más relevantes para cada una. Así, puede construir una representación más útil de cada palabra teniendo en cuenta su contexto.

Por ejemplo, una cabeza de atención podría encargarse de dar atención a los adjetivos de un sustantivo:

- En la frase "La casa blanca y antigua", el modelo podría aprender que la palabra "casa" debe prestar más atención a "blanca" y "antigua", ya que son adjetivos que la describen.

Otra cabeza de atención, en cambio, podría especializarse en seguir relaciones gramaticales, como entre sujetos y verbos. Por ejemplo:

- En "El perro corre", esa cabeza podría ayudar al modelo a entender que "perro" es quien corre, relacionando correctamente el sujeto con su acción.

El patrón de atención

Para cada cabeza de atención tenemos 3 matrices, la Query(Q), la Key(K), y la Value(V):

- **Query (Q):** es la "pregunta" que hace el sustantivo para descubrir qué adjetivos lo describen. **Analogía:** Imagina que la palabra "casa" lleva una tarjeta que dice: "¿Quién me está describiendo con adjetivos?"
- **Key (K):** es la "etiqueta" que lleva cada posible adjetivo para indicar qué tipo de descripción ofrece.

  Analogía: Cada adjetivo cuelga un cartelito en el que pone: "Ofrezco información de color" (por ejemplo, "blanca") "Ofrezco información de tamaño" (por ejemplo, "grande")
- **Value (V):** es la "información real" del adjetivo, es decir, la palabra en sí y su matiz completo. **Analogía:** Dentro de la tarjeta del adjetivo está escrito el adjetivo completo, como "blanca" o "grande", junto a todo lo que connota (luminosidad, volumen, etc.).

$$
\begin{aligned}
Q &= EW_q \\
K &= EW_k \\
V &= EW_v
\end{aligned}
$$

donde W_q, W_k, W_v son matrices aprendibles. Cada embedding se transforma así en un vector Query, Key y Value de dimensión, por ejemplo, d_q o d_k.

A continuación, para cada par (query, key) se calcula una puntuación de compatibilidad mediante el producto escalar: si denotamos Q de tamaño $n \times d_q$ y K de tamaño $n \times d_k$, obtenemos una matriz de puntuaciones de atención de tamaño $n \times n$ como:

$$\text{scores} = QK^\top$$

Para estabilizar el entrenamiento (y evitar valores de producto escalar demasiado grandes cuando d_k es grande), se suele escalar dividiendo por $\sqrt{d_k}$:

$$\text{scores\_escalados} = \frac{QK^\top}{\sqrt{d_k}}$$

Luego, sobre cada fila de esta matriz de puntuaciones (correspondiente a una query) se aplica `softmax` para obtener una distribución de pesos de atención:

$$A = \text{softmax}(\text{scores\_escalados}) \quad \text{(aplicado fila a fila)}$$

Cada elemento A_{ij} indica cuánto debe "atender" la posición i (query) a la posición j (key). Finalmente, se multiplica la matriz de pesos de atención A por los valores V:

$$\text{Output} = AV$$

produciendo una nueva representación para cada posición i, que es una combinación ponderada de los valores de todas las posiciones. Este mecanismo de atención se denomina "scaled dot-product attention".

Gracias a que las matrices W_q, W_k, W_v se optimizan durante el entrenamiento, el modelo aprende automáticamente qué relaciones y qué patrones de atención son útiles en diversos contextos lingüísticos o en otro tipo de secuencias.

Parámetros hasta el momento

Podemos ver que durante este proceso intervienen las 3 matrices anteriormente mencionadas, la Query, la Key, y la Value. En GPT3 la matriz Query y Key son del mismo tamaño, 12,288x128. La matriz Value, en cambio, tiene una pequeña trampa; aunque por la forma como se la ha descrito parece una matriz de 12,288x12,1288, en realidad está compuesta por el producto de 2 matrices, una de 12,288x128 y otra 128x12,288 (en realidad no funciona exactamente así, pero lo explicaremos de esta manera para simplificar el entenderlo). Esto sería calculado para una cabeza de atención, pero recordemos que tenemos más de una cabeza; en concreto, tenemos 96 cabezas de atención por capa, y en total tenemos 96 capas.

¿Cómo "almacenan" información los transformadores?

Para almacenar y transformar información, los transformadores utilizan bloques conocidos como MLP (Multilayer Perceptron). Un bloque MLP es, en esencia, una red neuronal feedforward que se aplica de forma independiente a cada vector de token, sin interacción entre ellos. Es decir, es una operación puramente local en cada posición, realizada en paralelo para todos los tokens.

Matriz	Fórmula	n parámetros
Embedding	$12,288 \times 50,257$	617,558,016
Query	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
Key	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
Value1	$128 \times 12{,}288 \times 96 \times 96$	14,495,514,624
Value2	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624

Cuadro 9: Parámetros de distintas matrices hasta attention head (elaboración propia)

Proceso dentro del MLP

Supongamos que tenemos el texto "Donald Trump es presidente" y que este ya ha pasado por un bloque de atención. En este punto, el vector correspondiente al token "Trump" puede haber incorporado información contextual sobre "Donald" gracias al mecanismo de atención.

El bloque MLP comienza multiplicando ese vector (denotado como E) por una primera matriz de pesos (aprendida durante el entrenamiento). Esta matriz puede interpretarse como un conjunto de filtros: si la representamos por filas Figura[4], cada fila define una 'dirección semántica' en el espacio del embedding.

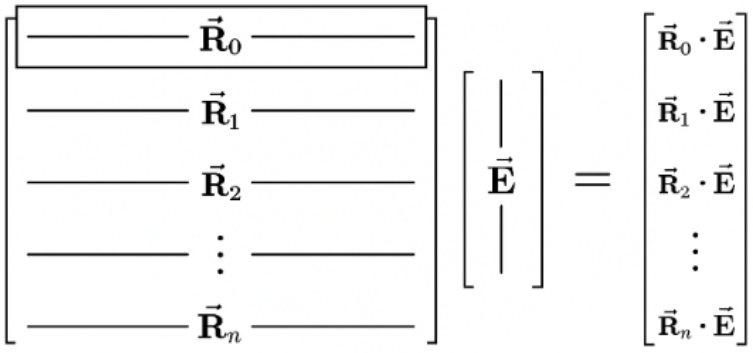

Figura 4: Imagen funcionamiento bloque MLP 1 (Elaboración propia)

Supongamos que la primera fila tiene dirección con significado "nombre Donald" y entonces el producto escalar entre R0 y E podríamos decir que si es más o menos igual a 1 E codifica "nombre Donald" y de lo contrario si es menor o igual a 0 no lo hace. Podemos imaginar también que la primera fila tiene la dirección de "primer nombre Donald y apellido Trump".

El resultado de esta multiplicación es un nuevo vector, al que se le suma

un vector de sesgo (bias), también aprendido. Luego, este vector pasa por una función de activación no lineal, como ReLU, que anula componentes negativos y permite identificar qué neuronas están activas (es decir, qué aspectos del significado están presentes).

$$n_0 + n_1 \mathrm{C}_1 + n_2 \mathrm{C}_2 + \ldots$$

$$\begin{bmatrix} | & | & | & | & | & & | \\ \vec{\mathbf{C}}_0 & \vec{\mathbf{C}}_1 & \vec{\mathbf{C}}_2 & \vec{\mathbf{C}}_3 & \vec{\mathbf{C}}_4 & \cdots & \vec{\mathbf{C}}_m \\ | & | & | & | & | & & | \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ \vdots \\ n_m \end{bmatrix} + \begin{bmatrix} | \\ \vec{\mathbf{B}} \\ | \end{bmatrix} =$$

Figura 5: Imagen funcionamiento bloque MLP 2 (Elaboración propia)

Después de la activación, se realiza una segunda multiplicación por otra matriz de pesos. Esta vez, conviene pensar en la matriz en términos de sus columnas, ya que cada una tiene la misma dimensión que el embedding. Podemos imaginar que algunas columnas Figura[5] representan conceptos como "presidente", "EE. UU." , o incluso combinaciones como "presidente estadounidense". Si una neurona del vector anterior está activa, su valor determinará cuánto se suma esa columna al resultado final. Finalmente, se añade otro vector de sesgo.

Así, si el vector de entrada codificaba algo como "Nombre Donald + Apellido Trump", el MLP puede transformarlo en un vector que incluya dimensiones como "presidente" o "EE. UU.", en función de lo aprendido durante el entrenamiento. Esta salida se suma al vector original, permitiendo que el modelo conserve parte del significado original mientras enriquece su representación con nueva información inferida.

Parámetros hasta el momento

Nota: también se tendrían que tener en cuenta los parámetros del BIAS pero son tan pocos que no hace falta que los contemos.

Obtención del "resultado"

Ahora que hemos visto los distintos elementos que intervienen en el transformador, vamos a repasar de forma general cómo es el proceso completo y

Matriz	**Fórmula**	**n parámetros**
Embedding	$12,288 \times 50,257$	617,558,016
Query	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
Key	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
Value1	$128 \times 12{,}288 \times 96 \times 96$	14,495,514,624
Value2	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
MLP up_proj	$49{,}152 \times 12{,}288 \times 96$	57,982,058,496
MLP down_proj	$12{,}288 \times 49{,}152 \times 96$	57,982,058,496

Cuadro 10: Parámetros de distintas matrices hasta MLP (elaboración propia)

cómo se acaba obteniendo el resultado final.

Primero de todo, el texto de entrada se divide en tokens, que son fragmentos de palabras o palabras completas dependiendo del vocabulario del modelo. A cada token se le asigna un vector numérico mediante una operación llamada embedding, que convierte estos tokens en representaciones en un espacio continuo de alta dimensión. Este embedding incluye además información posicional, para que el modelo sepa en qué orden aparecen los tokens.

En el caso de modelos como GPT, el bloque transformador está compuesto por 96 capas, y cada una de estas capas contiene 96 cabezas de atención (attention heads). Estas cabezas permiten que el modelo enfoque su atención en distintas partes del contexto al procesar cada token, extrayendo relaciones complejas entre ellos.

Después de pasar secuencialmente por estas capas, el modelo obtiene una representación final para cada token. En esta etapa entramos en la última fase: el unembedding.

El unembedding es el proceso inverso al embedding. Mientras que el embedding convierte tokens en vectores, el unembedding convierte vectores en distribuciones de probabilidad sobre todos los posibles tokens del vocabulario. Esto se hace mediante la multiplicación del vector de salida por la matriz de unembedding, seguida de la aplicación de una función softmax. El resultado es una probabilidad para cada token posible, indicando cuál es el más probable que venga a continuación. Finalmente, el modelo selecciona el token con mayor probabilidad (o aplica una estrategia como sampling o top-k para decidirlo) y lo genera como salida.

Este proceso se repite secuencialmente para generar los siguientes tokens,

uno por uno, hasta completar la respuesta del modelo.

Parámetros hasta el momento

Matriz	Fórmula	n parámetros
Embedding	$12,288 \times 50,257$	617,558,016
Query	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
Key	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
Value1	$128 \times 12{,}288 \times 96 \times 96$	14,495,514,624
Value2	$12{,}288 \times 128 \times 96 \times 96$	14,495,514,624
MLP up_proj	$49{,}152 \times 12{,}288 \times 96$	57,982,058,496
MLP down_proj	$12{,}288 \times 49{,}152 \times 96$	57,982,058,496
Unembedding	$50,257 \times 12,288$	617,558,016

Cuadro 11: Parámetros de distintas matrices final (elaboración propia)

Total de parámetros: 175,181,291,520 $\rightarrow$ 175 mil millones 181 millones 291 mil 520.

Así podemos ver de dónde salen el número de parámetros del modelo y cómo estos actúan dentro de los transformadores.

11.2. Perplexity en los LLMs

La **perplexity** (*perplejidad*)[18] es una métrica utilizada para evaluar modelos de lenguaje. Intuitivamente, mide cuán sorprendido está el modelo al ver una secuencia de palabras. Un valor de perplexity más bajo indica que el modelo predice mejor las palabras de una secuencia; es decir, tiene mayor capacidad para modelar el lenguaje.

Interpretación

La perplexity puede entenderse como el número promedio de palabras que el modelo considera posibles en cada paso de predicción. Por ejemplo, una perplexity de 50 indica que, en promedio, el modelo está tan inseguro como si tuviera que elegir entre 50 palabras igualmente probables.

Cabe destacar que en la práctica, el modelo asigna diferentes probabilidades a cada palabra. La perplejidad mide el "efecto promedio" de esas probabilidades variables, y sólo equivale al tamaño de vocabulario o al número de "opciones" cuando dichas probabilidades son idénticas.

Una perplejidad de 50 significa que, en términos de poder predictivo, el modelo actúa como si tuviera unas 50 opciones igualmente probables en cada paso. Pero esas 50 "opciones" son un promedio geométrico de las probabilidades reales, no un conteo literal.

Cálculo de la perplexity

Dado un modelo de lenguaje que asigna una probabilidad $P(w_1, w_2, \ldots, w_N)$ a una secuencia de palabras $w_1, w_2, \ldots, w_N$, la perplejidad se define como:

$$\text{perplexity} = P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i \mid w_1, \ldots, w_{i-1})\right)$$

donde:

- N es el número total de palabras en la secuencia.
- $P(w_i \mid w_1, \ldots, w_{i-1})$ es la probabilidad condicional asignada por el modelo a la palabra w_i dado el contexto anterior.

Funcionamiento

Para calcular la perplexity:

1. Se pasa una secuencia de texto por el modelo.
2. El modelo predice la probabilidad de cada palabra en la secuencia, dada su historia previa.
3. Se calcula el logaritmo de las probabilidades y se promedia.
4. Finalmente, se aplica la exponencial negativa del promedio de log-probabilidades.

Conclusión

La perplexity es una medida crucial en el entrenamiento y evaluación de modelos de lenguaje. Permite comparar distintos modelos y ajustes de hiperparámetros de forma cuantitativa. Sin embargo, una buena perplexity no siempre implica una buena calidad perceptiva del texto generado, por lo que se suele complementar con otras métricas o evaluaciones humanas.

12. Metodología

12.1. Métricas de medición

Perplexity

La perplexity, como se ha dicho en el apartado anterior, es una métrica estándar para evaluar modelos de lenguaje. Mide qué tan bien un modelo predice una secuencia de texto, ver Algoritmo 25.

- Una menor perplexity indica que el modelo es mejor al predecir la secuencia, lo que refleja una mejor calidad lingüística.
- Esta métrica se utilizará para asegurarse de que las técnicas de reducción de memoria no degraden significativamente la precisión del modelo. Porque aun no siendo el objetivo principal, también hay que tenerlo en cuenta.

Top-K Accuracy (TopK5)

La precisión Top-K (TopK5) mide si la palabra correcta aparece dentro de las K predicciones más probables del modelo en cada paso de inferencia. En este caso, se evalúa con K = 5.

