



UNIVERSIDAD
POLITÉCNICA
DE MADRID



EPES
Escuela Politécnica
de Enseñanza Superior



UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA POLITÉCNICA DE ENSEÑANZA SUPERIOR (EPES)
TRABAJO FIN DE GRADO
GRADO EN MATEMÁTICAS

Transcripción automática de manuscritos de Grothendieck mediante fine-tuning de LLM

Autor: Ignacio Martínez Cerezo

Dirigido por: Alberto Navarro Garmendia,
Jaime Arboleda Castilla

Madrid, Junio de 2026



Transcripción automática de manuscritos de Grothendieck mediante fine-tuning de LLM

Autor: Ignacio Martínez Cerezo

Dirigido por: Alberto Navarro Garmendia, Jaime Arboleda Castilla

Trabajo Fin de Grado, Junio de 2026

Escuela Politécnica de Enseñanza Superior (EPES)

Edificio C, Paseo Juan XXIII, 11. Madrid

Si deseas citar este trabajo, la entrada completa en BibTeX es la siguiente:

```
@mastersthesis{citekey,
  title   = {Transcripción automática de manuscritos de Grothendieck
             mediante fine-tuning de LLM},
  author  = {Ignacio Martínez Cerezo},
  school  = {Escuela Politécnica de Enseñanza Superior (EPES)},
  year    = {Junio de 2026},
  type    = {Trabajo Fin de Grado}
}
```

Este trabajo está bajo una Licencia Creative Commons «Atribución-NoComercial-CompartirIgual 4.0 Internacional»

Plantilla derivada de <https://github.com/blazaid/UPM-Report-Template>.



■ Agradecimientos

A mi abuelo, por despertarme el interés en las matemáticas.

A mis padres, hermanas y el resto de mi familia, por un apoyo ineludible.

A mis amigos, todos ellos, por soportarme... por difícil que se lo haya puesto.

A mis tutores, Alberto y Jaime, por los cientos de correos y las horas de reuniones.

A los anotadores, pues sin su labor nunca habría terminado a tiempo.

A Juan Francisco Padial, por prestarnos tan generosamente un lugar donde realizar el entrenamiento.

A todos los profesores y maestros de mi vida.

■ Resumen

Alexander Grothendieck fue uno de los matemáticos más prolíficos del siglo XX. Sin embargo, la vasta obra manuscrita de su etapa de aislamiento en Lasserre permanece inaccesible debido a su ilegibilidad y gran volumen. Este Trabajo Fin de Grado aborda el problema mediante el ajuste fino de un gran modelo de lenguaje visual (LVLM) de pesos abiertos, específicamente la versión de 8 mil millones de parámetros de Qwen3-VL. El entrenamiento se realiza de forma local utilizando adaptación de bajo rango cuantizada, conocida como QLoRA. Se ha desarrollado una infraestructura de ingesta y etiquetado colaborativo mediante Label Studio para generar un conjunto de datos propio a partir de antiguas transcripciones en TeX. El proceso formativo del modelo sigue un esquema de aprendizaje por currículum, exponiéndolo progresivamente a muestras desde una sola línea hasta fragmentos multilínea. Se aplica además una contextualización estocástica para que el modelo aprenda a inferir iterativamente en secuencias largas. Los resultados evidencian la viabilidad técnica de especializar este tipo de arquitecturas en la transcripción de textos manuscritos con alta densidad matemática, abriendo la puerta a la digitalización de sus obras inéditas.

Palabras clave: Reconocimiento óptico de caracteres, Grandes modelos de lenguaje visual, Ajuste fino, Transcripción de manuscritos, Alexander Grothendieck, QLoRA, Aprendizaje por currículum

■ Abstract

Alexander Grothendieck was one of the most prolific mathematicians of the 20th century. However, the vast handwritten manuscripts from his late isolation in Lasserre remain completely unread due to their illegibility and sheer volume. This Bachelor's Thesis tackles this issue by fine-tuning an open-weight Large Vision-Language Model (LVLM), specifically Qwen3-VL 8B. The training is executed locally employing Quantized Low-Rank Adaptation (QLoRA). A collaborative ingestion and annotation pipeline was built utilizing Label Studio to extract a custom dataset from older TeX transcriptions. The model's training strictly follows a curriculum learning approach, progressively introducing the neural network to samples ranging from single text lines to complex multi-line blocks. Furthermore, stochastic contextualization is applied to force the model to properly conduct iterative inference over long sequences. The results demonstrate the technical feasibility of specializing these architectures for transcribing dense, math-heavy handwritten texts, paving the way for the digitization of Grothendieck's unpublished work.

Keywords: Optical character recognition, Large vision-language models, fine tuning, Manuscript transcription, Alexander Grothendieck, QLoRA, Curriculum learning

■ Índice General

1. Introducción	1
1.1. Alexander Grothendieck	2
1.2. La obra de Lasserre	7
1.3. Sistemas de OCR	10
1.4. Modelo	16
2. El transformer	20
2.1. Introducción al NLP	20
2.2. RNN	22
2.3. El transformer	28
2.4. Entrenamiento por ajuste fino	39
2.5. Adaptadores de visión	47
3. El dataset	50
3.1. Obtención	51
3.2. Transcripción y pretratamiento	51
3.3. Etiquetado	53
3.4. Aumentado de datos	59
3.5. División entrenamiento/validación	60
3.6. Sistema de evaluación	62
4. Modelo y entrenamiento	65
4.1. Modelo empleado	66
4.2. Entrenamiento	70
4.3. Resultados obtenidos	75
5. Conclusiones y trabajo futuro	85
5.1. Conclusiones generales y análisis del rendimiento	86
5.2. Posibles trabajos futuros	88
A. Implementación	101
A.1. Pretratamiento de <i>La longue marche</i>	102
A.2. Label Studio, servidor de Oracle	106

A.3. Ingesta de datos de Label Studio	107
A.4. Data collators, Unsloth	122
B. Notas sobre LLMs	130
B.1. Características de un LLM	130
B.2. Etapas del entrenamiento	136
B.3. Ajuste fino supervisado	138
B.4. Entrenamiento de refuerzo	140
B.5. Alucinaciones	143
Bibliografía	146

■ Índice de Figuras

Figura 1	Izquierda: Fotografía de Alexander Grothendieck y Jean-Pierre Serre en una fiesta en 1958. Derecha: Fotografía de Alexander Grothendieck y Laurent Schwartz, 1961. [1]	4
Figura 2	Fotografía de Grothendieck, en uno de sus viajes a Vietnam durante la guerra, tomada en 1967. Durante este período realizaba seminarios sobre álgebra en zonas de guerra como forma de activismo pacifista.	5
Figura 3	Izquierda: Ubicación de Lasserre en el pirineo francés (imagen de Google Maps). Derecha: Residencia de Grothendieck en Lasserre, derecha. [1]	7
Figura 4	Fotografía de los documentos, organizados en clasificadores, en sus posiciones originales en su casa de <i>Lasserre</i> . Tomada tras su muerte en 2014. [1]	8
Figura 5	Fragmento de una de las páginas de <i>El problema del Mal</i> . Fuente: Libération	8
Figura 6	Diagrama simplificado de la estructura del modelo TrOCR presentado por Microsoft.	13
Figura 7	Diagrama de una neurona en una RNN procesando el elemento t -ésimo de una secuencia. En este, $h_{t-1}^{[\ell]}$ representa el <i>estado oculto</i> de la capa ℓ producido al procesar el elemento $(t - 1)$ -ésimo. Notemos que este valor debe almacenarse entre fragmentos para poder emplearse.	23
Figura 8	Diagrama del despliegue de una red neuronal recurrente para el cálculo de retropropagación.	24
Figura 9	Diagrama general de una RNN de tipo seq2seq.	25
Figura 10	Diagrama del proceso de activación del decoder con un bloque de atención. El bloque de atención recibe todos los estados ocultos y el	

	vector de <i>query</i> S_t , generando el estado oculto V_{t+1} para la activación del decoder.	27
Figura 11	Diagrama de un transformer de tipo encoder-decoder.	29
Figura 12	Bloque de atención policéfala. La salida de todas las cabezas de atención se concatena y combina mediante W_O en una sola matriz de salida de las dimensiones correctas.	33
Figura 13	Diagrama del encoder como composición de bloques de encoder.	35
Figura 14	Diagrama de un bloque de encoder donde, mediante « $\oplus$ » estamos simbolizando la adición residual, en la que el resultado de la autoatención y su entrada, apropiadamente denotada «Entrada», se suman. Notemos que el bloque de atención puede tener múltiples cabezas. [2], [3]	36
Figura 15	Diagrama del decodificador como composición de bloques decodificantes.	37
Figura 16	Diagrama del decodificador en un transformer. [2], [3]	37
Figura 17	A la izquierda un ejemplo de una de las cartas, a la derecha una de las páginas de <i>La longue marche à travers la théorie de Galois</i>	52
Figura 18	Interfaz configurada de Label Studio ante una tarea ya anotada. Captura tomada directamente del servidor de Label Studio, accesible en grothendieck.online	55
Figura 19	Campo de corrección en la interfaz de Label Studio	56
Figura 20	Esquema de la relación entre el <i>bucket</i> , la instancia de cómputo (el «servidor») y los anotadores.	58
Figura 21	Representación de los patrones de atención. A la izquierda la atención estándar; en el centro un patrón de atención diagonal por bloques, notemos que no permite a la información propagarse entre bloques; a la derecha el patrón de doble bloque, empleado por Qwen3, mediante el que se generalizan los patrones de atención de 4096 tókenes a mayores longitudes. [4], [5]	67
Figura 22	Diagrama de la obtención de las representaciones intermedias V_1, V_2, V_3 , así como de las finales, V_4 , a partir del ViT SigLIP2.	68
Figura 23	Diagrama del proceso de enriquecimiento progresivo <i>deepstack</i> empleado para procesamiento de tókenes visuales en la familia <i>Qwen3-VL</i>	70