- Esta métrica es útil para observar si el modelo sigue siendo útil, aunque la palabra exacta no sea la más probable, pero esté dentro del conjunto relevante.
- Se utiliza para complementar la perplejidad y capturar más información sobre la calidad de predicción.
- Inicialmente se planteaba usar accuracy, pero al ser un modelo bastante "pequeño"la métrica resultante era muy poco informativa y se optó por esta.

Media de los picos de memoria

Se evalúa la media de los picos de memoria a lo largo de varias inferencias realizadas sobre diferentes entradas.

- Esta métrica sirve para obtener una visión general del comportamiento de uso de memoria bajo diferentes escenarios.
- Gracias a esta nos podemos hacer una idea de los requisitos mínimos que necesita el dispositivo para poder ejecutar el LLM.

Pico de memoria más alto

El pico máximo de memoria representa el valor más alto de uso de memoria alcanzado durante cualquier punto del proceso de inferencia.

- Es una métrica crítica para dispositivos con límite físico estricto, ya que superar ese límite implica fallos o caídas del sistema.
- Permite identificar configuraciones que podrían ser inviables.

Latencia

La latencia es el tiempo que tarda el modelo en generar una respuesta desde el momento en que recibe una entrada hasta que produce una salida.

- Se mide por inferencia, y su optimización es importante para aplicaciones en tiempo real o con interacción directa con el usuario.
- En el estudio se evalúa cómo las optimizaciones de memoria afectan esta latencia.

TTFT (Time To First Token)

El TTFT mide el tiempo que tarda el modelo en generar el primer token después de recibir la entrada.

- Es una métrica sensible al preprocesamiento, carga del modelo y el pipeline inicial.
- Es especialmente relevante para aplicaciones donde la respuesta parcial puede empezar a mostrarse antes de completarse.

TPOT (Time Per Output Token)

El TPOT representa el tiempo promedio que tarda el modelo en generar cada token, después del primero.

- A diferencia del TTFT, TPOT mide la eficiencia del modelo durante la generación continua.
- Esta métrica permite observar cuán sostenibles son las optimizaciones en la generación a largo plazo.

12.2. Códigos

12.2.1. Cálculo de Perplexity

Algorithm 1 Cálculo de NLL sobre secuencia de texto

Require: Texto de entrada *text*
Ensure: Suma de log-verosimilitud negativa *nll_sum* y número de tokens *n_tokens*

encodings ← tokenizer(*text*) con truncado a 5000 tokens
input_ids ← *encodings*["*input_ids*"]
seq_len ← longitud de *input_ids*
nll_sum ← 0,0
n_tokens ← 0
prev_end_loc ← 0
for *begin_loc* = 0 **to** *seq_len* **step** *stride* **do**
    end_loc ← mín(*begin_loc* + *max_length*, *seq_len*)
    trg_len ← *end_loc* − *prev_end_loc*
    chunk_ids ← *input_ids*[:, *begin_loc* : *end_loc*]
    target_ids ← clone(*chunk_ids*)
    target_ids[:, : −*trg_len*] ← −100 ▷ Ignorar tokens fuera del objetivo
    Evaluar modelo sin gradientes:
        loss ← model(*chunk_ids*, *labels* = *target_ids*).*loss*
    num_valid ← conteo de tokens ≠ −100
    batch_sz ← tamaño del batch
    num_loss_tokens ← *num_valid*
    nll_sum ← *nll_sum* + *loss* × *num_loss_tokens*
    n_tokens ← *n_tokens* + *num_loss_tokens*
    prev_end_loc ← *end_loc*
    if *end_loc* == *seq_len* **then**
        break
    end if
end for
return *nll_sum*, *n_tokens*

Este código es usado para calcular la perplexity para un solo texto. Su funcionamiento es el siguiente:

- **Tokenización:** El texto se convierte a `input_ids` mediante el tokenizador, truncándolo a un máximo de 5000 tokens y pasándolo a tensores de PyTorch en el dispositivo correspondiente.

- **Inicialización:** Se inicializan las variables acumuladoras para la suma de la log-verosimilitud negativa (`nll_sum`) y el número total de tokens evaluados (`n_tokens`).
- **Procesamiento por bloques:** El texto se recorre en bloques de longitud `max_length`, con solapamiento controlado por `stride`. Cada bloque es una ventana deslizante sobre el texto original. La variable `trg_len` indica cuántos tokens nuevos (no solapados desde el bloque anterior) se evalúan en cada iteración.
- **Preparación del objetivo:** Se enmascaran todos los tokens anteriores a `trg_len` con el valor `-100`, para que no contribuyan al cálculo de la pérdida. Solo se evalúan los nuevos tokens del bloque actual.
- **Cálculo de la pérdida:** Se pasa el bloque al modelo en modo evaluación y se obtiene la pérdida de tipo `CrossEntropyLoss`, correspondiente a la log-verosimilitud negativa media por token evaluado.
- **Acumulación de métricas:** La pérdida obtenida se multiplica por el número de tokens válidos y se suma a `nll_sum`. También se acumula el total de tokens en `n_tokens`.
- **Finalización:** El bucle se detiene una vez se ha alcanzado el final del texto. Finalmente, la función retorna la suma total de la log-verosimilitud negativa y el número total de tokens evaluados, lo que permite calcular la *perplejidad* más adelante.

12.2.2. Cálculo de Top-k Accuracy

Algorithm 2 Cálculo de `Top-k Accuracy` sobre texto

Require: Texto de entrada *text*, tamaño del top-k, longitud máxima L_{max}, y stride
Ensure: Número de aciertos y total de predicciones
1: Codificar el texto: *input_ids* ← `tokenizer`(*text*)
2: Mover a dispositivo de cómputo: *input_ids* ← *input_ids*.`to(device)`
3: Obtener longitud de la secuencia: *seq_len* ← `size`(*input_ids*)
4: Inicializar: *correct* ← 0, *total* ← 0
5: **for** *start* = 0 to *seq_len* − 1 con paso *stride* **do**
6: *end* ← mín(*start* + L_{max}, *seq_len*)
7: *input_chunk* ← *input_ids*[:, *start* : *end*]
8: *labels* ← *input_chunk*[:, 1 :] ▷ Tokens objetivo (tokens siguientes)
9: *inputs* ← *input_chunk*[:, : −1] ▷ Tokens de entrada (sin el último)
10: *logits* ← `model`(*inputs*).`logits`
11: Obtener top-k índices: *topk* ← `topk`(*logits*, k)
12: Comparar predicciones con los objetivos:
match ← (*topk* == *labels.unsqueeze*(−1)).`any(dim=-1)`
13: *correct* ← *correct* + `sum`(*match*)
14: *total* ← *total* + `numel`(*match*)
15: **if** *end* == *seq_len* **then**
16: **break**
17: **end if**
18: **end for**
19: **return** *correct*, *total*

Esto código es usado para calcular la Top-k Accuracy para un solo texto, su funcionamiento es el siguiente:

- **Tokenización:** El texto se convierte en `input_ids` utilizando el tokenizador. No se aplica truncado por defecto, lo que permite obtener todos los tokens. Los tensores se mueven al dispositivo correspondiente.
- **Inicialización:** Se inicializan dos contadores: `correct`, que almacena el número de predicciones correctas, y `total`, que acumula el número total de intentos de predicción.
- **Procesamiento por bloques:** La secuencia se recorre en bloques deslizantes de tamaño máximo `max_length`, avanzando en saltos de longi-

tud `stride`. Este enfoque permite procesar secuencias largas sin superar los límites de memoria del modelo.

- **Preparación de entrada y etiquetas:** Para cada bloque, se separan los tokens de entrada (`inputs`) y las etiquetas esperadas (`labels`) desplazando la secuencia una posición hacia la derecha. Esto permite evaluar la capacidad del modelo para predecir el siguiente token.
- **Predicción:** El bloque de entrada se pasa por el modelo, obteniendo una distribución de probabilidad (logits) sobre el vocabulario en cada posición.
- **Extracción del top-k:** Se seleccionan los `k` tokens con mayor probabilidad en cada posición del bloque mediante la operación `torch.topk`.
- **Comparación y evaluación:** Se comprueba si la etiqueta real de cada posición aparece dentro del conjunto `top-k` correspondiente. Si está presente, se considera un acierto.
- **Acumulación de métricas:** Se suman los aciertos al contador `correct` y se añade el número total de predicciones realizadas al contador `total`.
- **Finalización:** El bucle se detiene una vez se ha procesado todo el texto. La función retorna el número total de aciertos y el número total de intentos, lo que permite calcular la *top-k accuracy* como accuracy = $\frac{\text{correct}}{\text{total}}$.

12.2.3. Cálculo de memoria usada

Algorithm 3 Medición del pico de memoria durante la inferencia

Require: Texto de entrada *text*
Ensure: Memoria máxima utilizada durante la generación, en GB

1: **Sin gradientes:** Desactiva el cálculo de gradientes con `torch.no_grad()`
2: Tokenizar *text* con truncamiento y padding hasta 2048 tokens
3: Mover los tensores al `device`
4: Reiniciar estadísticas de memoria pico de PyTorch en `device`
5: Llamar a `model.generate` con:
   - `input_ids`, `attention_mask`
   - `max_new_tokens`
   - `do_sample = False`
   - `pad_token_id = tokenizer.eos_token_id`
6: Obtener memoria pico usada en bytes con `torch.cuda.max_memory_allocated`
7: Convertir a GB: peak_gb $\leftarrow$ peak_bytes/$1e9$
8: **return** peak_gb

Esto código es usado para calcular la memoria máxima usada en una inferencia, su funcionamiento es el siguiente:

- **Desactivación de gradientes:** Se usa el contexto `torch.no_grad()` para evitar el cálculo y almacenamiento de gradientes, ya que solo se realiza inferencia.
- **Tokenización:** El texto se convierte en tensores con el tokenizador. Se trunca hasta 2048 tokens y se aplica padding. El resultado se mueve al dispositivo correspondiente.
- **Reinicio de estadísticas de memoria:**
  Se llama a `torch.cuda.reset_peak_memory_stats` para reiniciar la métrica de uso máximo de memoria antes de comenzar la inferencia.
- **Inferencia:** Se realiza una llamada a `model.generate`, generando una secuencia sin muestreo (modo determinista) y controlando el número máximo de tokens nuevos a generar. Se especifica el token de padding como el `eos_token_id`.

- **Medición del uso de memoria:** Una vez terminada la generación, se consulta la cantidad máxima de memoria GPU usada durante todo el proceso con `torch.cuda.max_memory_allocated`.
- **Conversión a GB:** El resultado, originalmente en bytes, se convierte a gigabytes dividiéndolo por 10^9.
- **Devolución del resultado:** Finalmente, la función retorna la memoria pico usada en GB, lo que permite evaluar la eficiencia del modelo en cuanto a consumo de recursos.

12.2.4. Cálculo de latencias

Algorithm 4 Medición de tiempos de generación: TTFT, latencia total y TPOT

Require: Texto de entrada *text*
Ensure: Tiempo para el primer token (`TTFT`), latencia total, y tiempo por token (`TPOT`)

Tokenizar *text* con truncación hasta 2048 tokens
Mover los `input_ids` al `device`
Sincronizar GPU con `torch.cuda.synchronize`
Iniciar cronómetro `start_ttft`
Inferencia: Generar 1 token (TTFT)
Sincronizar GPU y medir duración como `ttft`
Sincronizar GPU y comenzar `start_total`
Inferencia: Generar hasta `max_new_tokens` tokens
Sincronizar GPU y medir `total_latency`
Calcular número de tokens generados: $n = \text{len(output)} - \text{len(input)}$
if $n > 1$ **then**
    $tpot \leftarrow \frac{\text{total\_latency} - \text{ttft}}{n-1}$
else
    $tpot \leftarrow 0{,}0$
end if
return `ttft`, `total_latency`, `tpot`

Esto código es usado para calcular distintos tipos de latencias usadas en una inferencia, su funcionamiento es el siguiente:

- **Tokenización:** El texto se tokeniza con truncamiento hasta 2048 tokens, y se convierte a tensores para pasar al dispositivo de ejecución (por ejemplo, GPU).

- **Sincronización inicial:** Se sincroniza la GPU para asegurar que no haya operaciones pendientes antes de iniciar la medición de tiempo.
- **Medición de TTFT (Time To First Token):**
  - Se registra el tiempo de inicio.
  - Se realiza una generación de solo 1 token.
  - Se sincroniza la GPU y se calcula la duración, que representa cuánto tarda el modelo en generar el primer token.
- **Medición de latencia total:**
  - Se repite el proceso de sincronización.
  - Se mide el tiempo que tarda en generar hasta `max_new_tokens`.
  - La duración completa de la generación se guarda como `total_latency`.
- **Cálculo de TPOT (Time per Output Token):**
  - Se calcula cuántos tokens se generaron más allá del primero.
  - Si hay más de uno, se calcula el tiempo promedio por token adicional (excluyendo el primero).
- **Resultado final:** La función retorna los tres tiempos principales:
  - `ttft`: tiempo hasta el primer token.
  - `total_latency`: tiempo total de inferencia.
  - `tpot`: tiempo promedio por token adicional.

12.2.5. Pruning

Para el pruning tenemos varias funciones que vamos a ir viendo, estas han sido obtenidas del repositorio de github [19].

Algorithm 5 Cálculo de Importancia de Pares de Neuronas

Require: Matrices de pesos `gate_weight` y `up_weight`
Ensure: Vector `importance_scores` con la importancia por neurona
1: `gate_max_abs` ← máximo por fila de `gate_weight` + valor absoluto del mínimo por fila
2: `up_max_abs` ← máximo por fila de `up_weight` + valor absoluto del mínimo por fila
3: `importance_scores` ← `gate_max_abs` + `up_max_abs`
4: **return** `importance_scores`

- **Entrada:** El algoritmo recibe dos matrices: `gate_weight` y `up_weight`, correspondientes a los pesos de las capas `gate_proj` y `up_proj`.
- **Cálculo del peso absoluto máximo:** Para cada fila (es decir, cada neurona) en ambas matrices, se obtiene el valor absoluto más grande, sumando el máximo positivo y el módulo del mínimo negativo. Esta operación permite capturar la fuerza total del peso, independientemente del signo.
- **Importancia del par de neuronas:** Se suman los valores obtenidos de ambas matrices por fila. Esto representa una medida compuesta de la .ªctivación potencial"del par de neuronas gate + up.
- **Salida:** Se retorna un vector con un valor de importancia para cada par de neuronas.

Algorithm 6 Pruning de Neuronas del MLP

Require: Módulo `mlp`, porcentaje a eliminar `prune_percent`
Ensure: Nuevas capas `gate_proj`, `up_proj`, `down_proj` con menor dimensión, y el nuevo tamaño intermedio `k`

1: Obtener `gate_weight` y `up_weight` desde `mlp`
2: Calcular `importance_scores` usando `compute_neuron_pair_importance`
3: `original_intermediate_size` ← número de filas de `gate_weight`
4: `num_neuron_pairs_to_prune` ← `prune_percent` × `original_intermediate_size`
5: `k` ← `original_intermediate_size` − `num_neuron_pairs_to_prune`
6: **if** `k` ≤ 0 **then**
7: **throw** Error: `k` inválido
8: **end if**
9: Obtener índices de las `k` neuronas más importantes con `topk`
10: Crear nuevas capas lineales reducidas con tamaño intermedio `k`
11: Copiar los pesos seleccionados desde las capas originales
12: **return** nuevas capas y nuevo tamaño intermedio `k`

- **Entrada:** El procedimiento recibe un bloque MLP (con subcapas `gate_proj`, `up_proj`, `down_proj`) y un porcentaje de pruning (`prune_percent`).
- **Extracción de pesos:** Se extraen los tensores de pesos de las capas `gate_proj` y `up_proj`, que son usados para calcular la importancia relativa de cada neurona.

- **Cálculo de importancia:** Se aplica una función que mide la importancia de cada par de neuronas basándose en sus pesos máximos absolutos en ambas capas. Esto permite priorizar las neuronas que más contribuyen a la activación.
- **Selección de neuronas a conservar:** Se calcula cuántas neuronas deben eliminarse, y cuántas deben conservarse. Se verifica que el número resultante no sea inválido (es decir, que no se eliminen todas).
- **Filtrado de neuronas:** Se obtienen los índices de las neuronas más importantes usando `torch.topk`, ordenando los índices para conservar el orden original.
- **Creación de nuevas capas:** Se inicializan nuevas capas lineales con tamaño intermedio reducido en base a los índices seleccionados.
- **Copia de pesos:** Se transfieren los pesos de las neuronas seleccionadas a las nuevas capas, manteniendo la conectividad estructural correcta.
- **Salida:** Se retornan las tres nuevas capas pruned (`gate_proj`, `up_proj`, `down_proj`) y el nuevo tamaño intermedio para integrarlo en el modelo original.

Algorithm 7 Pruning de Neuronas en Todas las Capas del Modelo

Require: Modelo `model`, porcentaje de pruning `prune_percent`
Ensure: Modelo con capas `MLP` actualizadas y tamaño intermedio modificado

Inicializar `new_intermediate_size` como `None`
for cada `layer` en `model.model.layers` **do**
    `mlp` ← `layer.mlp`
    `new_gate_proj`, `new_up_proj`, `new_down_proj`, `new_size` ← `prune_neuron_pairs(mlp, prune_percent)`
    Reemplazar `mlp.gate_proj`, `up_proj`, `down_proj` con las capas nuevas
    if `new_intermediate_size` es `None` **then**
        `new_intermediate_size` ← `new_size`
    end if
end for
Actualizar `model.config.intermediate_size` con `new_intermediate_size`
return `model`

- **Propósito:** Esta función recorre todas las capas del modelo, aplicando poda estructural (pruning) a los bloques MLP internos, reduciendo el tamaño intermedio de las proyecciones.
- **Inicialización:** Se define `new_intermediate_size` como `None`, ya que el tamaño reducido será el mismo en todas las capas y solo necesita calcularse una vez.
- **Iteración sobre capas:** El bucle recorre todas las capas del modelo accediendo a `model.model.layers`.
- **Selección del bloque MLP:** Dentro de cada capa, se accede al submódulo `mlp`, que contiene las tres proyecciones internas: `gate_proj`, `up_proj`, y `down_proj`.
- **Aplicación del pruning:** Se llama a la función `prune_neuron_pairs` que devuelve versiones más pequeñas de las capas mencionadas, manteniendo las neuronas más importantes.
- **Reemplazo de capas:** Se sobrescriben las capas originales por sus versiones pruned en la instancia del modelo.
- **Actualización del tamaño intermedio:** Una vez se obtiene el nuevo tamaño intermedio (`new_size`) desde la primera capa, se guarda en la configuración del modelo para mantener la consistencia estructural.
- **Resultado:** Se retorna el modelo con todas las capas MLP reducidas en tamaño, manteniendo únicamente las neuronas más relevantes para una inferencia eficiente.

12.2.6. Knowledge Distillation

Este código esta basado en el código del repositorio de github[20], al cual se le han aplicado unos pocos cambios para optimizarlo además de tener que arreglar varias partes porque no compilaba.

Algorithm 8 Entrenamiento por destilación: student model a partir de teacher model

Tokenización: Aplicar tokenización a todo el dataset con truncamiento a 128 tokens, padding y etiquetas iguales a los `input_ids`.
Dataloader: Crear un `DataLoader` con padding dinámico y batches pequeños.
Configurar modelos: Teacher en modo evaluación; student con optimizador `AdamW`.
for cada `epoch` en `num_epochs` **do**
    Poner el modelo student en modo entrenamiento
    Inicializar `total_loss` en cero
    for cada `batch` en `dataloader` **do**
        Mover datos a GPU
        Obtener `teacher_logits` con `model()`
        Aplicar temperatura: `teacher_logits /= temperature`
        Obtener `student_logits` con `student_model()`
        Calcular `KL divergence loss` entre `teacher` y `student`
        Acumular gradientes: `loss.backward()`
        if fin de acumulación de gradientes **then**
            `optimizer.step()`, `optimizer.zero_grad()`
        end if
        Actualizar `total_loss`
    end for
    Imprimir pérdida promedio por epoch
end for

- **Tokenización:** Se define una función para convertir los ejemplos en tensores PyTorch y generando etiquetas iguales a `input_ids` (modelo tipo causal).
- **Preprocesamiento:** Se tokeniza el dataset completo con un `map`, se eliminan las columnas originales y se fuerza el recálculo desactivando la caché.
- **DataLoader:** Se usa `DataCollatorWithPadding` para manejar dinámicamente el padding durante el batch, permitiendo procesamiento eficiente.
- **Configuración de modelos:** El modelo teacher se pone en evaluación; el student tiene un optimizador `AdamW` con un `learning rate` reducido para estabilidad.

- **Bucle de entrenamiento:** Cada epoch inicia con el student en modo entrenamiento y un contador de pérdida inicializado.
- **Procesamiento del batch:** Se mueven tensores a GPU, y el teacher genera logits suavizados con temperatura. Luego, el student infiere y genera su propia salida.
- **Cálculo de pérdida:** Se calcula la `KL Divergence` entre las distribuciones del teacher y del student, usando softmax y log-softmax, escalando por temperatura.
- **Acumulación de gradientes:** Se permite acumular gradientes durante varios pasos (`accumulation_steps`) antes de hacer `optimizer.step()` para simular un batch mayor.
- **Seguimiento de pérdida:** La pérdida escalada se acumula para cada batch.
- **Fin de época:** Se calcula la pérdida promedio para la época y se imprime como métrica de entrenamiento.

12.3. Dataset empleado

Para la realización de los experimentos de este trabajo se ha empleado el conjunto de datos C4 (Colossal Clean Crawled Corpus) en su versión en inglés. Este dataset es un corpus masivo de texto obtenido a partir de páginas web públicas, recopiladas mediante Common Crawl, y posteriormente depurado para eliminar contenido de baja calidad, duplicado o no textual. C4 está diseñado para tareas de modelado de lenguaje y es ampliamente utilizado en la comunidad de investigación en NLP (Natural Language Processing) por su tamaño, variedad y limpieza. Su versión en inglés contiene cientos de gigabytes de texto representando una amplia gama de dominios temáticos, lo que lo convierte en un recurso robusto y generalista para evaluar modelos de lenguaje.

Idealmente, para una evaluación más representativa, lo más adecuado habría sido emplear el conjunto de datos exacto con el que fue entrenado el modelo base utilizado en este trabajo, concretamente LLaMA 3.2–1B. Sin embargo, los datos originales de entrenamiento de los modelos LLaMA no son públicos, ya que Meta no ha hecho disponible la información precisa sobre las fuentes o el corpus completo utilizado. Debido a esta limitación, se ha optado por utilizar C4 como alternativa, al tratarse de un dataset de propósito general, ampliamente aceptado en la comunidad científica, y suficientemente

representativo para llevar a cabo los experimentos de reducción de memoria durante la inferencia sin comprometer la validez de los resultados.

12.4. Problemas durante el desarrollo

Durante el desarrollo de este Trabajo de Fin de Grado surgieron diversas dificultades técnicas y logísticas que condicionaron el planteamiento inicial y obligaron a realizar varios ajustes en el enfoque del proyecto.

En primer lugar, una de las ideas iniciales consistía en emplear DejaVu, una herramienta prometedora por su originalidad y aparente facilidad de uso. Sin embargo, esta biblioteca está diseñada para ser implementada en una estructura de sistema muy específica. Tras varios intentos de adaptarla a una arquitectura más sencilla y compatible con los recursos disponibles, se concluyó que no era viable su integración en el proyecto, debido a las limitaciones técnicas y a la falta de documentación suficiente para su reconfiguración.

Otro obstáculo importante fue la limitación de recursos computacionales. El portátil utilizado durante el desarrollo no contaba con la capacidad necesaria para ejecutar el modelo y realizar pruebas de forma eficiente. Se contempló el uso de un clúster que el tutor me ofreció, pero este se encontraba frecuentemente saturado, lo que hacía inviable esperar a que estuviera disponible. Por ello, fue necesario contratar una suscripción de pago a Google Colab Pro, con el fin de disponer de una infraestructura adecuada para la experimentación.

Además, el modelo finalmente seleccionado requería de una licencia específica para su uso, cuyo proceso de obtención resultó más complejo de lo previsto. Afortunadamente, se consiguió la autorización correspondiente.

En lo que respecta a la ejecución de los experimentos, se encontraron dificultades relacionadas con el tiempo de procesamiento, ya que muchas de las pruebas requerían varias horas para completarse. En varias ocasiones fue necesario repetir experimentos debido a errores en la configuración del entorno o a resultados incoherentes, probablemente causados por fallos en el entorno de ejecución de Google Colab. Estos problemas obligaron a reiniciar el proceso experimental, lo que supuso una pérdida de tiempo.

13. Experimentos

13.1. Set-up experimental

Todos los experimentos se llevarán a cabo en un entorno proporcionado por Google Colab, utilizando una GPU NVIDIA T4. Para las pruebas, se emplearán los primeros 3001 elementos del conjunto de evaluación C4. En los experimentos centrados en la evaluación de métricas de precisión, se establecerá una longitud máxima de entrada de 5000 tokens. Por otro lado, en los experimentos destinados a medir tiempos de inferencia y uso de memoria, se limitará la generación a 50 tokens de salida por muestra.

Además, se ha optado por utilizar el modelo **LLaMA 3.2-1B**, una elección motivada principalmente por dos razones. En primer lugar, al tratarse de un modelo ampliamente difundido, se dispone de abundante documentación y recursos en línea, lo cual facilita la resolución de posibles errores durante el desarrollo. En segundo lugar, al ser un modelo de tamaño reducido, permite ejecutar experimentos y aplicar distintas técnicas de optimización sin requerir hardware especialmente potente, lo que resulta ideal en un entorno con recursos computacionales limitados.

13.2. Cuantización

La cuantización es una técnica para reducir la precisión numérica de los pesos (y a veces activaciones) de un modelo (por ejemplo, de 32-bit float (FP32) a 8-bit entero (INT8) o incluso 4bits) con el fin de:

- Disminuir el tamaño en memoria del modelo.
- Reducir consumo energético, especialmente importante en inference en dispositivos con recursos limitados.
- Reducir la memoria necesaria para ejecutar el modelo.

¿Cómo se cuantiza un tensor?

Supongamos que tenemos un vector de pesos:

$$\mathbf{w} = [w_1, w_2, \ldots, w_n]$$

donde cada w_i es un número en precisión flotante de 32 bits (FP32). Para cuantizar este vector a, por ejemplo, INT8 (enteros de 8 bits), se sigue el siguiente proceso general:

1) Determinar el rango

Se calcula el valor mínimo y máximo del tensor:

$$w_{\min} = \text{mín}(w_1, \ldots, w_n), \quad w_{\max} = \text{máx}(w_1, \ldots, w_n)$$

Esto puede hacerse por fila (en el caso de matrices) o por canal (en convoluciones), según la granularidad deseada.

2) Calcular escala y punto cero (zero-point)

La escala s y el punto cero z se calculan como:

$$s = \frac{w_{\max} - w_{\min}}{2^k - 1}$$

$$z = \text{round}\left(-\frac{w_{\min}}{s}\right)$$

donde k es el número de bits usado para representar cada valor cuantizado (por ejemplo, $k = 8 \Rightarrow 2^8 - 1 = 255$).

3) Cuantizar

Cada elemento w_i se convierte a un valor cuantizado q_i mediante:

$$q_i = \text{clip}\left(\text{round}\left(\frac{w_i}{s}\right) + z,\ 0,\ 2^k - 1\right)$$

donde `clip` limita el valor dentro del rango permitido de $[0, 2^k - 1]$.

Los valores q_i se almacenan como enteros sin signo de k bits (por ejemplo, `uint8` si $k = 8$).

4) De-cuantización (durante inferencia)

Para recuperar una aproximación del valor original en punto flotante, se usa:

$$\hat{w}_i = (q_i - z) \cdot s$$

Este proceso se realiza normalmente "on-the-fly" durante la inferencia, sin reconstruir todo el tensor en FP32.

13.2.1. Experimento

Para llevar a cabo este experimento se utilizará la librería BitsAndBytes [21], que permite aplicar técnicas de cuantización eficiente sobre modelos de lenguaje a gran escala. Además, se utilizará una versión del modelo cargado en float16. Aunque esta representación no constituye una cuantización en sentido estricto, se considerará dentro del análisis por tratarse de una reducción de precisión numérica con fines similares: disminuir el uso de memoria y acelerar la inferencia.

En la Figura 6 se muestran los resultados obtenidos al aplicar distintos niveles de *cuantización* (FP32, FP16, INT8 e INT4) al modelo, incluyendo también la versión base en punto flotante de 32 bits como referencia comparativa. Por su parte, la Figura 7 presenta la correspondiente frontera de Pareto, lo que permite analizar el compromiso entre rendimiento y eficiencia alcanzado en cada caso. Finalmente, en la Tabla 12 se detallan el resto de métricas obtenidas durante el proceso experimental.