Figura 24	Tres de los 303 ejemplos con los que cuenta este pequeño conjunto de datos de evaluación.	76
Figura 25	Tres de los 50 ejemplos con los que cuenta el conjunto de datos de evaluación de matemáticas.	76
Figura 26	CER promedio del modelo base y el modelo entrenado en cada página del conjunto de validación.	79
Figura 27		80
Figura 28		80
Figura 29		80
Figura 30	CER promedio del modelo base y el modelo entrenado en cada página del conjunto de validación, retirando las muestras con matemáticas.	81
Figura 31	CER promedio del modelo base y el modelo entrenado en cada página del conjunto de validación, retirando las muestras con matemáticas y las cartas.	82
Figura 32	Separación de un párrafo único en dos párrafos efectivos mediante un algoritmo de clustering. Se calcularía la calidad de diferentes agrupamientos, cambiando el número de grupos, para decidir si se divide o no el párrafo, y en qué subpárrafos.	95
Figura 33	Dos páginas del documento original descargado de la web de la Universidad de Montpellier. Los bordes verdes exteriores marcan los bordes de página.	102
Figura 34	Imagen de dos cajas-imagen poligonales con rotaciones muy diferentes, sus respectivos polígonos (rojo) y sus rectángulos de área mínima (verde) sobre un fondo negro, en sus posiciones relativas en la imagen original.	110
Figura 35	Un fragmento de una de las páginas manuscritas	111
Figura 36	Diferentes grupos de líneas, ya sea entre páginas o dentro de una misma página, presentan muy diferentes inclinaciones. Esto descarta el orden lexicográfico en cualquiera de sus alternativas, y dificulta (o imposibilita también) escoger una expresión para la combinación de coordenadas que funcione en todos los casos. ..	114
Figura 37	Dos párrafos que solapan completamente. Puesto que hay palabras escritas sobre otras, no es posible recuadrar las líneas dividiendo limpiamente en dos párrafos en una misma anotación.	115

Figura 38	Diagrama de la división en dos tareas para separación de párrafos entrelazados.	116
Figura 39	Diagrama de la generación del dataset, su subida a <i>Huggingface</i> y su posterior carga, de forma paralela al modelo, en el equipo donde se ejecuta el <i>fine-tuning</i>	123
Figura 40	Diagrama de funcionamiento simplificado de las partes de <i>Unsloth</i> que intervienen en nuestro entrenamiento y hemos personalizado.	124

■ Índice de Tablas

Tabla 1	Resultados del modelo base sobre los conjuntos de datos sintéticos.	76
Tabla 2	Resultados del modelo base sobre nuestro conjunto de evaluación. .	78
Tabla 3	Resultados del modelo extraído del <i>checkpoint</i> 1200 sobre nuestro conjunto de evaluación.	78

1.

Introducción

Padre de la teoría de esquemas, de topos, de motivos, de la cohomología *étale*... Alexander Grothendieck fue uno de los matemáticos más prolíficos del siglo XX, y probablemente de todos los tiempos. Su visión, guiada por un particular enfoque a la hora de hacer matemáticas de forma sutil y sin grandes *muestras de fuerza*, revolucionó entre otras, la geometría algebraica. Dejó su eminente impronta en muchos otros campos, siendo su obra germinal e instrumental para los trabajos que llevaron a la Medalla Fields de, entre otros, Pierre Deligne (1978), la de Gerd Faltings (1986) y la de Vladimir Voevodsky (2002), además de obtener Grothendieck la suya propia en 1966.

La obra de Grothendieck puede dividirse en tres grandes bloques: una primera parte de su producción, que sigue a su tesis; aquella que, debido a su descontento con la comunidad matemática, siguió a un retiro parcial de diez años; y, finalmente, la que realizaría en aislamiento los últimos 23 años de su vida. Mientras que las dos primeras son bien conocidas y estudiadas, habiendo resultado en trabajos, ideas y exploraciones de la más alta calidad matemática, la última parte de su obra permanece, aún a día de hoy, 11 años tras su muerte, sin haberse leído.

La justificación es sencilla: la tarea de transcripción de estas obras, tanto por su volumen y dificultad como por requerirse estar físicamente en París¹ para su acceso por cuestiones legales, resulta prácticamente inasumible. Nuestra intención con el presente Trabajo Fin de Grado es realizar un ajuste fino sobre

¹En la *Bibliothèque Nationale de France*, (BNF).

un gran modelo de lenguaje visual (VLM o LVLM) a partir de manuscritos de Grothendieck, para afinarlo en la tarea de transcripción de forma que sea posible automatizar la digitalización de la última etapa de su obra.

1.1. Alexander Grothendieck

La completitud de esta sección está basada en [6], así como las notas biográficas de [7] y, en menor medida, la propia obra de Grothendieck [8].

Nacido en 1928 de padre ruso anarquista y madre alemana, viviría desde su más tierna infancia los efectos de la Alemania nazi. Sus padres, presionados por la situación política vivida y movidos por su ideología, abandonaron el país dejando atrás a un Grothendieck de apenas cinco años en una casa de acogida. Seis años más tarde, Grothendieck tendría que viajar solo hasta Francia desde Hamburgo para reencontrarse con su madre; sería en Francia, tras ser recluido con ella en el campo de refugiados de Rieucros en 1940, donde terminaría su infancia y pasaría la mayor parte de su vida.

1.1.1. Una educación nómada

Durante su estancia en el campo de Rieucros junto a su madre, en condiciones de vida extremadamente difíciles y pese a las privaciones de la guerra y el encierro, Grothendieck gozaba del relativo privilegio de poder asistir a la escuela. Comenzaría su educación secundaria en la localidad cercana de Mende, donde permanecería escolarizado hasta 1942². En este año la situación política y social francesa se complicó con el endurecimiento del régimen de Vichy y las deportaciones masivas. Para protegerlo, Alexander fue enviado a una zona de refugio excepcional en Chambon-sur-Lignon.

Una vez en Chambon-sur-Lignon, Grothendieck comenzaría sus estudios en el Colegio Cévénol, una institución educativa humanista y pacifista con un ambiente intelectualmente estimulante y ético, donde Grothendieck completaría sus estudios en 1945.

Una vez terminado el bachillerato, un Grothendieck muy insatisfecho con las matemáticas que ha recibido hasta el momento se traslada, junto a su madre,

²Su padre, que había sido llevado al campo de Le Vernet, sería entregado a los nazis este mismo año, siendo poco después asesinado en Auschwitz.

a una aldea cercana a Montpellier, donde pretende cursar la Licenciatura en Matemáticas. Tanto él como su madre, debilitada y enferma tras su paso por el campo de reclusión, vivieron una vida de extrema pobreza, subsistiendo mediante una beca de estudios, el dinero ganado por un joven Grothendieck vendimiando y lo poco que podían cultivar.

El nivel matemático de la Universidad de Montpellier no era el más alto, lo que, lejos de desilusionar a Grothendieck, le serviría como combustible intelectual y propulsor para asentar un autodidactismo que se había fraguado ya en el instituto. Tras tener que cursar un tercer año de la licenciatura por suspender una asignatura (al cometer un error de cálculo – Grothendieck sufría con el cálculo numérico), llamó la atención de uno de sus mentores, que le recomendaría viajar a París para encontrarse con Élie Cartan. Gracias a una beca, Grothendieck consigue seguir la recomendación y viajar a París en 1948. Allí, en lugar de a un Élie ya retirado, conoce a su hijo Henri Cartan, que habría de tomar una posición clave en la formación de la nueva generación de matemáticos franceses que siguió a la II Guerra Mundial.

Es en el París del 48 donde Grothendieck se sumerge en la efervescente escena matemática del momento, donde por fin consigue hallar *qué son las matemáticas*, algo alejadas de cómo se le habían presentado en Montpellier. Tras el curso, Cartan y André Weil le recomiendan familiarizarse con la teoría de espacios vectoriales topológicos, para lo que le sugieren marcharse a la universidad de Nancy y tomar contacto con L. Schwartz, J. Dieudonné, R. Godement y J. Delsarte.

Ya en Nancy y bajo la tutela de Laurent Schwartz y Jean Dieudonné, estos tratan de calibrar el potencial de Grothendieck mediante una lista de 14 problemas. Menos de un año después y con todos ellos resueltos, la genialidad de Grothendieck les es ahora abundantemente evidente. Se embarca entonces en la elaboración de su tesis, una que, por su ingente tamaño, tuvo que ser cribada y reducida por sus tutores, pues había escrito el equivalente a seis distintas.

Pese a su genialidad, las dificultades económicas de la época forzaron que buscara un puesto de profesor en la Universidad de São Paulo, pues no tenía la nacionalidad francesa. Pese a esto, Grothendieck no se desconectó del panorama matemático francés, realizando durante este periodo numerosos viajes a París. En 1953 Grothendieck comienza a abandonar los temas proporcionados por sus tutores para acercarse a corrientes más puras, entre ellas la geometría algebraica. En esta línea, Grothendieck se instala en 1955 en Kansas, donde se aceleraría y comenzaría a florecer este cambio en sus intereses.

1.1.2. Grothendieck como polo de la matemática

Grothendieck publica en 1957 el *Tohoku* [9], donde propone las categorías abelianas. Este mismo año publica también una generalización del Teorema de Riemann-Roch, una versión *relativa* del mismo, para lo que introdujo la noción de Teoría- K . Esta demostración serviría como muestra definitiva para la comunidad matemática de la calidad de Grothendieck, organizando Armand Borel y Jean-Pierre Serre un seminario en la Universidad de Princeton para estudiar su demostración, redactarla y publicarla en su nombre.

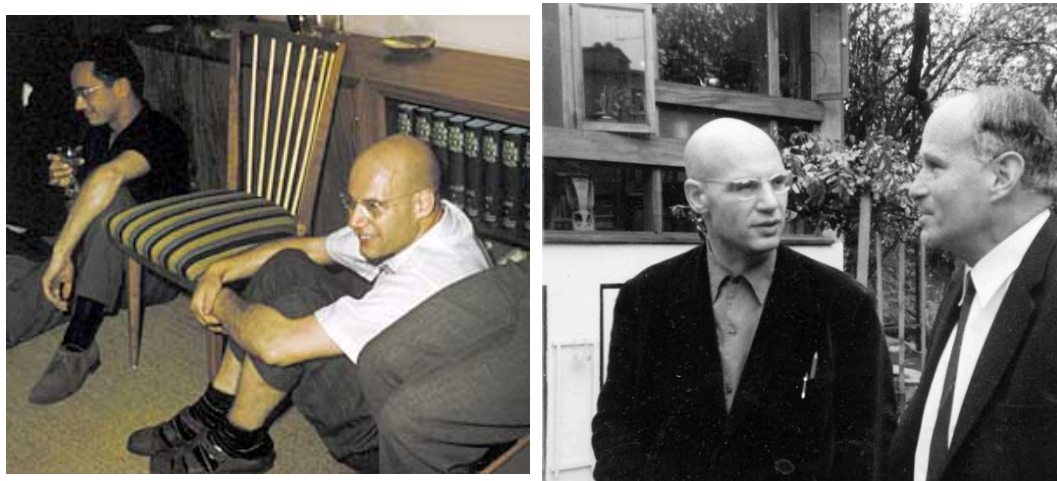


Figura 1. Izquierda: Fotografía de Alexander Grothendieck y Jean-Pierre Serre en una fiesta en 1958. Derecha: Fotografía de Alexander Grothendieck y Laurent Schwartz, 1961.[1]

Se funda en 1958 en Bures-sur-Yvette, cercano a París, el *Institut des Hautes Études Scientifiques* (IHÉS). Allí, Grothendieck y Diudonné son los primeros en ser nombrados profesores permanentes. Fue en el IHÉS donde Grothendieck desarrollaría el núcleo de su obra, donde escribiría los *Éléments de Géométrie*

Algèbrique y dirigiría el *Séminaire de Géométrie Algébrique* que sirvió como crisol y atractor para gran parte del talento matemático de la obra. Entre otras, Grothendieck introdujo durante este periodo la noción de *topos* y de *sitio*, la cohomología ℓ -ádica y la teoría de esquemas.