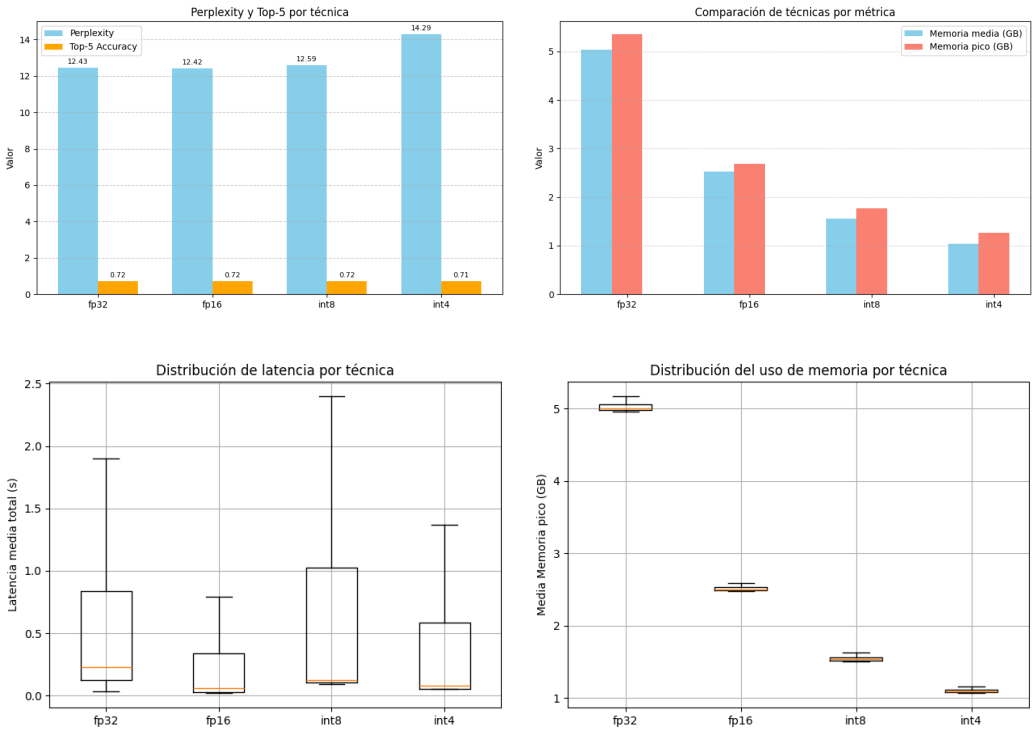

Figura 6: Métricas obtenidas al aplicar cuantización (elaboración propia)

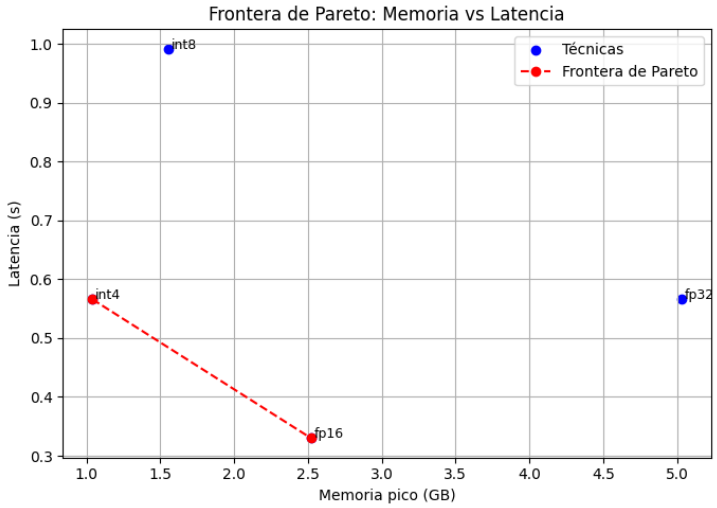

Figura 7: Gráfico de frontera de Pareto de cuantización (elaboración propia)

Cuadro 12: Comparativa de rendimiento entre diferentes niveles de cuantización (elaboración propia)

Métrica	**FP32**	**FP16**	**INT8**	**INT4**
TTFT (media)	0.261 s	0.06 s	0.136 s	0.084 s
TTFT (peak)	1.241 s	0.297 s	0.389 s	0.312 s
TPOT (media)	0.007 s	0.006 s	0.02 s	0.011 s
TPOT (peak)	0.049 s	0.058 s	0.102 s	0.061 s
Latencia (media)	0.573 s	0.330 s	0.992 s	0.566 s
Latencia (peak)	2.905 s	2.712 s	4.662 s	2.898 s
Memoria (media)	5.033 GB	2.521 GB	1.556 GB	1.104 GB
Memoria (peak)	5.357 GB	2.684 GB	1.771 GB	1.269 GB

13.3. Pruning

El *pruning* (poda) es una técnica utilizada en el ámbito del aprendizaje profundo con el objetivo de reducir el tamaño y la complejidad de los modelos, eliminando pesos o estructuras que tienen poca o ninguna influencia en la salida del modelo. Esta estrategia permite disminuir el uso de memoria y el coste computacional durante la inferencia, sin comprometer significativamente el rendimiento del modelo.

Existen principalmente dos tipos de *pruning*: **estructurado** y **no estructurado**.

Pruning no estructurado

En el *pruning* no estructurado se eliminan conexiones individuales (pesos concretos) dentro de las matrices del modelo, basándose generalmente en algún criterio de importancia, como el valor absoluto de los pesos. Es un enfoque muy granular que permite una reducción considerable del número total de parámetros. Sin embargo, este tipo de poda genera matrices dispersas (sparse), lo cual puede dificultar su aprovechamiento en hardware tradicional, ya que muchos aceleradores actuales están optimizados para trabajar con matrices densas.

Pruning estructurado

Por otro lado, el *pruning* estructurado elimina bloques completos del modelo, como pueden ser canales, filtros, cabezas de atención o incluso capas enteras. Aunque este enfoque suele ser menos agresivo en cuanto a la cantidad total de parámetros eliminados, tiene la ventaja de producir modelos más eficientes y fáciles de ejecutar en hardware convencional, ya que mantiene la estructura del modelo más coherente y aprovechable.

Ambos enfoques tienen ventajas e inconvenientes, y la elección entre uno u otro depende del contexto de uso, las restricciones del sistema y los objetivos de compresión o aceleración que se persigan.

Pruning para el experimento

En nuestro caso hemos escogido el pruning estructurado.

Un bloque MLP (*Multilayer Perceptron*) suele estar compuesto por capas que amplían la dimensión del vector de activación y otras que lo devuelven a su tamaño original. Este patrón permite una transformación no lineal más rica de los datos.

En el caso concreto del modelo utilizado, el bloque MLP contiene dos capas de proyección relevantes: `gate_proj` y `up_proj`. Ambas capas realizan una proyección desde 2048 a 8192 dimensiones. Esta duplicación en el escalado podría estar relacionada con mecanismos de compuerta o *gating mechanisms*, los cuales permiten controlar de forma selectiva el flujo de información mediante pesos aprendidos que activan o inhiben ciertas entradas.

Aunque entender completamente el propósito de estas capas requeriría consultar la documentación oficial o incluso el código fuente del modelo, la estructura observada sugiere, como mínimo, que estas capas trabajan en conjunto. En otras palabras, no pueden tratarse como capas lineales independientes.

Esto tiene una implicación clave a la hora de aplicar técnicas como pruning: cualquier operación sobre una de estas capas debe replicarse sobre la otra. Más importante aún, a la hora de identificar qué neuronas son más o menos relevantes, no podemos evaluarlas de forma aislada por capa, sino que deben considerarse como pares asociados.

Este enfoque garantiza que se respete la arquitectura interna del modelo y que el comportamiento aprendido no se degrade al realizar modificaciones estructurales.

Para evaluar la importancia relativa de cada par de neuronas en las capas `gate_proj` y `up_proj`, se aplica una norma basada en los pesos absolutos máximos.

Dado un vector de pesos por neurona, se define la importancia como la suma del valor absoluto del mayor y menor peso en dicha neurona. Matemáticamente:

$$\text{Importancia}_{\text{gate}}(i) = \text{máx}(W_{\text{gate},i}) + |\,\text{mín}(W_{\text{gate},i})| \tag{7}$$

$$\text{Importancia}_{\text{up}}(i) = \text{máx}(W_{\text{up},i}) + |\,\text{mín}(W_{\text{up},i})| \tag{8}$$

La puntuación total de importancia para el par de neuronas se calcula como:

$$\text{Score}(i) = \text{Importancia}_{\text{gate}}(i) + \text{Importancia}_{\text{up}}(i) \tag{9}$$

Esta métrica considera que pesos con gran magnitud (positiva o negativa) implican una mayor contribución funcional, y por tanto, mayor importancia del nodo en el modelo.

13.3.1. Experimento

En la Figura 8 se presentan los resultados obtenidos al aplicar distintos niveles de *pruning* estructurado (20 %, 50 % y 70 %) sobre el modelo. Por su parte, la Figura 9 muestra la correspondiente frontera de Pareto, lo que permite analizar el compromiso entre rendimiento y eficiencia tras cada nivel de reducción. Finalmente, en la Tabla 13 se recoge de forma detallada el resto de métricas relevantes obtenidas durante el proceso experimental. Cabe aclarar que cuando hablamos de un % de pruning este haec referencia únicamente a la capa MLP que es donde se ha realizado el pruning, no al número de parámetros total del modelo.

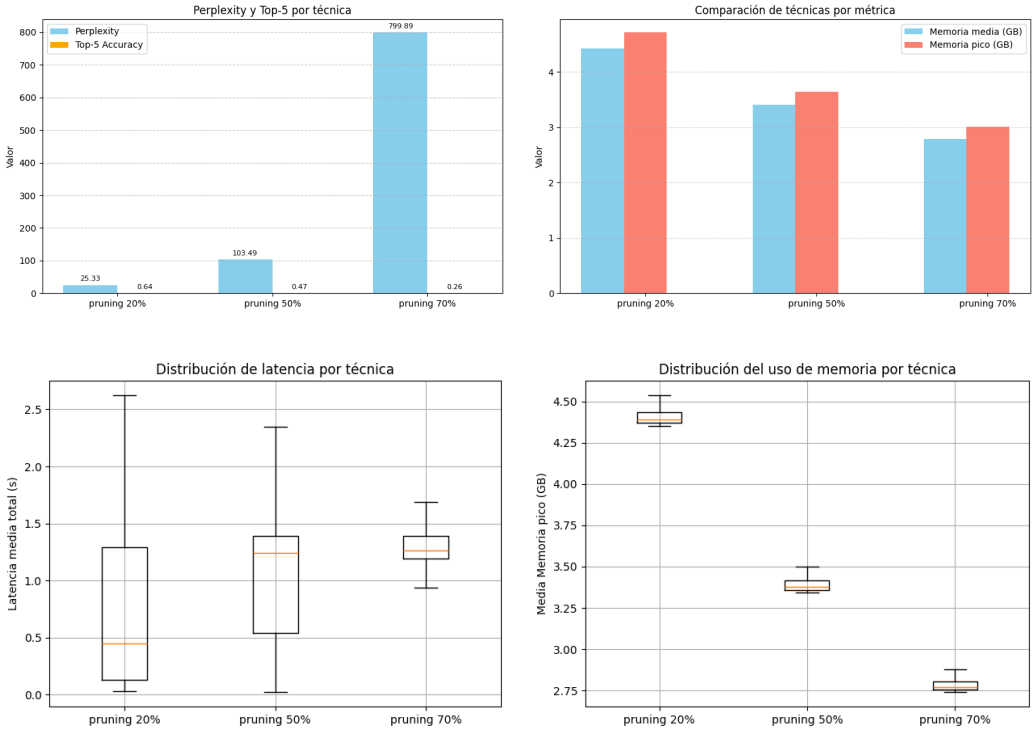

Figura 8: Métricas obtenidas al aplicar pruning (elaboración propia)

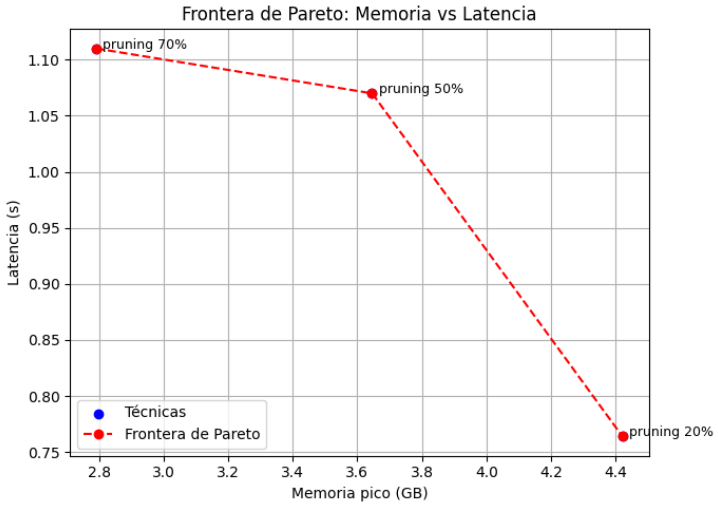

Figura 9: Gráfico de frontera de Pareto para pruning (elaboración propia)

Cuadro 13: Comparativa de rendimiento entre distintos niveles de pruning estructurado (elaboración propia)

Métrica	**Pruning 20 %**	**Pruning 50 %**	**Pruning 70 %**
TTFT (media)	0.226 s	0.166 s	0.126 s
TTFT (peak)	1.050 s	0.803 s	0.635 s
TPOT (media)	0.012 s	0.019 s	0.021 s
TPOT (peak)	0.033 s	0.034 s	0.43 s
Latencia (media)	0.764 s	1.070 s	1.110 s
Latencia (peak)	2.622 s	2.345 s	2.216 s
Memoria (media)	4.421 GB	3.403 GB	2.792 GB
Memoria (peak)	4.719 GB	3.646 GB	3.008 GB

13.3.2. pruning + Knowledge Distillation

Knowledge Distillation (KD) es una técnica de compresión de modelos que permite transferir el conocimiento aprendido por un modelo grande y complejo (conocido como *teacher*) a un modelo más pequeño y eficiente (denominado *student*). Durante este proceso, el modelo estudiante no se entrena únicamente con las etiquetas verdaderas del conjunto de datos, sino que también aprende a imitar las salidas del modelo maestro. Al aprender de estas señales más ricas, el modelo estudiante puede lograr un rendimiento similar al del modelo original, pero con una menor complejidad computacional y menor tamaño.

Debido a la notable pérdida de *accuracy* y *perplexity* observada tras aplicar *pruning* al modelo, se decidió complementar esta técnica con *Knowledge Distillation* con el objetivo de recuperar parte del rendimiento perdido. Esta decisión se basa en la premisa de que no resulta útil reducir el consumo de memoria si ello conlleva un deterioro significativo en la capacidad predictiva del modelo, comprometiendo así su utilidad práctica. No obstante, dado que el objetivo principal del trabajo no es maximizar la precisión del modelo, el proceso de *distillation* se aplicó únicamente de forma parcial, con un entrenamiento limitado, con el fin de evaluar si era posible obtener mejoras sin realizar un ajuste exhaustivo.

Pruning 20 %

A continuación, se analizan las variaciones en las métricas de rendimiento tras la aplicación de *Knowledge Distillation* al modelo previamente reducido mediante un *pruning* estructurado del 20 %. En la Figura 10 se representan gráficamente las principales diferencias observadas antes y después del proceso de destilación. Por su parte, la Tabla 14 recoge de forma detallada el resto de métricas relevantes, permitiendo una evaluación más completa del impacto de esta técnica sobre el comportamiento del modelo.

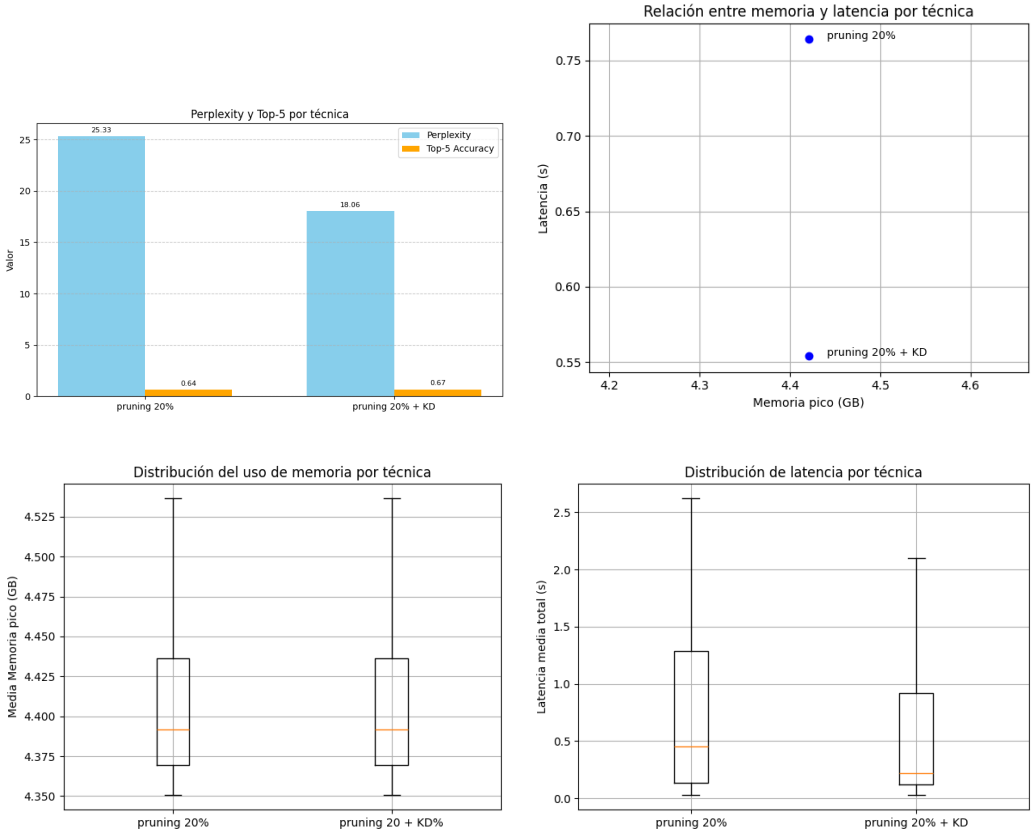

Figura 10: Métricas obtenidas al aplicar KD al modelo con pruning de 20 % (elaboración propia)

Cuadro 14: Comparativa de rendimiento entre el modelo con Pruning 20 % y Pruning 20 % con Knowledge Distillation (elaboración propia)

Métrica	**Pruning 20 %**	**Pruning 20 % + KD**
TTFT (media)	0.226 s	0.243 s
TTFT (peak)	1.050 s	1.141 s
TPOT (media)	0.012 s	0.007 s
TPOT (peak)	0.033 s	0.037 s
Latencia (media)	0.764 s	0.554 s
Latencia (peak)	2.622 s	1.141 s
Memoria (media)	4.421 GB	4.421 GB
Memoria (peak)	4.719 GB	4.715 GB

Pruning 50 %

A continuación, se examinan los cambios en las métricas de rendimiento tras aplicar el proceso de Knowledge Distillation al modelo previamente reducido mediante un pruning estructurado del 50 %. La Figura 11 ilustra visualmente las principales diferencias observadas antes y después de la destilación. Asimismo, en la Tabla 16 se detalla el resto de métricas relevantes, lo que permite realizar una evaluación más exhaustiva del efecto de esta técnica en el desempeño del modelo.

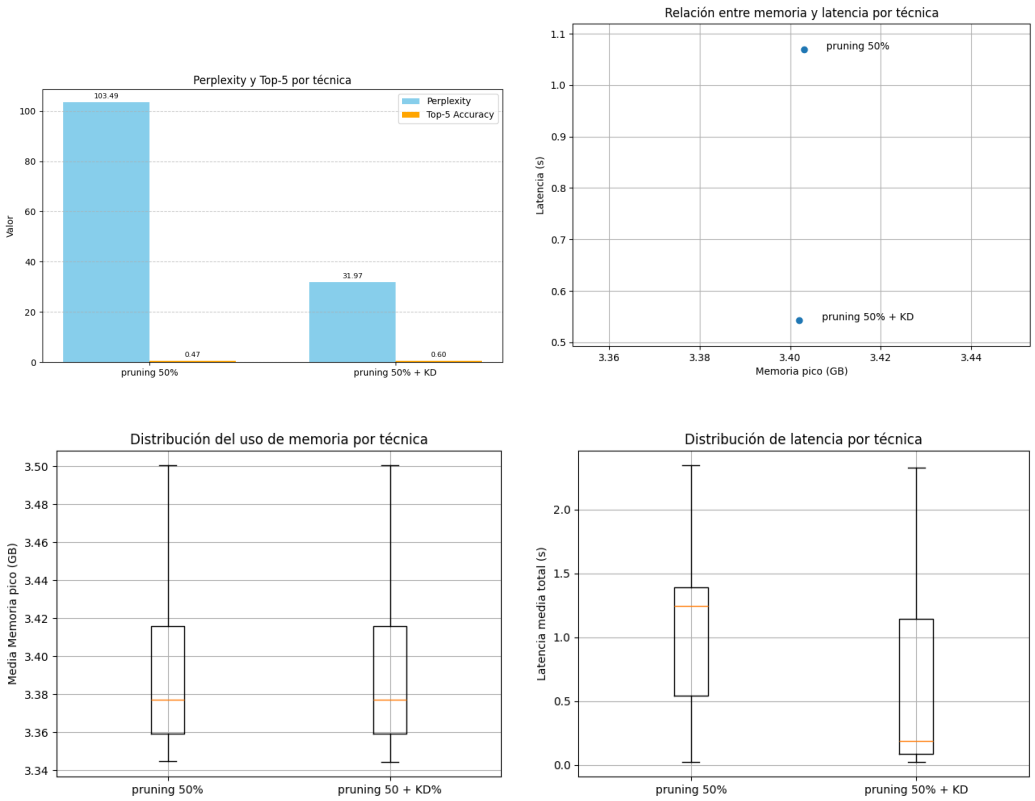

Figura 11: Métricas obtenidas al aplicar KD al modelo con pruning de 50 % (elaboración propia)

Cuadro 15: Comparativa de rendimiento entre el modelo con Pruning 50 % y Pruning 50 % con Knowledge Distillation (elaboración propia)

Métrica	**Pruning 50 %**	**Pruning 50 % + KD**
TTFT (media)	0.166 s	0.163 s
TTFT (peak)	0.803 s	0.804 s
TPOT (media)	0.019 s	0.009 s
TPOT (peak)	0.034 s	0.035 s
Latencia (media)	1.070 s	0.543 s
Latencia (peak)	2.345 s	2.325 s
Memoria (media)	3.403 GB	3.402 GB
Memoria (peak)	3.646 GB	3.646 GB

Pruning 70 %

A continuación se analizan las variaciones en las métricas de rendimiento tras aplicar Knowledge Distillation al modelo previamente reducido mediante un pruning estructurado del 70 %. La Figura 12 muestra de forma gráfica las principales diferencias antes y después de la destilación, mientras que la Tabla 15 recoge el resto de métricas clave para llevar a cabo una comparación detallada del impacto de esta técnica en el comportamiento del modelo.

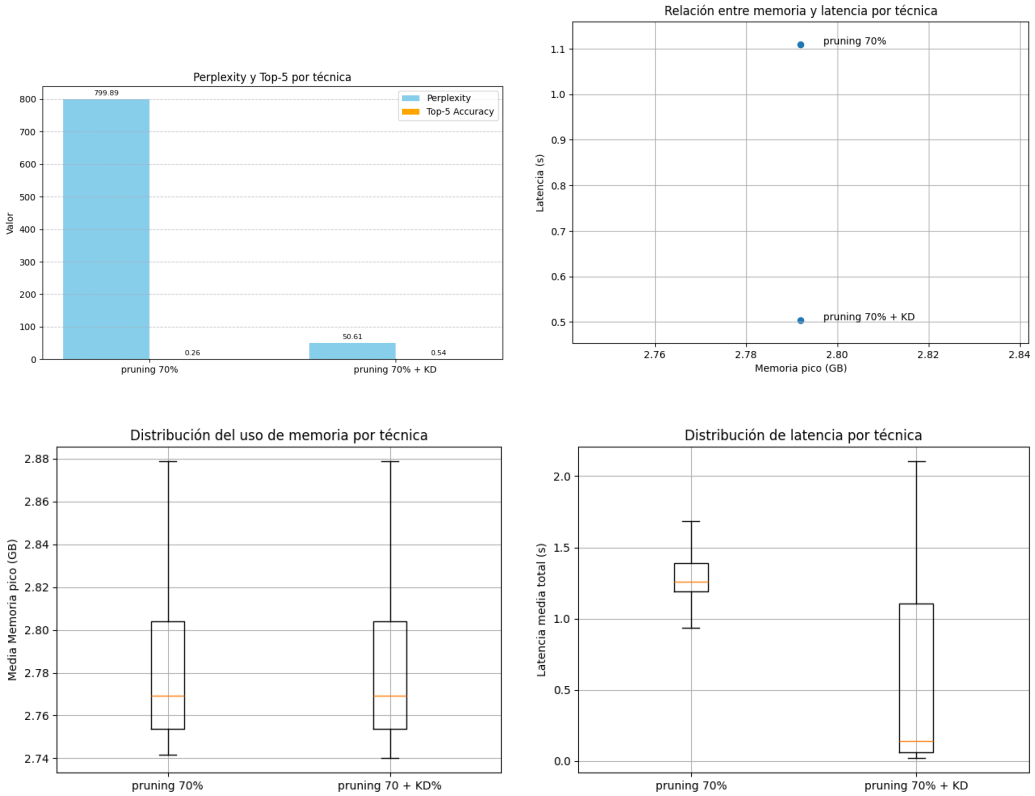

Figura 12: Métricas obtenidas al aplicar KD al modelo con pruning de 70 % (elaboración propia)

Cuadro 16: Comparativa de rendimiento entre el modelo con Pruning 70 % y Pruning 70 % con Knowledge Distillation (elaboración propia)

Métrica	**Pruning 70 %**	**Pruning 70 % + KD**
TTFT (media)	0.126 s	0.125 s
TTFT (peak)	0.635 s	0.624 s
TPOT (media)	0.021 s	0.008 s
TPOT (peak)	0.043 s	0.034 s
Latencia (media)	1.110 s	0.504 s
Latencia (peak)	2.216 s	2.104 s
Memoria (media)	2.792 GB	2.792 GB
Memoria (peak)	3.008 GB	3.008 GB

13.4. Pruning + Knowledge Distillation + Cuantización

13.4.1. Pruning 20 % + KD+ Cuantización

En la siguiente sección se examinan las variaciones en las métricas de rendimiento tras aplicar un pipeline de optimización compuesto por un pruning estructurado al 20 %, seguido de Knowledge Distillation y posterior cuantización. La Figura 13 ilustra gráficamente cómo evolucionan las principales métricas antes y después de estas transformaciones, incluyendo también el comportamiento del modelo FP32 "original" como referencia. Por su parte, la Tabla 17 presenta en detalle el resto de indicadores clave, permitiendo así una comparación exhaustiva entre la versión completa y las variantes optimizadas.

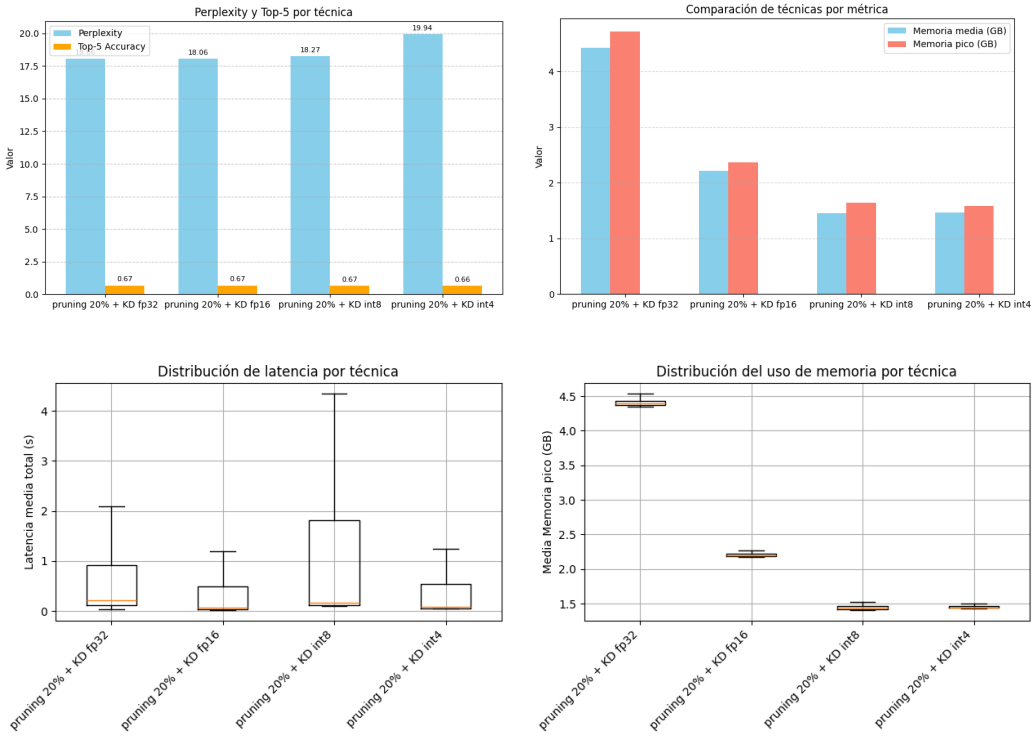

Figura 13: Métricas obtenidas al aplicar quantización al modelo con pruning de 20 % + KD (elaboración propia)