Grothendieck llevó a cabo una labor descomunal en el IHÉS, no solo trabajando individualmente en obras de un enorme calado, sino inspirando y contagiando en su entorno un ritmo de trabajo casi sobrehumano. Esta labor culminaría en 1966 con la Medalla Fields, en el Congreso Internacional de Matemáticos de Moscú.

El ambiente, sin embargo, no estaba exento de tensiones. Grothendieck había tenido una infancia vertebrada por la guerra, lo que lo marcaría toda su vida. Como conocedor de sus horrores, era una persona intensamente pacifista y, cuando un día descubrió que el IHÉS recibía financiación militar, se opuso a ello frontalmente. Tras intentar y fallar en amasar el apoyo de sus compañeros para pedir el rechazo de los fondos, Grothendieck presentó su renuncia del IHÉS. Grothendieck terminaría dejando atrás no solo este fructífero y efervescente ambiente, sino también la ciudad de París al no lograr establecerse de manera estable en ninguna de sus universidades.

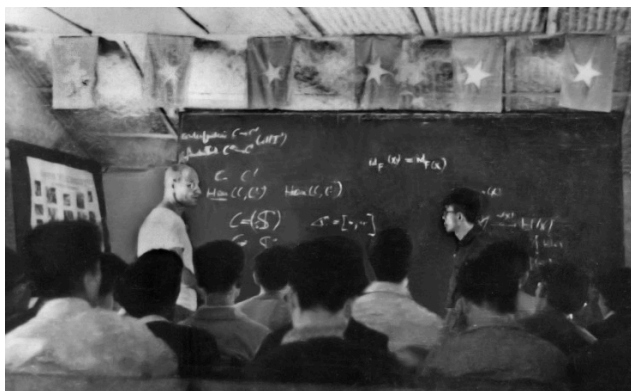


Figura 2. Fotografía de Grothendieck, en uno de sus viajes a Vietnam durante la guerra, tomada en 1967. Durante este período realizaba seminarios sobre álgebra en zonas de guerra como forma de activismo pacifista.

1.1.3. Grothendieck y la comunidad matemática

Comenzaría entonces una etapa de su vida marcada por un fuerte activismo pacifista y ecologista. Tras permanecer entre 1970 y 1973 en el Collège de France, donde apenas realizó labor matemática, aceptó un puesto en su *alma mater*, la

Universidad de Montpellier. Una vez allí comenzaría un progresivo alejamiento con sus colegas y exalumnos de París y, aunque con numerosos parones, continuó discretamente su actividad matemática, entre otras dirigiendo tesis doctorales.

Tras años de enfriamiento, Grothendieck comienza en 1981 con *La longue marche à travers la théorie de Galois* (un manuscrito de 1600 páginas donde introduce la geometría algebraica anabeliana) una particular dinámica de difusión de sus escritos: alejados del habitual formato de artículo y siendo tomos enormes, las obras de esta época se transmiten de mano a mano o mediante correspondencia entre un selecto grupo de matemáticos cercanos a su visión. Debido a su altísima calidad, esta parte de su obra consiguió perdurar en el tiempo, siendo de este período tres importantes obras que sucedieron a *La longue marche à travers la théorie de Galois: À la poursuite des champs* (1983), *Esquisse d'un programme* (1983) y *Les Dérivateurs* (1991).

Grothendieck se encuentra durante este periodo, sin embargo, con un enorme obstáculo: el mundo matemático había cambiado durante su ausencia, y sus teorías habían tomado rumbos que él ya no compartía. Su desacuerdo con la comunidad quedó manifiesto en su obra *Récoltes et Semailles*, escrita entre 1985 y 1986, donde mezcla autobiografía con sociología y filosofía. En ella, Grothendieck manifiesta sentirse traicionado, mostrando un fuerte rechazo a una comunidad que, siente, ha pervertido y alienado su obra.

1.1.4. Aislamiento y muerte

Grothendieck abandonó formalmente Montpellier en 1984, trasladándose al *Centre national de la recherche scientifique* (CNRS), donde conseguiría un puesto gracias al potencial que muestra su trabajo en *Esquisse d'un programme*. Trabajaría en el CNRS durante cuatro años, hasta su jubilación en 1988. Este mismo año Grothendieck rechazó el premio Crafoord de las Ciencias de Suecia, en línea con su rechazo a la comunidad matemática y sus motivaciones.

Tras terminar *Les Dérivateurs* en 1991, con 63 años y ya jubilado, Grothendieck toma la decisión de cortar todo lazo con la sociedad: limpió el que había sido su hogar, quemó gran cantidad de papeles personales y se marchó, sin dejar rastro ni destino conocido, a un lugar indefinido.

Conocemos, a día de hoy, el lugar a donde se marchó: fue Lasserre, una pequeña y remota aldea en el pirineo francés de apenas 200 habitantes.

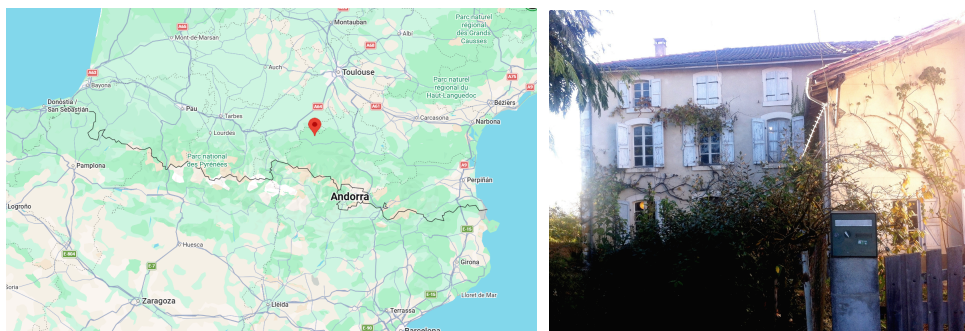


Figura 3. Izquierda: Ubicación de Lasserre en el pirineo francés (imagen de [Google Maps](#)).

Derecha: Residencia de Grothendieck en Lasserre, derecha. [1]

Tras una breve temporada en la zona, abandonaría también las escasas incipientes amistades que había forjado con sus reducidos vecinos, eliminando así el último reducto de sociedad que quedaba en su vida. Grothendieck comenzaría entonces una vida de casi completa soledad, que se prolongaría hasta su muerte 23 años más tarde, en 2014.

1.2. La obra de Lasserre

La escritura para Grothendieck era un requisito casi obligatorio para el pensamiento. No es de extrañar que durante su aislamiento Grothendieck no perdiera esta particular forma de pensar por tanto, al entrar en el que había sido su último domicilio, se descubriera que nunca había dejado de escribir. Se encontraron decenas de miles de páginas, que, organizadas en diferentes volúmenes, tratan desde la matemática pura hasta la filosofía o incluso teología y misticismo.



Figura 4. Fotografía de los documentos, organizados en clasificadores, en sus posiciones originales en su casa de Lasserre. Tomada tras su muerte en 2014. [1]

De entre estas obras organizadas *a posteriori* en tomos, la más copiosa de ellas es *El problema del Mal*, que cuenta con en torno a 40 000 páginas. Se trata, en concreto, de una obra de temática principalmente filosófica, en la que Grothendieck reflexiona sobre el origen del mal en el mundo y su experiencia con él. La obra no solamente es impactante por su extensión, sino también por su casi inextricable caligrafía, propia de un Grothendieck que permaneció en aislamiento hasta 23 años.

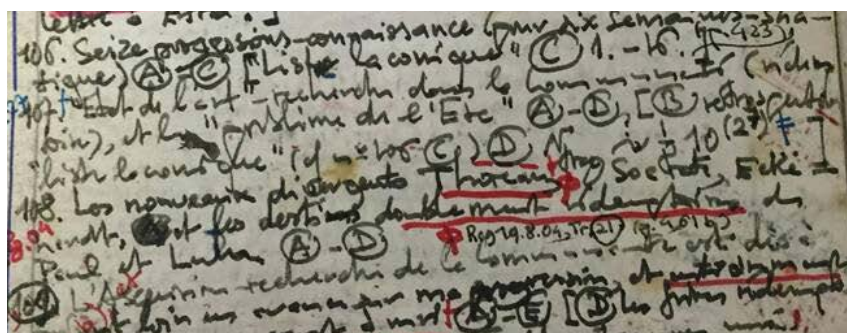


Figura 5. Fragmento de una de las páginas de *El problema del Mal*. Fuente: [Libération](#).

La combinación de extensión y dificultad hacen tanto de este texto como de sus congéneres de Lasserre un formidable reto: transcribirlos, o siquiera leerlos, resulta virtualmente imposible si se hace a mano. Sin mejor acceso a su contenido es poco probable que pueda leerse en los próximos años, prolongando así los más de once que la obra lleva ya sin ser leída, desde su descubrimiento en 2014.

1.2.1. La situación del manuscrito

La mayor parte de sus escritos anteriores a 1990 se encuentran en la Universidad de Montpellier, pues Grothendieck nombró en 1990 albacea a Jean Malgoire, que terminaría donándolos al órgano universitario para que los preservase y permitiese su acceso. De las 20 000 [10] páginas con las que contaba este fondo, solo algunas son accesibles, siendo el resto correspondencia privada con terceros con cuya aprobación de publicación no se cuenta. Muchos de estos son también accesibles mediante el Grothendieck Circle, una iniciativa que nace de J. Malgoire y otros discípulos de Grothendieck para conservar y promover su obra. La situación se enturbia considerablemente cuando se considera, sin embargo, que Grothendieck firmó en 2010 una declaración de no publicación [11], con evidentes consecuencias en la comercialización, explotación y diseminación de su obra.

La situación de los documentos hallados en Lasserre, por tanto, está profundamente marcada tanto por la publicación de este documento como por el traspaso de la propiedad de Grothendieck a sus cinco hijos tras su muerte. La situación, lejos de estar asentada, permanece desde hace años en una disputa legal tripartita entre los herederos de Grothendieck, la *Bibliothèque Nationale de France* (BNF) y la Universidad de Montpellier. [12], [13]

Actualmente, es posible solicitar una revisión de los documentos originales en calidad de investigador, siempre bajo la supervisión del personal de la BNF. Estos no pueden sacarse de la biblioteca, así como tampoco pueden extraerse fotografías ni escaneos de los mismos de la biblioteca (Alberto Navarro Garmendia, 24 de junio de 2026).

1.2.2. El valor y futuro de estos manuscritos

Grothendieck, pese a ser una persona profundamente atormentada y haber llevado una vida esculpida por abundantes desgracias, era un genio. No solamente es sobresaliente su obra matemática, sino también sus reflexiones sobre su comunidad y sus disquisiciones filosóficas en *La Clef des Songes*. Todos son temas que tratan en los documentos que dejó en Lasserre. [12], [13]

Su obra sería la base para el trabajo de numerosos matemáticos posteriores, entre ellos las tres Medallas Fields mencionadas en la introducción. El estudio de su obra, sin embargo, termina de forma abrupta con su separación de la

sociedad, pues no existe la posibilidad de leerla de forma sencilla; por un lado por la particular situación legal de los escritos y, por otro, por su dificultad de descifrado.

1.3. Sistemas de OCR

Como ya hemos establecido, la transcripción manual de la obra tardía de Grothendieck es, en la práctica, imposible de realizar (como sí fue posible con otras obras como *Les Dérivateurs* [14]). La única alternativa es, por tanto, conseguir un sistema que las transcriba de forma automática o facilite considerablemente la labor de un transcriptor.

Definición (Sistema de OCR): Llamamos *sistema de OCR* (del inglés *Optical Character Recognition*) a todo sistema capaz de transformar texto manuscrito presente en formato digital (como una imagen de un texto o un SVG de los caracteres en él presentes) en texto codificado para un sistema informático. ○

Es habitual que un sistema de OCR se divida en, al menos, dos submodelos: uno, el segmentador, divide el texto en fragmentos más pequeños (párrafos, líneas o palabras) y posiblemente corrige distorsiones o su orientación, mientras que otro, el de reconocimiento, transcribe el texto en cada uno de estos segmentos.

Centrándonos en el reconocimiento, actualmente existe una enorme diversidad de arquitecturas; a grandes rasgos, podemos realizar la siguiente clasificación de sistemas de OCR:

1. **OCR tradicional.**
2. **OCR basado en *transformers*** (TrOCR).
3. **Modelos grandes de lenguaje de visión** (*Vision Language Models*).

Existe una amplia gama de modelos dentro de cada una de estas categorías³, que se ofrecen incluso a nivel comercial para digitalización de textos. Estas soluciones, sin embargo, tienen un desempeño muy limitado en la obra de Grothendieck, por contar con tres características principales:

1. La difícil caligrafía de Grothendieck, especialmente la de las últimas etapas de su vida, pues los documentos no son artículos científicos con pretensión a ser publicados.

³La gama es, tal vez, más limitada en la segunda categoría.

2. Su compleja organización y composición, pues Grothendieck a menudo releía y corregía en la propia página original sus textos.
3. Su complejísimo contenido, pues o bien se trata de documentos de matemática extremadamente abstracta, o de filosofía profundamente entrelazada con su particular forma de pensar.

En estas aplicaciones altamente especializadas es necesario, por tanto, un proceso de adaptación y afinado del modelo para mejorar su rendimiento. Cada modelo específico emplea un sistema de afinado particular, para lo que primero debemos conocer su estructura.

1.3.1. Sistema de OCR tradicional

Los sistemas de reconocimiento de caracteres tradicionales funcionan principalmente extrayendo y clasificando *características* del texto presente en una imagen. Estos se conocen como sistemas *offline*, puesto que parten del resultado final; por otro lado, los sistemas *online* parten de imágenes sucesivas de la composición del texto, lo que los dota de información sobre la sucesión de trazos. Por la naturaleza de nuestra aplicación deseada, restringiremos nuestra exposición a los *offline*.

Dentro de estos se distinguen, asimismo, numerosas subclases [15], de entre las cuales las más relevantes son:

Coincidencia de plantillas (*template matching* a nivel de píxel) Se emplean plantillas que se deslizan a través de una muestra buscando máxima coincidencia. Si el valor de *activación* de una plantilla concreta en una posición de la imagen supera un umbral preestablecido, se marca como que se ha detectado un carácter en esa posición. Este tipo de modelos funcionan especialmente bien cuando se reconocen documentos escaneados y la tipografía, impresa o a máquina, es conocida.

Para poder extender estos modelos a situaciones menos controladas, se desarrollaron técnicas para poder transformar *ad hoc* las plantillas durante el reconocimiento, permitiendo deformaciones y transformaciones afines. Esto permite que, incluso si el escaneo no es perfecto, los caracteres se reconozcan de igual forma.

Reconocimiento de patrones estructurales Técnica similar a la anterior, pero, en lugar de emplear plantillas que representan caracteres íntegros, los descompone en átomos (bucles, ángulos rectos, líneas horizontales,

verticales, diagonales y curvas) y los busca, así como la forma en la que se interconectan (formando un doblez, ramificándose, encima, debajo, etc.), en la imagen. A partir de los niveles de activación de los patrones en cada región, y de la información sobre sus conexiones, se infiere cuáles son los caracteres detectados en cada región.

Modelos estadísticos Se basan en las propiedades estadísticas inherentes a las imágenes de diferentes caracteres. A partir de un conjunto de muestras y el número de caracteres diferentes se produce una agrupación de la misma en tantos subconjuntos como se haya indicado. Dependiendo de la métrica definida varía la clasificación obtenida, pudiendo emplearse diferentes métodos para el clustering como puede ser k -NN o modelos markovianos con estados ocultos. A partir de esta clasificación de los diferentes caracteres se compone el «texto más probable».

Esta métrica puede enriquecerse teniendo en cuenta no solamente el nivel de carácter, sino también cada grupo de caracteres formando palabras y cada grupo de palabras.

Modelos de Kernel sistemas basados en arquitecturas de tipo Kernel, que, como el resto de estas arquitecturas, basan su funcionamiento en el *truco del kernel*. En él, se calcula la similitud entre nuestros datos y los caracteres que buscamos en un espacio (a menudo infinito-dimensional) donde son fácilmente separables, todo ello sin tener que calcular esta representación explícitamente. En este espacio transformado se pueden emplear una amplia variedad de técnicas para detectar patrones y clasificar o agrupar nuestras muestras.

Redes neuronales artificiales (ANN) Basan su funcionamiento en redes neuronales entrenadas para el reconocimiento de caracteres. Las redes empleadas en este tipo de sistemas son sencillas: redes *feedforward* (perceptrones multicapa o MLP), redes neuronales recurrentes (RNN), redes neuronales convolucionales (CNN), etc. A menudo un mismo modelo de OCR empleaba simultáneamente diferentes arquitecturas de red neuronal en diferentes etapas [16]. Por ejemplo puede usarse una CNN para extraer las características visuales de una imagen y, tras esto, una RNN que procesa estas características visuales y produce la transcripción. A esto podía añadirse otro modelo más que evalúa la adecuación gramatical y semántica, realizando correcciones esporádicas y así paliar algunos de los errores.

Cada uno de estos modelos presenta un conjunto diferente de mejoras y desventajas frente al resto. De entre ellos, los modelos de Kernel dominaron hasta que las redes neuronales comenzaron a crecer, lo que les permitió finalmente ofrecer mejores resultados.

Esto se debe, en parte, a que empleando redes neuronales convolucionales se emplean procesos que, siendo similares al reconocimiento de patrones en esencia, se optimizan de forma automática para la inferencia mecánica. Así, eliminando la creación de patrones manuales, es capaz de asegurarse un cierto grado de optimalidad (al menos local) de estos patrones y conexiones de la red⁴.

1.3.2. TrOCR

Presentado por un equipo de Microsoft [16], el *OCR basado en transformers* (TrOCR) se propuso como una alternativa más *limpia* a las arquitecturas basadas en múltiples tipos de redes neuronales dispuestas secuencialmente. En lugar de diferentes tipos de redes neuronales, el sistema de TrOCR emplea un único tipo de bloque, el transformer, que estudiaremos con más profundidad en el **Capítulo 2**.

Un *transformer completo* emplea dos bloques: un codificador y un decodificador. En el caso de procesamiento de lenguaje, la primera parte recibe texto como entrada y la segunda produce texto como salida. El transformer, originalmente presentado para este tipo de datos, recibiría con la aparición del *Image Transformer* [17] (y, especialmente, el posterior *Vision Transformer* [18]) la posibilidad de procesar imágenes. El enfoque de TrOCR es emplear únicamente un codificador visual y un decodificador textual, es decir, es un modelo que únicamente puede recibir imágenes como entrada⁵ y producir texto como salida.

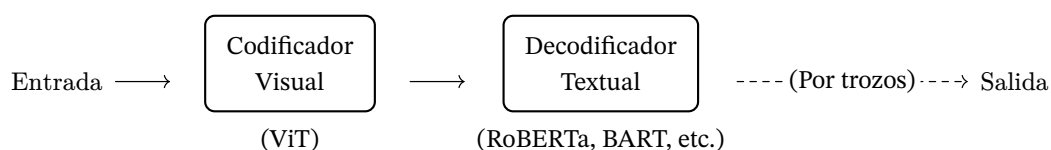


Figura 6. Diagrama simplificado de la estructura del modelo TrOCR presentado por Microsoft.

⁴Véase «[The bitter lesson](#)».

⁵El decodificador textual, pues es un modelo autorregresivo, sí puede técnicamente recibir texto como entrada, pero condiciona su funcionamiento de formas impredecibles. Esto se debe a que el modelo solo está entrenado para recibir su propia transcripción en proceso como entrada del *decoder* textual.

Por la naturaleza de este modelo, su aplicabilidad se enfoca especialmente a imágenes de textos aislado o que carecen de contexto lingüístico. Es texto (como la matrícula de un coche o una imagen de un logo) que no se encaja en una estructura textual superior más rica. Esta especialización se debe a que el modelo estructuralmente no acepta ese «contexto textual», sino únicamente visual: incluso en caso de tener acceso a este contexto textual, el modelo no tiene forma de procesarlo.