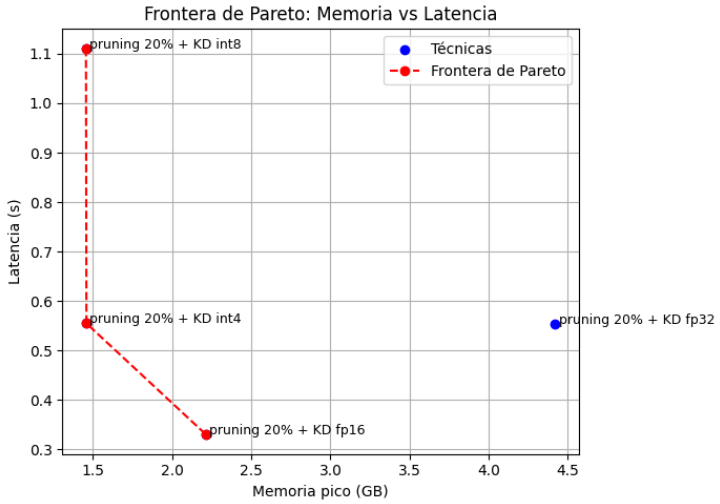

Figura 14: Gráfico de frontera de Pareto para pruning 20 % + KD cuantizado (elaboración propia)

Cuadro 17: Comparativa de rendimiento para Pruning 20 % + Knowledge Distillation en configuraciones FP32, FP16, INT8 e INT4 (elaboración propia)

Métrica	**FP32**	**FP16**	**INT8**	**INT4**
TTFT (media)	0.243 s	0.061 s	0.177 s	0.082 s
TTFT (peak)	1.141 s	0.291 s	0.604 s	0.289 s
TPOT (media)	0.007 s	0.006 s	0.021 s	0.011 s
TPOT (peak)	0.037 s	0.042 s	0.120 s	0.061 s
Latencia (media)	0.554 s	0.330 s	1.111 s	0.555 s
Latencia (peak)	2.722 s	1.710 s	6.046 s	2.780 s
Memoria (media)	4.421 GB	2.215 GB	1.456 GB	1.459 GB
Memoria (peak)	4.715 GB	2.363 GB	1.640 GB	1.583 GB

13.4.2. Pruning 50 % + KD+ Cuantización

En esta sección se analizan los cambios en el rendimiento del modelo tras aplicar una cadena de optimización compuesta por un *pruning* estructurado del 50 %, seguido de un proceso de *Knowledge Distillation* y su posterior cuantización a diferentes precisiones. La Figura 15 muestra de forma visual la evolución de las métricas más relevantes en cada etapa del pipeline, tomando

como punto de partida el comportamiento del modelo base en formato FP32. Adicionalmente, la Tabla 18 recopila los resultados numéricos detallados, lo que permite contrastar de forma precisa el impacto de cada técnica de optimización sobre el rendimiento global del modelo.

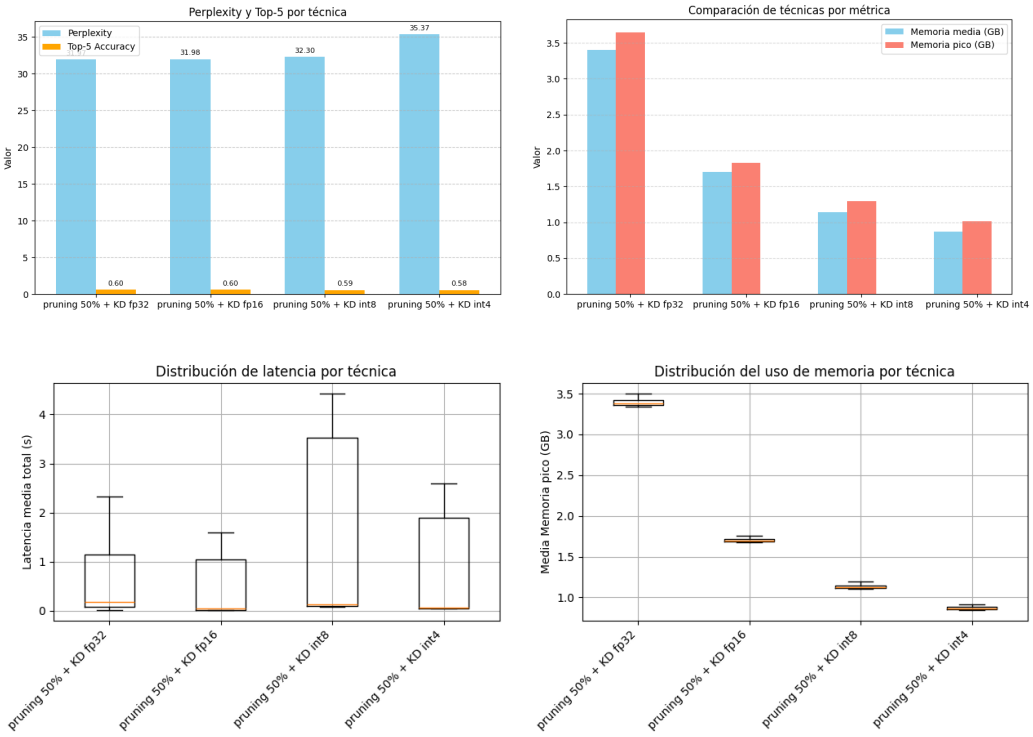

Figura 15: Métricas obtenidas al aplicar quantización al modelo con pruning de 50 % + KD (elaboración propia)

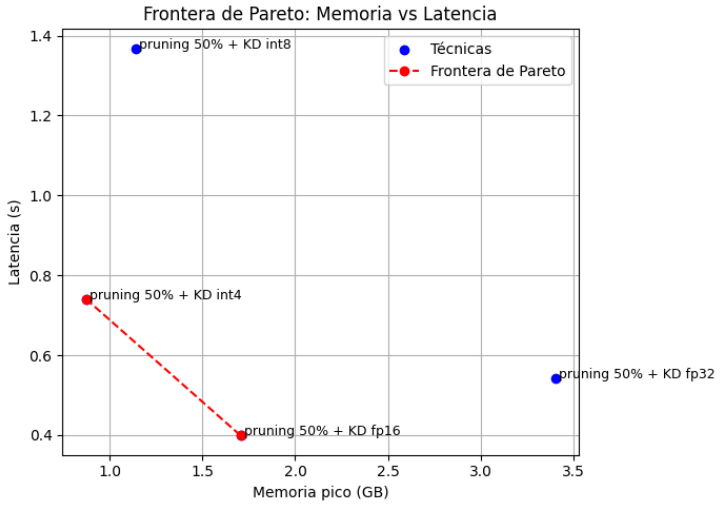

Figura 16: Gráfico de frontera de Pareto para pruning 50 % + KD cuantizado (elaboración propia)

Cuadro 18: Comparativa de rendimiento para Pruning 50 % + Knowledge Distillation en configuraciones FP32, FP16, INT8 e INT4 (elaboración propia)

Métrica	**FP32**	**FP16**	**INT8**	**INT4**
TTFT (media)	0.163 s	0.042 s	0.126 s	0.067 s
TTFT (peak)	0.804 s	0.212 s	0.314 s	0.195 s
TPOT (media)	0.009 s	0.008 s	0.028 s	0.015 s
TPOT (peak)	0.035 s	0.032 s	0.096 s	0.064 s
Latencia (media)	0.543 s	0.399 s	1.368 s	0.741 s
Latencia (peak)	2.325 s	1.594 s	4.421 s	2.597 s
Memoria (media)	3.402 GB	1.706 GB	1.141 GB	0.873 GB
Memoria (peak)	3.646 GB	1.828 GB	1.292 GB	1.015 GB

13.4.3. Pruning 70 % + KD+ Cuantización

Esta sección está dedicada al análisis del rendimiento del modelo tras aplicar una estrategia de optimización que incluye un *pruning* estructurado del 70 %, seguido de *Knowledge Distillation* y cuantización en diferentes formatos numéricos. En la Figura 17 se representan gráficamente las principales métricas obtenidas a lo largo del proceso, incluyendo como referencia el ren-

dimiento del modelo original sin optimizar (FP32). Por otro lado, la Tabla 19 ofrece un desglose cuantitativo de los indicadores más relevantes, facilitando así una evaluación comparativa precisa entre las distintas versiones del modelo.

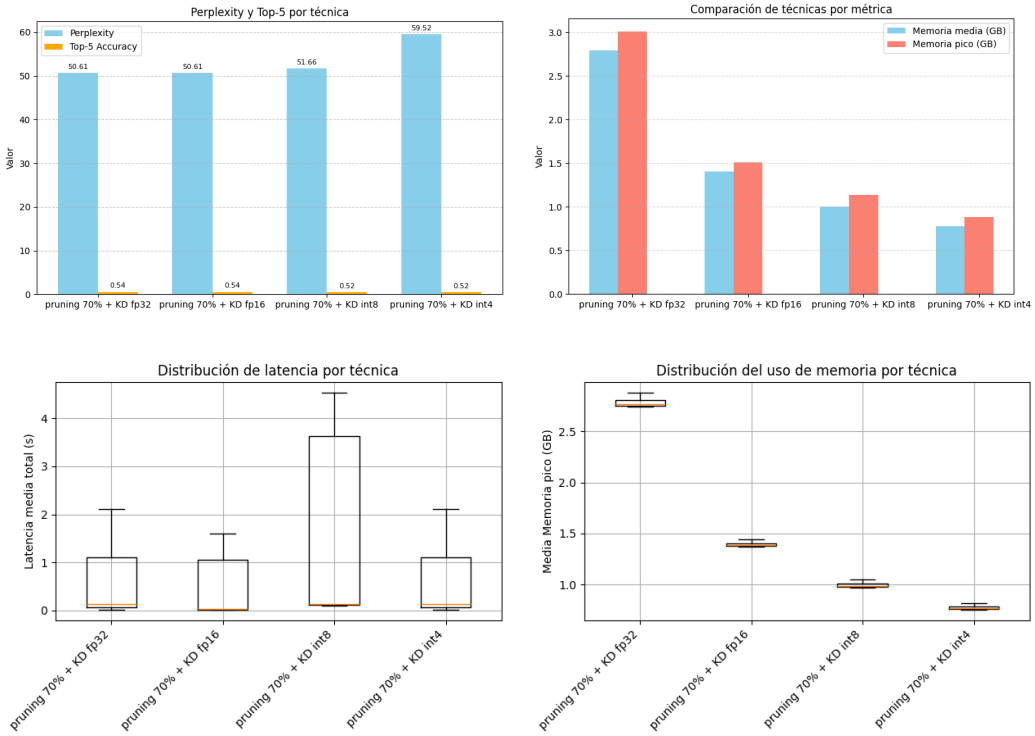

Figura 17: Métricas obtenidas al aplicar quantización al modelo con pruning de 70 % + KD (elaboración propia)

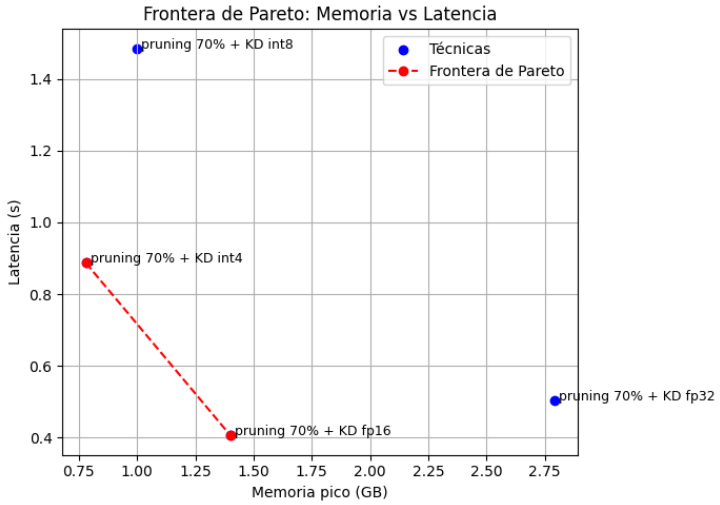

Figura 18: Gráfico de frontera de Pareto para pruning 50 % + KD cuantizado (elaboración propia)

Cuadro 19: Comparativa de rendimiento para Pruning 70 % + Knowledge Distillation en configuraciones FP32, FP16, INT8 e INT4 (elaboración propia)

Métrica	**FP32**	**FP16**	**INT8**	**INT4**
TTFT (media)	0.125 s	0.039 s	0.141 s	0.066 s
TTFT (peak)	0.624 s	0.169 s	0.404 s	0.249 s
TPOT (media)	0.008 s	0.008 s	0.029 s	0.018 s
TPOT (peak)	0.034 s	0.032 s	0.100 s	0.070 s
Latencia (media)	0.504 s	0.406 s	1.485 s	0.887 s
Latencia (peak)	2.104 s	1.608 s	4.530 s	3.469 s
Memoria (media)	2.792 GB	1.401 GB	0.999 GB	0.780 GB
Memoria (peak)	3.008 GB	1.509 GB	1.138 GB	0.880 GB

14. Análisis de los Resultados

14.1. Experimentos

14.1.1. Experimento 1: Cuantización

A partir de los datos y las gráficas obtenidas, extraemos las siguientes conclusiones clave sobre el compromiso memoria–latencia–calidad de cada técnica de inferencia:

1. **Frontera de Pareto**

$$\{\texttt{FP16},\ \texttt{INT4}\}$$

   - `FP32` está dominada por `INT4` (más memoria y latencia casi idéntica).
   - `INT8` está dominada por `INT4` (peor latencia y menos reducción de memoria).

2. **Calidad del modelo** (perplejidad y Top-5 Accuracy)

Técnica	Perplejidad	Top-5
`FP32`	12,43	0,7214
`FP16`	12,42	0,7217
`INT8`	12,59	0,720
`INT4`	14,29	0,7055

   - `FP16` iguala prácticamente la perplejidad de `FP32` y mejora ligeramente el Top-5.
   - `INT8` presenta un pequeño aumento de perplejidad manteniendo Top-5 casi constante.
   - `INT4` muestra la mayor degradación.

   Por tanto, podemos observar que en este caso la cuantización apenas afecta la precisión del modelo, probablemente porque se trata de un modelo de tamaño reducido, donde el impacto de reducir la representación numérica es menor que en arquitecturas más grandes. Únicamente al aplicar INT4 se aprecia una degradación significativa, lo cual resulta coherente dado el drástico recorte en la fidelidad de los valores almacenados.

3. **Latencia**

- `FP16` es la más rápida y con baja dispersión.
- `FP32` e `INT4` tienen latencias similares, pero `FP32` consume 5× más memoria.
- `INT8` es la más lenta y con gran variabilidad, por lo que no aporta ventaja en rapidez.

4. **Uso de memoria**

   - `FP16` reduce el uso a la mitad respecto a `FP32`.
   - `INT8` e `INT4` bajan el pico a 1.77GB y 1.26GB respectivamente.
   - `INT4` es la que mas memoria reduce dejandolo en 1.269GB el pico.

5. **Conclusiones Finales**

   - `FP16`: mejor equilibrio general menos memoria, menos latencia, y con muy poca o casi ninguna pérdida de calidad frente a `FP32`).
   - `INT4`: solo en memoria extremadamente limitada, asumiendo gran degradación de calidad.
   - `INT8`: no competitivo para este escenario (fuera de Pareto).

Los resultados muestran que la aplicación de cuantización no ha comprometido en ningún momento la precisión del modelo, lo cual respalda su uso como una técnica fiable dentro de pipelines de optimización. Además, el impacto sobre el rendimiento computacional ha sido significativo, consiguiéndose reducciones notables en el uso de memoria y en ciertos casos, también en la latencia.

Respecto al comportamiento observado en la variante INT8, su menor eficiencia en términos de latencia podría estar relacionada con la arquitectura específica de la GPU utilizada, ya que no todos los dispositivos cuentan con soporte optimizado para operaciones en este formato. Por otro lado, el menor tamaño de representación de INT4 permite aprovechar mejor el paralelismo de hardware, aún y no estar optimizado para ello, lo que explicaría su mejor desempeño en algunos escenarios. En conjunto, estos resultados subrayan la importancia de considerar tanto las características del modelo como las del entorno de ejecución al seleccionar una técnica de cuantización.

14.1.2. Experimento 2: Pruning

A partir de los datos y las gráficas obtenidas para las tres variantes de pruning (20 %, 50 % y 70 %), extraemos las siguientes conclusiones clave sobre el compromiso memoria–latencia–calidad de cada técnica:

1. **Frontera de Pareto**

$$\{\ 20\,\%,\ 50\,\%,\ 70\,\%\}$$

   - Ninguna de las tres configuraciones domina en ambos ejes (*memoria pico* y *latencia*).
   - Pruning 20 % es la más rápida, pero utiliza más memoria; Pruning 70 % minimiza la memoria a costa de una mayor latencia; Pruning 50 % ofrece un punto intermedio.

2. **Calidad del modelo** (Perplejidad y Top-5 Accuracy)

Técnica	Perplejidad	Top-5
Pruning 20 %	25,33	0,64
Pruning 50 %	103,49	0,47
Pruning 70 %	799,89	0,26

   - La perplejidad y la Top-5 se degradan progresivamente al aumentar el nivel de pruning.
   - Hasta el 50 % la calidad se mantiene en niveles razonables; al 70 % aparece una degradación drástica.

3. **Latencia**

   - *Pruning 20 %*: $\overline{L} = 0{,}764\,\text{s}$, pico 2.622 s — la más rápida en media, pero muy dispersa.
   - *Pruning 50 %*: $\overline{L} = 1{,}070\,\text{s}$, pico 2.345 s — latencia intermedia y dispersión moderada.
   - *Pruning 70 %*: $\overline{L} = 1{,}110\,\text{s}$, pico 2.216 s — la más lenta en media, pero con menor variabilidad en el pico.

4. **Uso de memoria**

   - *Pruning 20 %*: $\overline{M} = 4{,}421\,\text{GB}$, pico 4.719 GB — mayor consumo pero también mayor velocidad.

- *Pruning 50 %*: $\overline{M} = 3{,}403\,\text{GB}$, pico 3.646 GB — reducción significativa con latencia moderada.
- *Pruning 70 %*: $\overline{M} = 2{,}792\,\text{GB}$, pico 3.008 GB — mínima memoria, aunque a costa de mayor latencia y pérdida de calidad total.

5. **Conclusiones Finales**

   - **Pruning 20 %** resulta óptimo si se busca *baja latencia* y el sistema dispone de memoria suficiente, manteniendo buena calidad.
   - **Pruning 50 %** constituye un compromiso equilibrado entre reducción de memoria y latencia, con una degradación controlada de la calidad.
   - **Pruning 70 %** es adecuado únicamente en entornos con memoria muy limitada, aceptando un rendimiento más lento y una pérdida de precisión notable.

La pérdida de calidad observada tras aplicar pruning es esperable, dado que esta técnica elimina directamente neuronas o canales del modelo, reduciendo su capacidad de representación. Asimismo, la alteración de la arquitectura interna puede ocasionar un incremento de la latencia: por ejemplo, al modificar el tamaño y la forma de las capas, algunas operaciones dejan de aprovechar eficientemente los kernels optimizados del hardware.

14.1.3. Experimento 3: Pruning + Knowledge Distillation

A continuación se describen los resultados comparativos entre el modelo con pruning al 20 %, 50 % , 70 % y su versión optimizada añadiendo Knowledge Distillation (KD).

Pruning 20 %+ KD

1. **Calidad del modelo**

Técnica	Perplejidad	Top-5
Pruning 20 %	25,33	0,64
Pruning 20 % + KD	18,06	0,67

   - KD reduce la perplejidad de 25.33 a 18.06 y mejora Top-5 Accuracy de 0.64 a 0.67.

- Esto indica que el estudiante recupera información perdida tras el pruning.

2. **Latencia**
   - *Pruning 20 %*: media 0.764 s, pico 2.622 s.
   - *Pruning 20 % + KD*: media 0.554 s, pico 1.141 s.
   - KD reduce la latencia media un poco y la latencia pico bastante.
3. **Uso de memoria**
   - *Pruning 20 %*: media 4.421 GB, pico 4.719 GB.
   - *Pruning 20 % + KD*: media 4.421 GB, pico 4.715 GB.
   - Prácticamente no hay incremento de memoria al añadir KD.
4. **Conclusiones Finales**
   - Knowledge Distillation tras pruning al 20 % recupera gran parte de la calidad (perplejidad y Top-5) y reduce significativamente la latencia.
   - Gran beneficio den latecia sin coste en memoria.

Pruning 50 % + KD

1. **Calidad del modelo**

Técnica	Perplejidad	Top-5
Pruning 50 %	103,49	0,47
Pruning 50 % + KD	31,97	0,60

   - KD reduce la perplejidad de 103.49 a 31.97, recuperando gran parte de la calidad perdida tras el pruning agresivo.
   - Además, mejora la Top-5 Accuracy de 0.47 a 0.60, lo que indica una mejor capacidad del estudiante para predecir las opciones más probables.
2. **Latencia**
   - *Pruning 50 %*: media 1.070 s, pico 2.345 s.
   - *Pruning 50 % + KD*: media 0.543 s, pico 2.325 s.

- KD reduce la latencia media casi a la mitad, manteniendo prácticamente igual la latencia pico.

3. **Uso de memoria**

   - *Pruning 50 %*: media 3.403 GB, pico 3.646 GB.
   - *Pruning 50 % + KD*: media 3.402 GB, pico 3.646 GB.
   - No hay apenas variación en el consumo de memoria al aplicar KD tras el pruning.

4. **Conclusiones Finales**

   - Aplicar Knowledge Distillation tras pruning al 50 % permite recuperar buena parte de la calidad del modelo (perplejidad y Top-5) que se pierde con un pruning tan agresivo.
   - Se consigue además una reducción drástica de la latencia media sin coste adicional en memoria, lo que lo hace especialmente interesante para despliegues en entornos con restricciones de rendimiento.

Pruning 70 % + KD

1. **Calidad del modelo**

Técnica	Perplejidad	Top-5
Pruning 70 %	799,89	0,26
Pruning 70 % + KD	50,61	0,54

   - KD reduce la perplejidad de 799.89 a 50.61, recuperando gran parte de la calidad perdida tras un pruning tan agresivo, pero dejando el modelo aún bastante mal.
   - Mejora la Top-5 Accuracy de 0.26 a 0.54, lo que demuestra que el estudiante aprende a identificar las respuestas más probables.

2. **Latencia**

   - *Pruning 70 %*: media 1.110 s, pico 2.216 s.
   - *Pruning 70 % + KD*: media 0.504 s, pico 2.104 s.
   - Con KD la latencia media se reduce a menos de la mitad, mientras que la latencia pico sólo disminuye ligeramente.

3. **Uso de memoria**

   - *Pruning 70 %*: media 2.792 GB, pico 3.008 GB.
   - *Pruning 70 % + KD*: media 2.792 GB, pico 3.008 GB.
   - Prácticamente no hay variación en el consumo de memoria al aplicar KD.

4. **Conclusiones Finales**

   - La Knowledge Distillation tras un pruning del 70 % permite recuperar de forma notable la calidad del modelo (perplejidad y Top-5) que se pierde con un pruning tan elevado.
   - Además, reduce drásticamente la latencia media sin ningún coste adicional en memoria, lo que lo hace muy adecuado para entornos con recursos limitados o aplicaciones de baja latencia.

Conclusiones finales para Pruning KD

Podemos observar claramente que el modelo ha recuperado una parte significativa de su precisión gracias al proceso de *Knowledge Distillation* (KD), aunque aún queda margen de mejora. Es importante destacar que el entrenamiento utilizado para esta recuperación no ha sido óptimo, ya que el objetivo principal de este Trabajo de Fin de Grado no era maximizar la precisión del modelo, sino explorar la reducción del uso de memoria y su equilibrio con la latencia. Por este motivo, solo se ha realizado un reentrenamiento parcial con el fin de evaluar hasta qué punto podía recuperarse la precisión. Es razonable pensar que, con un entrenamiento más exhaustivo y ajustado, el modelo podría recuperar aún más precisión.

También es importante señalar que es normal que el uso de memoria no haya variado. Esto se debe a que el proceso de KD no modifica la estructura o arquitectura del modelo, sino únicamente sus pesos. Por tanto, la cantidad de memoria requerida sigue siendo la misma. En cuanto a la latencia, es esperable que esta disminuya, ya que el modelo destilado tiende a realizar inferencias de forma más eficiente al haber aprendido una representación más condensada del conocimiento del modelo maestro, lo cual puede traducirse en una ejecución más rápida, especialmente si se han eliminado redundancias o complejidades innecesarias en los patrones de activación.

Hemos comprobado que, en todos los casos, la versión del modelo optimizada mediante *Knowledge Distillation* (KD) ofrece un mejor rendimiento general que la versión original sin destilación. A partir de este punto, nos

centramos exclusivamente en comparar entre sí las distintas versiones del modelo que ya incorporan KD, evaluando el impacto de las distintas técnicas de cuantización aplicadas. En esta comparación se incluyen las versiones cuantizadas en `fp16`, `int8` e `int4`, así como la versión sin cuantizar en `fp32`, que sirve como referencia base para analizar las diferencias en precisión, uso de memoria y latencia.

14.1.4. Experimento 4: Pruning+ KD + Cuantización

A continuación se presentan los resultados comparativos entre los modelos con *pruning* aplicado en distintos niveles (20 %, 50 % y 70 %) y sus respectivas versiones optimizadas mediante *Knowledge Distillation* (KD). Para cada porcentaje de *pruning*, se realiza una comparación considerando también las distintas técnicas de cuantización aplicadas: `fp32`, `fp16`, `int8` e `int4`. Esta comparación permite analizar el impacto combinado del *pruning*, la destilación y la cuantización sobre el rendimiento, la precisión, el uso de memoria y la latencia del modelo.

Pruning 20 %+ KD cuantizado

1. **Calidad del modelo**

Técnica	Perplejidad	Top-5
Pruning 20 % + KD FP32	18,06	0,67
Pruning 20 % + KD FP16	18,06	0,67
Pruning 20 % + KD INT8	18,27	0,67
Pruning 20 % + KD INT4	19,94	0,66

   - FP16 mantiene idéntica calidad a FP32 (perplejidad 18.06, Top-5 0.67).
   - INT8 apenas introduce degradación (perplejidad 18.27, Top-5 0.67).
   - INT4 presenta un ligero empeoramiento (perplejidad 19.94, Top-5 0.66), aun así aceptable.

2. **Latencia**

   - *FP32*: media 0.554 s, pico 2.722 s.
   - *FP16*: media 0.330 s, pico 1.710 s.
   - *INT8*: media 1.111 s, pico 6.046 s.
   - *INT4*: media 0.555 s, pico 2.780 s.

- FP16 reduce notablemente latencia media y pico respecto a FP32.
- INT4 ofrece latencia media similar a FP32 pero con menor pico.
- INT8, pese a bajo consumo, su pico es muy alto por descompresión/compresión.

3. **Uso de memoria**
   - *FP32*: media 4.421 GB, pico 4.715 GB.
   - *FP16*: media 2.215 GB, pico 2.363 GB.
   - *INT8*: media 1.456 GB, pico 1.640 GB.
   - *INT4*: media 1.459 GB, pico 1.583 GB.
   - FP16, INT8 e INT4 reducen drásticamente memoria media y pico en comparación con FP32.
4. **Frontera de Pareto (Memoria vs Latencia)**
   - Técnicas en la frontera de Pareto:
     - *Pruning 20 % + KD INT8*: $\left(1{,}64\,\text{GB},\ 0{,}18\,\text{s}\right)$
     - *Pruning 20 % + KD INT4*: $\left(1{,}58\,\text{GB},\ 0{,}15\,\text{s}\right)$
     - *Pruning 20 % + KD FP16*: $\left(2{,}36\,\text{GB},\ 0{,}33\,\text{s}\right)$
   - FP32 no es Pareto-óptimo frente a FP16, INT8 e INT4.
   - La elección depende del trade-off deseado entre menor memoria (INT8/INT4) y menor latencia media (FP16).
5. **Conclusiones Finales**
   - La cuantización tras pruning 20 % + KD permite ajustar finamente el balance calidad-rendimiento: FP16 ofrece la mejor latencia media con calidad intacta. INT8/INT4 maximizan la reducción de memoria, con un ligero coste de calidad o latencia pico.
   - Dependiendo de la aplicación (latencia estricta vs memoria limitada), puede seleccionarse la técnica Pareto-óptima correspondiente.

Al aplicar cuantización a 4 bits (`int4`), hemos observado que la ganancia en reducción de espacio de memoria ha sido menor de lo esperado. Esto se debe a que la estructura resultante de las capas del modelo no es múltiplo de 64 elementos, lo cual impide un empaquetado óptimo de los valores de 4 bits. En la práctica, los sistemas de vectores y registros de 64 bits requieren que los datos se alineen en bloques de ese tamaño; al no cumplirse dicha

condición, es necesario añadir relleno (*padding*) para completar cada bloque, lo que anula parte de la reducción teórica de memoria. Por tanto, aunque la cuantización `int4` reduce el ancho de cada parámetro, la falta de alineación en bloques de 64 ocasiona un uso de espacio subóptimo en comparación con otras precisiones que sí se ajustan a este requisito.

Pruning 50 %+ KD cuantizado

1. **Calidad del modelo**

Técnica	Perplejidad	Top-5
Pruning 50 % + KD FP32	31,97	0,60
Pruning 50 % + KD FP16	31,98	0,60
Pruning 50 % + KD INT8	32,30	0,59
Pruning 50 % + KD INT4	35,37	0,58

   - FP16 mantiene prácticamente la misma perplejidad y Top-5 que FP32 (31.98 vs. 31.97 y 0.60).
   - INT8 introduce un ligero deterioro (perplejidad 32.30, Top-5 0.59).
   - INT4 penaliza algo más la calidad (perplejidad 35.37, Top-5 0.58) pero sigue siendo aceptable.