1.3.3. LLM multimodales para transcripción

Un modelo de lenguaje grande (LLM) es un modelo que emplea, al menos parcialmente, la arquitectura del transformer y cuenta con cientos o miles de millones de parámetros. Este tipo de modelos, especializados en procesar y producir⁶ lenguaje natural, conocen las propiedades estadísticas del lenguaje.

Los primeros LLM eran *unimodales*, en tanto que únicamente podían procesar texto. También es unimodal el ya descrito TrOCR, pues únicamente acepta imágenes como entrada.

Definición (Unimodalidad): Un modelo, en nuestro caso un LLM, se dice *unimodal* cuando únicamente es capaz de procesar entradas de una naturaleza, por ejemplo contenido textual. ○

Los LLM, al procesar texto de forma obligatoria, en el caso de ser unimodales no nos son útiles en la tarea de transcripción.

Definición (Multimodalidad en LLM): Decimos que un LLM es *multimodal* cuando admite como entrada información en un formato diferente al texto, como imágenes o audio. ○

Una implementación parcial de la multimodalidad consiste en equipar a los modelos con herramientas auxiliares (como otros sistemas de OCR y reconocimiento de escenas) que les proporcionen una descripción de la imagen. Lejos de ser óptima, esta solución es válida únicamente cuando la tarea del LLM se reduce a *incorporar* información visual a una respuesta, pero no requiere una comprensión profunda y fundamental del contenido de la imagen (pues el modelo realmente *no puede verla*). En caso de emplearla, estamos limitando

⁶Existen variantes que no producen texto, sino que solo obtienen una representación latente del mismo. Véase la [Subsección 2.3.5](#) del [Capítulo 2](#).

al modelo del lenguaje: este no intercepta una representación rica en contexto del anterior modelo, sino el producto de un modelo transcriptor completo y encapsulado que funciona de forma independiente.

Una solución a esta problemática son los adaptadores visuales⁷, que emplean transformers adaptados a entradas visuales, habitualmente ViT y variantes [19], [20], para recibir una versión rica en contexto de la información visual. Esta representación puede introducirse directamente en el LLM como «tókenes visuales» [20], [21], o pueden enriquecerse con esta información visual las representaciones intermedias de los tókenes textuales durante atención [19], [22]. La función del adaptador es, en este caso, convertir la representación de las imágenes del modelo visual en una representación congruente con la que espera el modelo y proporcionar al LLM una manera de fusionarlas.

1.3.4. Enfoques específicos dentro del OCR

Notemos que, de igual manera a como hemos podido realizar diversas distinciones dentro de los diferentes modelos, también es posible hacerlo dentro de los textos a procesar.

Definición (Tipos de textos según su grafía): Podemos distinguir los textos que recibimos en tres tipos dependiendo de la naturaleza de la grafía que contienen:

1. Texto digital: Renderizado directamente en pantallas, ya sea en píxeles (imágenes) o en formatos vectoriales como `otf`.
2. Texto impreso o mecanografiado: Texto generado por fuentes tipográficas digitales, impreso sobre papel u otro material y escaneado o fotografiado. Presenta una alta regularidad y precisión, pero puede contener errores de edición.
3. Texto manuscrito: Escrito manualmente por humanos, introduce variabilidad en el trazo, ligaduras, inclinación variable, diferentes estilos y variantes de cada letra...

○

Cada uno de estos tres tipos de textos sufre de distorsiones ligeramente diferentes, y por tanto las técnicas de OCR a emplear dentro de cada categoría son ligeramente diferentes. Mientras que la coincidencia de plantillas es una opción

⁷Explicados en detalle en la [Sección 2.5](#) del [Capítulo 2](#).

sencilla y extremadamente eficaz en texto digital, es una técnica prácticamente inútil con texto manuscrito general.

Definición (Tipos de textos según su entorno): Podemos distinguir los textos, especialmente los impresos y manuscritos, según la naturaleza de su ubicación:

1. Texto documento: Páginas escaneadas donde el fondo es relativamente homogéneo y el texto está bien alineado.
2. Texto escena: Texto fortuito en el mundo real, con fondos complejos e impredecibles, pudiendo el texto presentar orientaciones o curvaturas cambiantes.

○

Finalmente, podemos clasificar los tipos textuales según su contenido semántico, lo que será clave en nuestro modelo.

Definición (Tipos de texto según su contenido): Dependiendo de si el texto porta o no estructura gramatical e información semántica coherente podemos clasificarlo en tres grupos:

1. Texto sintáctico: Con estructura gramatical y sentido semántico.
2. Texto semiestructurado: Sentido semántico escaso o nulo, pero con una estructura predecible y lógica, como fechas o matrículas de una misma serie.
3. Texto asintáctico: Texto sin estructura lógica, semántica ni gramatical. Son ejemplos identificadores de productos, texto aleatorio o contraseñas.

○

1.4. Modelo

Las diferentes características de cada clase de modelos motivan su uso en diferentes contextos: mientras que las técnicas clásicas funcionan excepcionalmente bien en texto digital e impreso (tipologías conocidas pero con posibles distorsiones), o TrOCR es extremadamente útil a la hora de transcripción de texto semiestructurado, incluso en diversas escenas.

Nuestro conjunto de datos es texto documento manuscrito y sintáctico, que, además de presentar numerosas abreviaturas y geometrías de líneas no convencionales, está plagado de notación matemática densa y anotaciones. Esto lo hace un problema de transcripción difícil incluso para humanos. Resulta esclarecedor para la elección de modelos, por tanto, tener en cuenta la manera

en que realiza la transcripción de este tipo de textos un humano. Similar a un proceso de descifrado, en ocasiones se abandona el orden lógico de lectura para emplear información del texto colindante: cuando se encuentra una palabra suficientemente difícil de descifrar, se emplea el contexto, el contenido de palabras anteriores y siguientes si son transcribibles, para informar al transcriptor en su decisión para la palabra faltante⁸. Este proceso, comúnmente empleado de forma aislada durante la lectura o transcripción de texto manuscrito, es constante en este texto debido a su dificultad.

Nos encontramos, entonces, ante un problema de transcripción en el que el texto no solo es sintáctico, sino que además resulta indispensable contar con el contexto para una correcta transcripción. Esta consideración descarta inmediatamente aquellos modelos que no tengan en cuenta la estructura del lenguaje (como el OCR clásico), así como aquellos cuya construcción del contexto no pueda ser procedural (no puedan recibir contexto de páginas o fragmentos ya transcritos para su siguiente transcripción), como TrOCR.

La última revolución en transcripción de textos manuscritos que reconoce y emplea el contexto, conociendo la estructura del lenguaje, es el uso de LLM multimodales. Dentro de este tipo de modelos, existen dos vertientes principales:

1. Modelos comerciales de «pesos cerrados»: Aquellos modelos (familias GPT, Claude, Gemini, etc.) cuyos pesos o arquitectura no son públicos. Puesto que reciben inversiones enormes, suelen ser los modelos con mejor rendimiento en prácticamente todas las tareas. Esto incluye OCR, por lo menos a nivel base, comparado con otros modelos.
2. Modelos de «pesos abiertos»: Aquellos modelos (familias Qwen, Llama, Pixtral, etc.) cuya arquitectura y pesos son públicos y es, por tanto, posible ejecutarlos localmente. Pese a que tienen un peor rendimiento en OCR, estos modelos tienen la ventaja de ser, en muchas ocasiones, más pequeños en memoria y requerir menos recursos para ejecutarse de lo que requeriría un *modelo frontera*.

Cada uno de estos tipos de modelos presentan una problemática diferente. Por un lado, los modelos privativos no pueden aplicarse en esta situación: por la particular situación legal de los documentos, las imágenes no pueden enviarse

⁸Resulta un problema similar a aquel con el que se entrenan los transformer *encoder-only*, como BERT. Véase la [Subsección 2.3.5](#) del [Capítulo 2](#).

a servidores externos, un requerimiento de estos modelos alojados en grandes centros de datos. Por otro lado, los modelos de pesos abiertos, aunque sí son ejecutables localmente y por tanto conllevan menos problemas, tienen un rendimiento considerablemente peor en tareas de transcripción.

Los modelos de pesos abiertos, sin embargo, poseen una característica clave: puesto que tenemos acceso completo e irrestricto a sus parámetros, podemos ajustarlos como queramos para mejorar su rendimiento en casi cualquier tarea partiendo de un conjunto de entrenamiento.

1.4.1. Objetivos

Nuestro objetivo es adaptar un gran modelo de lenguaje de pesos abiertos para la tarea de transcribir manuscritos de Grothendieck. Pretendemos que el modelo sea lo suficientemente general como para poder transcribir tanto textos con fórmulas matemáticas como sin ellas, como es el caso de su obra tardía de Lasserre. Para esto, emplearemos técnicas de ajuste fino de LLM cuantizados, en concreto LoRA y QLoRA, entrenando al modelo de forma *local*.

Para alcanzar estos objetivos, hemos diseñado una arquitectura de entrenamiento que prioriza tanto la privacidad de los datos como la eficiencia computacional. Dada la imposibilidad legal de emplear modelos comerciales privativos que requieran el envío de imágenes a servidores externos de terceros, nuestra propuesta se basa en la ejecución y entrenamiento de modelos de pesos abiertos de forma estrictamente local.

El desarrollo práctico de este trabajo se divide en dos grandes bloques: la generación del conjunto de datos y el ajuste fino del modelo.

1.4.2. El conjunto de datos

Ante la inexistencia de un conjunto de datos estandarizado de la obra de Grothendieck, hemos partido de un documento extenso previamente transcrito. Para transformar este documento en pares de datos útiles para el modelo, hemos desplegado un servidor Label Studio que nos permite asociar fragmentos de la imagen con su texto.

Con el fin de automatizar y robustecer este proceso, hemos desarrollado un módulo de *Python* propio denominado *cropgen*. Este sistema procesa las transcripciones línea a línea y agrupa automáticamente líneas adyacentes, asig-

nándoles su transcripción conjunta. Esta estrategia actúa como un aumento de datos natural, realizándose durante el propio ajuste fino un aumento de datos adicional.

Para garantizar el rigor de las métricas y evitar cualquier contaminación, la separación entre los conjuntos de entrenamiento y validación se realiza de forma estricta a nivel de página.

1.4.3. Entorno para el ajuste fino

El ajuste fino de modelos masivos es prohibitivo en términos de memoria gráfica empleando métodos tradicionales. Para hacerlo viable, el entrenamiento se orquesta en un servidor remoto (equipado con 128 GB de memoria RAM y dos tarjetas gráficas RTX A6000 con respectivos 48 GB de VRAM) operando bajo WSL.