2. **Latencia**

   - *FP32*: media 0.543 s, pico 2.325 s.
   - *FP16*: media 0.399 s, pico 1.594 s.
   - *INT8*: media 1.368 s, pico 4.421 s.
   - *INT4*: media 0.741 s, pico 2.597 s.
   - FP16 reduce tanto latencia media como pico respecto a FP32.
   - INT4 ofrece latencia media intermedia y pico moderado.
   - INT8, aunque muy compacto, su latencia pico es elevada.

3. **Uso de memoria**

   - *FP32*: media 3.402 GB, pico 3.646 GB.
   - *FP16*: media 1.706 GB, pico 1.828 GB.
   - *INT8*: media 1.141 GB, pico 1.292 GB.
   - *INT4*: media 0.873 GB, pico 1.015 GB.

- Todos los esquemas cuantizados reducen drásticamente memoria media y pico respecto a FP32, siendo INT4 el más ligero.

4. **Frontera de Pareto (Memoria vs Latencia)**
   - Técnicas en la frontera de Pareto (memoria pico vs latencia media):
     - *Pruning 50 % + KD INT4*: $\left(1{,}015\,\text{GB},\ 0{,}741\,\text{s}\right)$
     - *Pruning 50 % + KD FP16*: $\left(1{,}828\,\text{GB},\ 0{,}399\,\text{s}\right)$
   - FP32 e INT8 quedan fuera de la frontera al ser dominadas por combinaciones con menor memoria y/o menor latencia.
   - La elección final depende del trade-off: *INT4* para minimizar memoria con latencia media moderada. *FP16* para obtener la latencia media más baja con memoria reducida.

5. **Conclusiones Finales**
   - La cuantización tras Pruning 50 % + KD permite ajustar finamente calidad y recursos: FP16 retiene calidad idéntica con notable reducción de latencia y memoria, mientras que INT4 maximiza el ahorro de memoria a costa de menor calidad y latencia intermedia.
   - Cada técnica Pareto-óptima responde a diferentes requisitos de despliegue: baja latencia (FP16) vs mínimo consumo de memoria (INT4).

Pruning 70 %+ KD cuantizado

1. **Calidad del modelo**

Técnica	Perplejidad	Top-5
Pruning 70 % + KD FP32	50,61	0,54
Pruning 70 % + KD FP16	50,61	0,54
Pruning 70 % + KD INT8	51,66	0,52
Pruning 70 % + KD INT4	59,52	0,52

   - FP16 mantiene idéntica perplejidad y Top-5 que FP32 (50.61, 0.54).
   - INT8 introduce un leve empeoramiento (perplejidad 51.66, Top-5 0.52).
   - INT4 penaliza más la calidad (perplejidad 59.52, Top-5 0.52), aunque sigue en rangos razonables.

2. **Latencia**
    - *FP32*: media 0.504 s, pico 2.104 s.
    - *FP16*: media 0.406 s, pico 1.608 s.
    - *INT8*: media 1.485 s, pico 4.530 s.
    - *INT4*: media 0.887 s, pico 3.469 s.
    - FP16 reduce tanto latencia media como pico respecto a FP32.
    - INT4 ofrece latencia media intermedia y pico moderado.
    - INT8, pese a ser muy compacto, su pico es muy elevado.
3. **Uso de memoria**
    - *FP32*: media 2.792 GB, pico 3.008 GB.
    - *FP16*: media 1.401 GB, pico 1.509 GB.
    - *INT8*: media 0.999 GB, pico 1.138 GB.
    - *INT4*: media 0.780 GB, pico 0.880 GB.
    - Todos los esquemas cuantizados reducen drásticamente memoria media y pico respecto a FP32, siendo INT4 el más ligero.
4. **Frontera de Pareto (Memoria vs Latencia)**
    - Técnicas en la frontera de Pareto (memoria pico vs latencia media):
        - *Pruning 70 % + KD INT4*: $\left(0{,}880\,\text{GB},\ 0{,}887\,\text{s}\right)$
        - *Pruning 70 % + KDF P16*: $\left(1{,}509\,\text{GB},\ 0{,}406\,\text{s}\right)$
    - FP32 e INT8 quedan dominadas por combinaciones con mejor memoria y/o latencia.
    - Elección según trade-off deseado: *INT4* para mínimo consumo de memoria con latencia media moderada. *FP16* para latencia media mínima con memoria reducida.
5. **Conclusiones Finales**
    - La cuantización tras Pruning 70 % + KD permite ajustar finamente el balance calidad-recursos: FP16 retiene perfectamente calidad y mejora latencia, INT4 maximiza el ahorro de memoria a costa de mayor perplejidad y latencia moderada.
    - Según el caso de uso (entornos de muy baja latencia vs memoria extremadamente limitada), cada punto de la frontera de Pareto ofrece la mejor opción.

Conclusiones finales para Pruning KD cuantizado

En conjunto, nuestros experimentos demuestran que la combinación de pruning agresivo con Knowledge Distillation y cuantización ofrece un abanico de configuraciones adaptables a distintos escenarios de despliegue. Si el objetivo principal es mantener la calidad del modelo casi intacta mientras se reduce la latencia, la opción de pruning moderado (20 %) + KD en FP16 emerge como la más equilibrada: conserva una perplejidad y Top-5 equivalentes a FP32 y disminuye sensiblemente tanto la latencia media como el consumo de memoria. Para entornos con restricciones de memoria muy estrictas (por ejemplo, dispositivos edge con pocos cientos de megabytes), los esquemas INT8 o INT4 tras pruning del 20 % y KD alcanzan mínimos absolutos de huella de memoria —con un coste casi imperceptible en accuracy—, aunque a cambio de picos de latencia algo mayores. A niveles de pruning más agresivos (50 % y 70 %), KD recupera sorprendentemente gran parte de la calidad perdida, y de nuevo FP16 brinda la mejor latencia media, mientras que INT4 maximiza el ahorro de memoria. Por tanto, la elección final dependerá de cuánto estemos dispuestos a sacrificar de accuracy: desde mantener la fidelidad total en FP16 hasta tolerar una pequeña degradación en INT4 a cambio de footprint mínimo. Además, para tareas muy específicas (traducción de un dominio concreto, chatbots de nicho, etc.), podría plantearse un pruning aún más extremo seguido de un fine-tuning especializado y potente, lo que probablemente permitiría lograr rendimientos de alta calidad con mínimos recursos y ejecutar el modelo sin problemas en dispositivos edge.

15. Conclusiones

A lo largo de los experimentos realizados se ha demostrado que cada técnica de optimización aporta ventajas claras según el escenario de despliegue. La cuantización en FP16 ofrece un excelente compromiso, reduciendo drásticamente el uso de memoria al tiempo que conserva buena parte de la calidad de inferencia, aunque con un ligero sacrificio en precisión y latencia en comparación con INT4. El pruning puro, especialmente en su punto intermedio de 50 % de poda, equilibra de forma consistente el consumo de memoria, la velocidad de respuesta y la exactitud del modelo. La incorporación de Knowledge Distillation recupera gran parte de la precisión perdida tras el pruning sin incrementar la huella de memoria y, de hecho, mejora la eficiencia de la inferencia. Finalmente, la combinación de pruning + KD + cuantización traza una frontera de Pareto muy flexible: mientras FP16 minimiza la latencia y preserva la fidelidad de salida, INT4 maximiza el ahorro

de memoria con una degradación moderada de calidad. Gracias a estas implementaciones, vemos claramente cómo nos acercamos al objetivo principal de "deployear" modelos en dispositivos más limitados y avanzar hacia escenarios de edge computing, donde un rendimiento ligeramente comprometido resulta aceptable para permitir un despliegue real en el punto de uso.

Aunque la calidad del modelo se ha visto algo comprometida, en ningún momento se ha buscado llevarla al extremo, sino alcanzar un punto óptimo que mantiene un amplio margen de mejora. Es lógico pensar que, mediante ajustes adicionales y reentrenamiento para casos de uso concretos, podríamos recuperar completamente la precisión perdida e incluso superar el rendimiento del modelo original en tareas especializadas. Esta flexibilidad demuestra el potencial de las técnicas combinadas para adaptar modelos de gran escala a entornos con recursos limitados sin renunciar por completo a la calidad.

15.1. Trabajo a futuro

Como líneas de trabajo futuro, resulta de particular interés profundizar en la escalabilidad de las diversas métricas de rendimiento tras la aplicación de las técnicas de optimización estudiadas. En concreto, cabría extender los experimentos incorporando secuencias de entrada de mayor longitud (tanto en número de tokens como en tamaño de batch) con el fin de evaluar la evolución de la latencia, el uso de memoria y la calidad de generación bajo condiciones más exigentes. Asimismo, sería altamente relevante replicar este análisis en modelos de lenguaje de mayor envergadura, con un número sustancialmente superior de parámetros, para determinar si los patrones de comportamiento observados en nuestra configuración inicial se mantienen, o bien si emergen nuevos cuellos de botella.

Del mismo modo, convendría evaluar el desempeño sobre distintas arquitecturas de hardware para analizar cómo influye la plataforma de ejecución en la efectividad de pruning, cuantización y knowledge distillation. Adicionalmente, sería beneficioso estudiar técnicas de optimización de la latencia que no comprometan significativamente la precisión, de modo que puedan compensar el aumento de latencia asociado a la reducción del uso de memoria.

De este modo, se pretende validar la aplicabilidad y robustez de las técnicas de pruning, cuantización y knowledge distillation en entornos cada vez más cercanos a escenarios de producción, caracterizados por cargas de trabajo intensivas y modelos de gran escala.

16. Bibliografía

Referencias

[1] Jie Peng et al. *Harnessing Your DRAM and SSD for Sustainable and Accessible LLM Inference with Mixed-Precision and Multi-level Caching*. 23 de oct. de 2024. DOI: 10.48550/arXiv.2410.14740. arXiv: 2410.14740[cs]. URL: http://arxiv.org/abs/2410.14740 (visitado 27-02-2025).

[2] Maria Jose Gamez. *Objetivos y metas de desarrollo sostenible*. Desarrollo Sostenible. URL: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (visitado 27-02-2025).

[3] Mirtha Moran. *Infraestructura*. Desarrollo Sostenible. URL: https://www.un.org/sustainabledevelopment/es/infrastructure/ (visitado 27-02-2025).

[4] Mirtha Moran. *Consumo y producción sostenibles*. Desarrollo Sostenible. URL: https://www.un.org/sustainabledevelopment/es/sustainable-consumption-production/ (visitado 27-02-2025).

[5] *Cambio climático*. Desarrollo Sostenible. URL: https://www.un.org/sustainabledevelopment/es/climate-change-2/ (visitado 27-02-2025).

[6] Mirtha Moran. *Reducir las desigualdades entre países y dentro de ellos*. Desarrollo Sostenible. URL: https://www.un.org/sustainabledevelopment/es/inequality/ (visitado 27-02-2025).

[7] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. 24 de mayo de 2023. DOI: 10.48550/arXiv.2305.14314. arXiv: 2305.14314[cs]. URL: http://arxiv.org/abs/2305.14314 (visitado 18-06-2025).

[8] Mingyang Zhang et al. *LoRAPrune: Structured Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning*. 8 de ago. de 2024. DOI: 10.48550/arXiv.2305.18403. arXiv: 2305.18403[cs]. URL: http://arxiv.org/abs/2305.18403 (visitado 18-06-2025).

[9] Yunqiang Li. *liyunqianggyn/Awesome-LLMs-Pruning*. original-date: 2023-11-07T10:11:45Z. 18 de jun. de 2025. URL: https://github.com/liyunqianggyn/Awesome-LLMs-Pruning (visitado 18-06-2025).

[10] Elias Frantar y Dan Alistarh. *SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot*. 23 de mar. de 2023. DOI: 10.48550/arXiv.2301.00774. arXiv: 2301.00774[cs]. URL: http://arxiv.org/abs/2301.00774 (visitado 18-06-2025).

[11] *Treball de Fi de Grau — Facultat d'Informàtica de Barcelona*. URL: https://www.fib.upc.edu/ca/estudis/graus/grau-en-enginyeria-informatica/treball-de-fi-de-grau (visitado 04-03-2025).

[12] *Sueldo: Analista Programador en España 2025*. Glassdoor. URL: https://www.glassdoor.es/Sueldos/analista-programador-sueldo-SRCH_KO0,20.htm (visitado 12-03-2025).

[13] *Sueldo: Programador en España 2025*. Glassdoor. URL: https://www.glassdoor.es/Sueldos/programador-sueldo-SRCH_KO0,11.htm (visitado 12-03-2025).

[14] *Sueldo: Director De Proyecto en España 2025*. Glassdoor. URL: https://www.glassdoor.es/Sueldos/director-de-proyecto-sueldo-SRCH_KO0,20.htm (visitado 12-03-2025).

[15] *Ley de IA — Configurar el futuro digital de Europa*. 17 de jun. de 2025. URL: https://digital-strategy.ec.europa.eu/es/policies/regulatory-framework-ai (visitado 18-04-2025).

[16] *Reglamento general de protección de datos (RGPD) — EUR-Lex*. Doc ID: 310401_2 Doc Sector: other Doc Title: General data protection regulation (GDPR) Doc Type: other Usr_lan: en. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=legissum:310401_2 (visitado 18-06-2025).

[17] Ashish Vaswani et al. *Attention Is All You Need*. 3 de ago. de 2023. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762 (visitado 18-06-2025).

[18] *Perplexity of fixed-length models*. URL: https://huggingface.co/docs/transformers/perplexity (visitado 18-06-2025).

[19] peremartra. *Large-Language-Model-Notebooks-Course/6-PRUNING/6_3_pruning_structured_1b_OK.ipynb at main · peremartra/Large-Language-Model-Notebooks-Course*. GitHub. URL: https://github.com/peremartra/Large-Language-Model-Notebooks-Course/blob/main/6-PRUNING/6_3_pruning_structured_llama3.2-1b_OK.ipynb (visitado 18-06-2025).

[20] peremartra. *Large-Language-Model-Notebooks-Course/6-PRUNING/7_1_knowledge_distillat at main · peremartra/Large-Language-Model-Notebooks-Course*. GitHub. URL: https://github.com/peremartra/Large-Language-Model-Notebooks-Course/blob/main/6-PRUNING/7_1_knowledge_distillation_Llama.ipynb (visitado 18-06-2025).

[21] *bitsandbytes*. URL: https://huggingface.co/docs/bitsandbytes/main/index (visitado 18-06-2025).

A. Repositorio con las gráficas empleadas

https://github.com/ItzGeness/TFG/tree/main/grafiacs

B. Repositorio con los códigos empleados

https://github.com/ItzGeness/TFG