Emplearemos técnicas de cuantización de modelos y adaptadores de bajo rango (QLoRA) gestionados a través de la librería Unsloth. Esto nos permite afinar modelos visuales avanzados (como los de la familia Qwen) modificando solo una fracción mínima de sus parámetros, acelerando drásticamente el entrenamiento sin comprometer el rendimiento en inferencia que requieren los manuscritos.

2.

El transformer

La arquitectura de los *grandes modelos de lenguaje* (LLM, por sus siglas en inglés *Large Language Models*) es el *transformer*. Propuesta originalmente en 2017 por un equipo de Google Brain [2], su impacto pronto la elevó del campo de la traducción, para lo que fue concebida, y la convirtió en una revolución en procesamiento de lenguaje natural. Su impacto pronto se expandiría a otros muy variados campos.

La componente principal del *transformer* es el *bloque de autoatención* (*self-attention block*), que tiene su origen en lo que, hasta su aparición, eran los modelos dominantes en procesamiento de lenguaje natural: las redes neuronales recurrentes de tipo secuencia a secuencia (*sequence-to-sequence RNN*).

Adaptar un LLM a nuestro problema exige, además de un conocimiento de estas arquitecturas, herramientas de optimización. Por ello, este capítulo expone también las técnicas de LoRA y QLoRA empleadas durante el entrenamiento, profundizando en el fundamento teórico que sustenta a la primera. Finalmente, se presentan los adaptadores visuales, la interfaz fundamental entre imágenes y modelos de lenguaje sobre la cual se cimienta el ajuste fino de este trabajo.

2.1. Introducción al NLP

El procesamiento del lenguaje natural (NLP, por sus siglas del inglés *Natural Language Processing*) mediante técnicas de aprendizaje profundo requiere un proceso previo de conversión entre secuencias textuales en datos procesables por redes neuronales.

Formalmente, sea $\mathcal{X}$ un conjunto de símbolos finito y discreto tal que $|\mathcal{X}| = N$ al que llamamos *vocabulario*. Una secuencia de texto se define entonces como una secuencia finita $T = x_1 x_2 \dots x_t \in \mathcal{X}^t$, donde nuestro espacio de secuencias, $\mathcal{X}^*$, se define como

$$\mathcal{X}^* = \bigcup_{t>0} \mathcal{X}^t.$$

El procesamiento de estas secuencias, en concreto la generación de una respuesta, a menudo involucra elaborar una predicción del siguiente elemento de esta secuencia. Esto se consigue infiriendo a partir de los datos una medida de probabilidad $\mathbb{P}$, de forma que la probabilidad de una secuencia cualquiera viene dada por

$$\mathbb{P}(x_1 x_2 \dots x_t) = \prod_{i=1}^t \mathbb{P}(x_i \mid x_1 \dots x_{i-1}).$$

Es precisamente el operador probabilidad condicionada el objetivo a recrear en NLP, pues nos permite generar elementos sintéticos de una secuencia o recuperar fragmentos de texto corrompidos.

Para poder procesar estas secuencias a menudo se emplean espacios vectoriales que sirven como proyección semántica de los símbolos en un espacio geométrico. Para ello se define un operador de embebimiento (a menudo una matriz o, en implementación, una simple tabla o diccionario) $E : \mathcal{X} \rightarrow \mathbb{R}^d$. Una secuencia, entonces, es transformable vectorialmente en una matriz mediante este operador,

$$X := \begin{pmatrix} E(x_1)^\top \\ E(x_2)^\top \\ \vdots \\ E(x_t)^\top \end{pmatrix} \in \mathcal{M}_{t \times d}.$$

El problema de definir la probabilidad del siguiente elemento en una secuencia se reduce entonces a obtener una medida de probabilidad discreta, $\hat{p}^T$, a partir de una secuencia $T = T_1 \dots T_t$, modulando los parámetros de una red neuronal concreta:

$$\mathcal{F}_\theta(T) = \hat{p}^T \in \mathbb{R}_{\geq 0}^N, \quad \sum \hat{p}_i^T = 1.$$

$$\text{tal que } \hat{p}_i^T \approx \mathbb{P}(T_t = x_i \mid x_j = T_j \ \forall 1 \leq j \leq t-1) \ \forall i = 1, \dots, |\mathcal{X}|$$

Nuestra intención a lo largo del procesamiento del lenguaje natural es, por tanto, escoger tanto una arquitectura $\mathcal{F}$ como una serie de parámetros θ específicos para esta arquitectura de forma que la distribución de probabilidad que predice nuestro modelo sea tan próxima como sea posible a la distribución real.

2.2. RNN

Las redes neuronales recurrentes (RNN, por sus siglas en inglés *Recurrent Neural Networks*) están específicamente diseñadas para procesar datos de tipo secuencial, como aquellos con dependencia temporal (sonido, vídeo), o un orden implícito (lenguaje, movimientos en un juego de mesa). La diferencia principal entre una red neuronal de tipo *feedforward* como un perceptrón multicapa y una RNN radica tanto en el orden del procesamiento como en qué activaciones se usan en cada momento. [23]

En una red neuronal *feedforward*, las activaciones viajan en un solo sentido; podemos ver la activación de la capa ℓ -ésima como una función de la capa $(\ell - 1)$ -ésima. Siendo $a^{[\ell]}$ el vector de activación de la capa ℓ -ésima, $n^{[\ell]}$ el número de neuronas, $z^{[\ell]}$ la transformación lineal que aplica esta capa a la entrada y $\sigma^{[\ell]}$ la función no lineal (por ejemplo ReLU o GeLU) que se compone con $z^{[\ell]}$, el diagrama de la red sería:

$$a^{[0]} \xrightarrow{\sigma^{[1]} \circ z^{[1]}} a^{[1]} \xrightarrow{\sigma^{[2]} \circ z^{[2]}} \dots \xrightarrow{\sigma^{[L-1]} \circ z^{[L-1]}} a^{[L-1]} \xrightarrow{\sigma^{[L]} \circ z^{[L]}} a^{[L]}$$

donde la información fluye de inicio a fin sin dejar *ningún rastro*. Procesar una cierta pieza de información en un paso concreto no influye de ninguna forma en cómo se procesa la siguiente.

Este comportamiento, por supuesto, es contrario a lo que supone procesar un elemento que forma parte de una secuencia, pues no permite tener en cuenta su *contexto*. Para procesar secuencias completas con una red convencional se necesita, por tanto, procesar la secuencia completa en una tanda, por lo que su arquitectura debe diseñarse en consecuencia. Cuando la longitud de las secuencias es desconocida, variable o demasiado grande, sin embargo, esta opción no es viable.

2.2.1. La idea general

El cambio principal en RNN respecto a redes convencionales está en cómo se disponen las conexiones entre neuronas. Se sustituye el grafo dirigido estratificado de los perceptrones multicapa por un pseudografo dirigido estratificado: cada capa está conectada, además de a la anterior y la posterior, también así misma. A esta conexión consigo misma, que permite a cada capa recibir información sobre su activación en un momento pasado, la denominaremos *bucle*. En presencia de este bucle, la salida de una capa deja de ser función única de aquello que ha recibido de la anterior y pasa a depender también de todo aquello que ha procesado con anterioridad. [24]

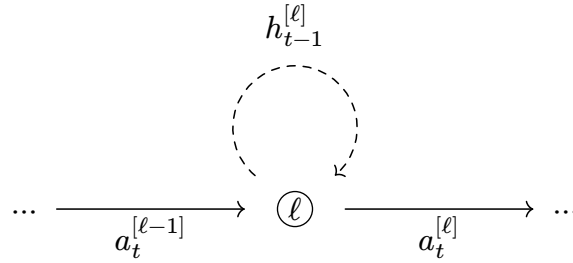


Figura 7. Diagrama de una neurona en una RNN procesando el elemento t -ésimo de una secuencia. En este, $h_{t-1}^{[l]}$ representa el *estado oculto* de la capa ℓ producido al procesar el elemento $(t - 1)$ -ésimo. Notemos que este valor debe almacenarse entre fragmentos para poder emplearse.

Una misma capa genera, entonces:

- Un lado un valor $h_t^{[l]}$, llamado *estado oculto* que no pasa a la siguiente capa.
- Su activación $a_t^{[l]}$, que es la información que recibe la siguiente neurona.

La generación de estos valores emplea dos funciones diferentes, $\sigma_h^{[l]}$ y $\sigma_a^{[l]}$. De igual forma que en una red *feedforward*, estas funciones se aplican elemento a elemento tras una transformación lineal de las entradas:

$$\mathbb{R}^{n_h^{[l]}} \ni h_t^{[l]} = \sigma_h^{[l]} \left(W_{h,a}^{[l]} a_t^{[l-1]} + W_{h,h}^{[l]} h_{t-1}^{[l]} + b_h^{[l]} \right),$$

donde $W_{h,a}^{[l]} \in \mathcal{M}_{n_h^{[l]} \times n_h^{[l-1]}}$ y $W_{h,h}^{[l]} \in \mathcal{M}_{n_h^{[l]} \times n_h^{[l]}}$. Tras esto, se calcula la activación

$$\mathbb{R}^{n_a^{[l]}} \ni a_t^{[l]} = \sigma_a^{[l]} \left(W_{a,h}^{[l]} h_t^{[l]} + b_a^{[l]} \right), \quad W_{a,h}^{[l]} \in \mathcal{M}_{n_a^{[l]} \times n_h^{[l]}}.$$

2.2.2. Retropropagación en RNN

El entrenamiento en redes neuronales se realiza empleando una variante de descenso de gradiente que involucra *retropropagación*. En retropropagación,

una aplicación de la regla de la cadena, se emplea la propiedad de las redes neuronales *feedforward* de que para calcular el error con respecto a una capa intermedia, basta con conocer sus parámetros y el error en capas posteriores.

El algoritmo de retropropagación fue originalmente concebido para redes neuronales convencionales, por lo que no puede aplicarse directamente a redes recurrentes. Para adaptarlo, naturalmente, basta con aplicar la regla de la cadena que, en este caso, es equivalente por la topología de la red a «desplegar» los estados pasados. Así, se transforma la RNN en una red feedforward [23] de mucho mayor tamaño.

Trazando estados anteriores como capas diferentes, se sustituye el problema de dependencia en cada capa por el de considerar una red neuronal enorme, de dimensiones mucho mayores a la RNN original.

Este «despliegue» de la red, sin embargo, no puede realizarse en toda su generalidad. Puesto que *a priori* no hay límite a la longitud de las secuencias que puede procesar una RNN sería necesario desplegar la red hasta que fuera de tamaño infinito. En consecuencia se debe tomar la decisión de en qué punto truncar, es decir, cuántos estados pasados considerar durante el entrenamiento como máximo: llamamos a este valor *tamaño de ventana* o *longitud de truncamiento*.

Esta sistemática posibilita realizar retropropagación en RNN, pero limita su aplicabilidad en secuencias de longitud mayor al tamaño de ventana: puesto que los parámetros no se han entrenado para tener en cuenta información tan pasada, el comportamiento puede degradarse en secuencias que superen el tamaño de ventana, especialmente cuando es por mucho [23].

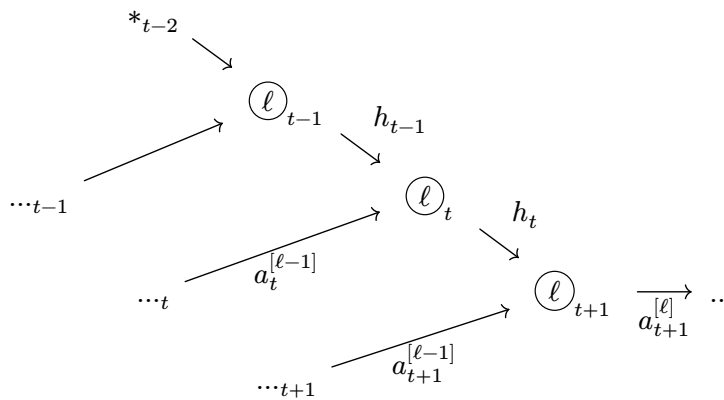


Figura 8. Diagrama del despliegue de una red neuronal recurrente para el cálculo de retropropagación.

Considerar tamaños de ventana demasiado grandes, por otro lado, tiende a generar el mismo tipo de problemas que las redes neuronales convencionales muy profundas: el problema del gradiente que desaparece o explota. En concreto en el caso de las RNN, la solución más aceptada fue el uso de una arquitectura alternativa, las *long short-term memory units* [23], que no trataremos.

2.2.3. Arquitectura seq2seq

Los modelos de RNN enfocados al lenguaje son modelos dobles, con dos partes diferenciadas dispuestas secuencialmente: codificador (*encoder*) y decodificador (*decoder*).

Definición (Codificador de RNN seq2seq): Llamamos *codificador* o *encoder* a un tipo de red neuronal recurrente cuya función es generar como salida un vector de embebimiento de una secuencia, que recibe como entrada dividida en fragmentos. Llamamos a la representación que genera de esta secuencia su *estado latente* [23]. Intuitivamente, este vector de embebimiento representa, resumido, el contenido de la secuencia completa, con todos sus elementos contextualizados.

Formalmente, la labor del codificador es tomar iterativamente una secuencia de entrada de la forma $x_1 \dots x_t \in \mathcal{X}^*$ y producir como salida una representación latente rica en significado V , que resume el contenido de la secuencia de entrada. $\bigcirc$

Definición (Decodificador de una RNN seq2seq): Llamamos *decodificador* o *decoder* a una red neuronal cuya función es, a partir del espacio latente obtenido del codificador, generar elemento a elemento la secuencia de salida [25].

Formalmente, la labor del decodificador es producir iterativamente una secuencia de salida $y_1 y_2 \dots y_\tau$ a partir de la representación latente V . $\bigcirc$

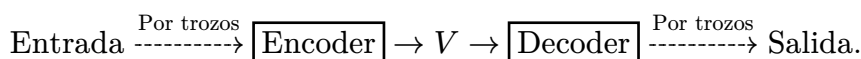


Figura 9. Diagrama general de una RNN de tipo seq2seq.

El codificador procesa la secuencia de entrada ya embebida elemento a elemento, produciendo tantos estados ocultos como elementos tiene la secuencia, t . Tras esto, el último estado oculto $h_t =: V$ se suministra como entrada al decodificador, que produce un estado latente que, tras desembeberse, se emplea como

primer elemento de la secuencia de salida⁹. Tras esto se le vuelve a suministrar el estado oculto que acaba de generar, perpetuando el proceso hasta que se alcanza un fragmento especial, <eos>, que denota el final de la generación.

En el contexto de los modelos de traducción, para lo que se propondría el transformer, la arquitectura era modular, consiguiéndose un espacio latente virtualmente igual para diferentes idiomas [25]. Esto se conseguía escogiendo el encoder del idioma original y el decoder del idioma al que se estaba traduciendo, entrenándose las parejas encoder-decoder para que estuvieran *alineadas* entre sí.

2.2.4. El mecanismo de atención en RNN

Un problema de esta arquitectura, que es además independiente de para cuántos pasos de tiempo se hayan entrenado sus bloques recurrentes individuales, reside en la dilución de la relevancia de los primeros fragmentos procesados: conforme más fragmentos de la entrada se procesan, más se enmascara progresivamente el significado de las primeras partes procesadas [23], [25], [26]. La base de la mayor parte de las soluciones que se plantean a este problema suele ser guardar, en lugar de solo el último, todos los estados ocultos generados, o guardarlos de forma inteligente y selectiva.

La solución *naïf* en este contexto (concatenar los estados ocultos de todos los momentos de la secuencia) no es viable: al ser variable la longitud de las secuencias, sería necesario escoger un tamaño máximo para poder procesar el vector concatenado (como dimensión de entrada del decodificador), lo que limitaría estructuralmente el funcionamiento del modelo a secuencias más cortas.

Otra posible solución es, a partir de todos los estados ocultos, tratar de extraer en cada paso de generación del decodificador una versión *refinada* de estos estados. Se pretende entonces que este refinamiento produzca un vector «relevante» para el elemento a generar, para lo que es necesario un nuevo sistema que medie entre el decoder y el conjunto de estados ocultos. Este intermediario deberá juzgar la relevancia en cada momento y producir este estado oculto refinado. Una arquitectura de estas características, la más empleada y de mayor impacto, se propone en [25], que presentó el *bloque de atención* en RNN.

⁹Realmente el objeto V , como se expone en la siguiente sección, no tiene por qué ser exactamente h_k (último estado oculto generado). Puede, en su lugar, generarse en función de varios de estos estados dependiendo de la arquitectura concreta empleada.

La labor del bloque de atención comienza cuando el codificador ha terminado su tarea, pues parte de todos los estados ocultos generados por este. Antes de generar cada nuevo elemento de la secuencia, el decoder envía su estado oculto al bloque de atención, un vector que llamamos *query*. La labor del bloque de atención es entonces hallar un coeficiente de alineamiento semántico entre este vector y cada uno de los estados ocultos. Tras esto, se pondera cada estado oculto por su alineamiento y se combinan produciendo un vector de «contexto rico». Finalmente este vector se introduce en el decoder junto a su estado oculto anterior para generar el siguiente bloque.

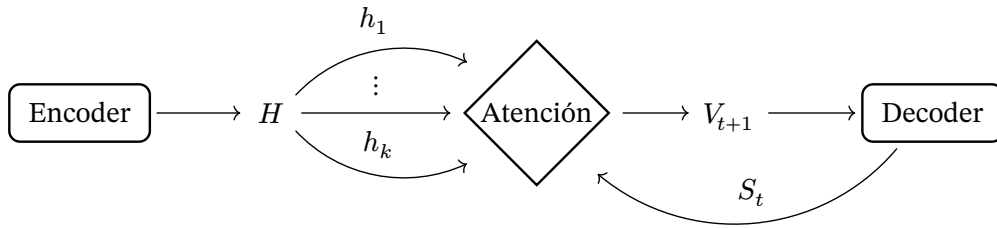


Figura 10. Diagrama del proceso de activación del decoder con un bloque de atención. El bloque de atención recibe todos los estados ocultos y el vector de *query* S_t , generando el estado oculto V_{t+1} para la activación del decoder.

Siendo S_t el estado oculto del decoder tras procesar el t -ésimo bloque de la secuencia y $H = (h_1, \dots, h_k)$ la secuencia de estados ocultos del encoder,

$$V_t = \sum_{i=1}^k \alpha_{t,i} \cdot H_i$$

donde $\alpha_{t,i}$ es una *puntuación de alineamiento* entre el i -ésimo vector de estado oculto H_i y siguiente bloque a generar, el t -ésimo. Se calcula aplicando softmax

$$\alpha_{t,i} = \frac{e^{\eta(S_{t-1}, H_i)}}{\sum_{\ell=1}^k e^{\eta(S_{t-1}, H_\ell)}}$$

a las puntuaciones según la función de score, η , definida habitualmente¹⁰ como

$$\eta(S_{t-1}, H) = S_{t-1}^\top W H$$

donde W es una matriz de pesos intermedios que sirve tanto para ponderar como para adaptar las dimensiones de S y H , pues no tienen por qué coincidir. Esto nos transforma la secuencia de puntuaciones dada por η en una medida de probabilidad discreta, soportada en la secuencia de estados ocultos, siendo

¹⁰Otra definición posible de esta función es empleando una red neuronal de tipo feedforward con una sola capa oculta, donde las neuronas asociadas a las entradas de cada uno de los dos vectores no están conectadas entre sí, $\eta(\vec{u}, \vec{w}) = \vec{v}^\top \cdot \tanh(W \cdot \vec{w} + U \cdot \vec{u})$

la salida V_t la esperanza de los estados ocultos tomando esta medida de probabilidad.

2.3. El transformer

La atención en RNN se emplea como una interfaz entre encoder y decoder que enriquece el contexto proporcionado al decoder en cada momento. En su lugar, [2] introdujo la noción de *autoatención* (*self-attention*), que cambia de forma fundamental cómo se emplea el bloque de atención: en lugar de como interfaz, pasa a ser el bloque principal de la arquitectura.

Análoga a una RNN seq2seq, el transformer original está constituido también de un *encoder* y un *decoder*. Pese a que ambos cumplen la misma función (respectivamente representar la entrada en un espacio latente y producir iterativamente la salida a partir de esta representación), el *encoder* y *decoder* del transformer apenas comparten similitud estructurales con sus homónimos en RNN de tipo seq2seq, ni siquiera siendo redes recurrentes [2], [23].

El mecanismo de atención aparece repetidamente tanto en la estructura de encoder como en la del decoder. De esta forma, los procesos de ambas componentes se benefician del mecanismo de autoatención.

La arquitectura del transformer parte de que se cuenta con una forma de reducir una secuencia de entrada a componentes más pequeños, llamadas *tókenes*. El conjunto de *tókenes* define el vocabulario del modelo según se ha expuesto en la introducción, y es el conjunto de elementos que puede procesar y producir el modelo. De forma estricta, un transformer genera una distribución de probabilidad soportada en su vocabulario, que se interpreta como la probabilidad de que el modelo genere cada token, y es un proceso externo el que se encarga de muestrear esta distribución y, en consecuencia, generar el texto.

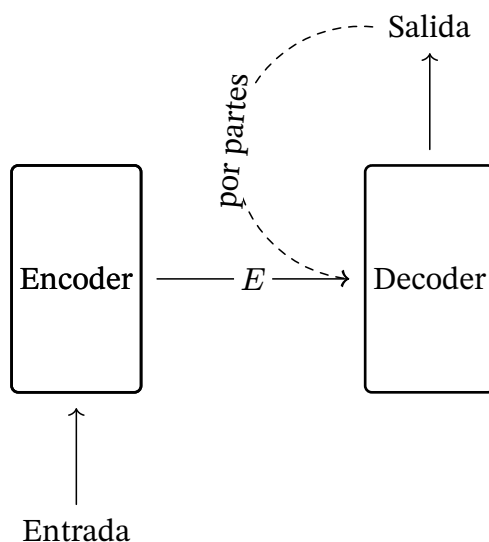


Figura 11. Diagrama de un transformer de tipo encoder-decoder.

2.3.1. Embebimiento y desembebimiento

La capacidad de procesamiento del transformer (como arquitectura general) es, como ejemplifica la variedad de ámbitos donde se aplica [18], [27], independiente de cómo se procese su entrada y su salida: es una arquitectura extremadamente versátil.

En el caso del texto, sin embargo, los procesos de entrada y salida son comunes a la mayor parte de LLM. La entrada sufre la siguiente secuencia de transformaciones antes de introducirse al modelo:

1. Se divide la entrada en tókenes.
2. Estos tókenes se embeben en un espacio vectorial de alta dimensión, a saber $\mathbb{R}^d$, mediante una matriz de embebimiento. De igual forma que con los RNN representan, de forma algo genérica, el significado de cada uno de estos tókenes sin tener en cuenta su contexto.
3. Se añaden marcadores de posición a todos los tókenes (embebimiento posicional). Estos embebimientos se generan a mano, a menudo siguiendo algún patrón progresivo, y sirven para que el modelo aprenda la estructura ordenada del texto.
4. Estos vectores ya están listos para comenzar a procesarse en el modelo.

La salida, por otro lado, requiere también de un proceso de transformación. El decodificador del transformer no produce como salida un nuevo elemento de

la secuencia ni una distribución de probabilidad, sino un nuevo vector rico en contexto por cada uno de los que se han introducido, y se sigue el siguiente proceso:

1. Se toma el último vector generado, correspondiente al último token procesado e interpretado como la predicción del modelo del siguiente token, y se desemebebe usando una matriz de desemebebeamiento de dimensiones $d \times N$, donde N es el tamaño del vocabulario. Esto resulta en un vector de *puntuaciones*. Estos valores, soportados sobre el vocabulario del modelo, se conocen como *lógits*.
2. Se aplica una función softmax con un parámetro fijo, llamado *temperatura*, lo que normaliza estas puntuaciones. Esto produce una distribución de probabilidad definida sobre los tókenes.
3. Se escoge un elemento de esta distribución siguiendo alguna regla.

Al token generado por el modelo para una secuencia de entrada lo llamamos *etiqueta* de esta entrada. Puesto que necesitamos concatenarla para continuar la generación, el modelo necesita también una forma de señalar que ha terminado el proceso de generación, lo que señala generando un token especial, a menudo `<eos>`.

Se explican ambos procesos con más detalle, así como la definición y utilidad de la temperatura en la función softmax, en el [Anexo B](#).

2.3.2. El mecanismo de autoatención

El bloque de autoatención, muy similar en concepto al bloque de atención de RNN, procesa en paralelo una cantidad de vectores que viene limitada por la ventana de contexto. Los vectores asociados a cada elemento de la secuencia, que ingresan a la red como vectores de embebimiento crudos, se enriquecen progresivamente con información contextual extraída del resto en el mecanismo de atención, hasta que abandonan la red.

Supongamos d la dimensión de los vectores de embebimiento, una secuencia de longitud n ya dispuesta, tras embeberse, en una matriz $X \in \mathcal{M}_{n \times d}$. Mediante tres productos matriciales se asocian a cada fila (muestra) tres nuevos vectores: *query*, *key* y *value*¹¹.

¹¹La dimensión de embedding de los vectores de *query* y *key* debe ser la misma en este mecanismo de autoatención, pues se realiza posteriormente entre ellos un producto escalar. La de los vectores *value*, por otro lado, no tiene por qué.

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

$$Q, K \in \mathcal{M}_{n \times d_k} \quad V \in \mathcal{M}_{n \times d_v}$$

Llamamos entonces vectores de *query* a las filas de Q , *key* a los de K y *value* a los de V . Sabiendo que nuestra intención es conseguir una métrica de relevancia de cada elemento de la secuencia para el resto, y generar un vector nuevo enriquecido en base a esto, la interpretación de los vectores es:

- *Query*: representan la «pregunta» que realiza cada elemento de la secuencia al resto. Son una representación semántica de qué tipo de contenido modifica el significado de ese elemento.
- *Key*: representan la «respuesta» que realiza cada elemento. Son una representación semántica de a qué tipo de elemento modifica este elemento de la secuencia.
- *Value*: representa la modificación que efectúa el elemento de la secuencia. Son una representación semántica del contenido del elemento de la secuencia cuando actúa como modificador.

En consecuencia, la relevancia del elemento j -ésimo para el i -ésimo se calcula como $\langle q_i, k_j \rangle$. Todos estos coeficientes se calculan como $QK^\top \in \mathcal{M}_{n \times n}$. Por motivos de estabilidad numérica, estos valores se «normalizan» dividiendo por la raíz cuadrada de su longitud,

$$R = \frac{1}{\sqrt{d_k}} QK^\top, \quad r_{i,j} = \frac{1}{\sqrt{d_k}} \cdot \langle q_i, k_j \rangle,$$

y se aplica a estos una función softmax (por filas), que actúa como función no lineal de este bloque, transformando puntuaciones y probabilidades

$$A = (a_{i,j})_{i,j=1}^n, \quad a_{i,j} = \frac{e^{r_{i,j}}}{\sum_{\ell=1}^n e^{r_{i,\ell}}}.$$

Finalmente, se genera la salida del vector de entrada i -ésimo ponderando los valores v_i por sus puntuaciones de alineamiento $a_{i,j}$,

$$Z = AV \in \mathcal{M}_{n \times d_v}, \quad z_i = \sum_{j=1}^n a_{i,j} v_j,$$

lo que, de nuevo, coincide con la esperanza de la distribución dada por estas probabilidades definida ahora sobre los vectores *value*.

2.3.2.1. Atención policéfala

Una problemática con el sistema de atención es que tiende a especializarse en un tipo de relación entre palabras. Por ejemplo, un bloque de atención puede especializarse en buscar adjetivos que modifiquen los sustantivos, o adverbios para los verbos, pero habitualmente no en demasiadas formas de modificación simultáneamente [2]. Se emplean entonces en el transformer simultáneamente varios mecanismos de atención (cabezas) en paralelo en un mismo bloque, lo llamaremos *atención policéfala*¹².

Así, cada cabeza de atención (cuyos pesos se inicializan aleatoria e independientemente del resto) puede especializarse en un tipo de búsqueda y enriquecimiento contextual. De esta forma, además, se evita que la atención de un elemento de la secuencia se concentre demasiado en sí mismo, proceso que sí ocurre con bloques de atención con una única cabeza.

Cada bloque de atención produce un resultado Z_k independiente del resto, que se ha calculado con sus matrices Q, K, V independientes.

En un bloque de atención policéfala se producen, por cada vector de entrada, tantos vectores de salida como cabezas haya. En consecuencia, estos vectores deben condensarse para no aumentar la dimensionalidad de entrada del siguiente bloque exponencialmente. Para ello, se concatenan las matrices de salida Z_k , cuyas columnas son los vectores de salida de cada bloque:

$$Z_{\text{poli}} = (Z_1 \ Z_2 \ \dots \ Z_p) \in \mathcal{M}_{n \times p d_v}.$$

Para adaptar las dimensiones de esta matriz a las de entrada de la siguiente capa, se multiplica por una matriz $W_O \in \mathcal{M}_{p d_v \times d_v}$ cuyos parámetros son también aprendibles.

$$Z_O = Z_{\text{poli}} W_O \in \mathcal{M}_{n \times d_v}$$

¹²En la literatura castellano se emplean, entre otras, «atención multicabezal», «atención de varias cabezas» o incluso «cabeza de multiatención», más similar en pronunciación pero menos fiel al término original inglés «*multiheaded attention*».

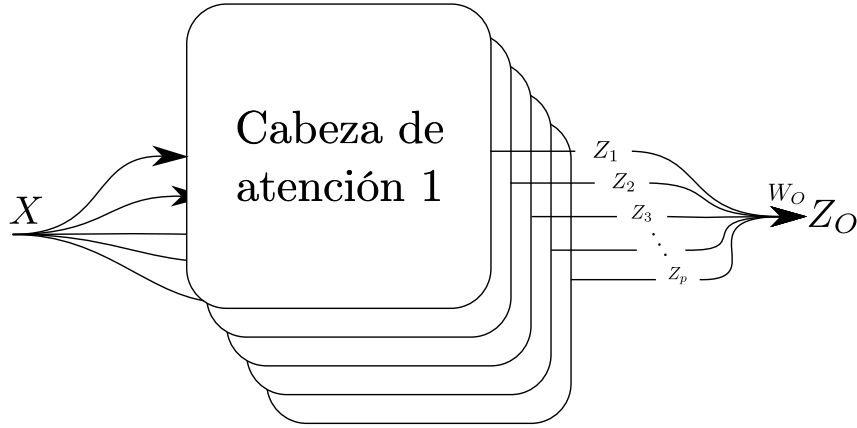


Figura 12. Bloque de atención policéfala. La salida de todas las cabezas de atención se concatena y combina mediante W_O en una sola matriz de salida de las dimensiones correctas.

Las cabezas de atención contribuyen la mayor parte de los parámetros de nuestro modelo, siendo además un componente esencial para el buen funcionamiento de los modelos. El problema principal que plantean es la latencia que añaden durante inferencia [28]. Para intentar reducir la cantidad de parámetros que contienen estos bloques de atención existen diferentes alternativas.

Definición (Atención multi-query): En esta alternativa a las cabezas de atención, todas las cabezas de atención comparten matrices W_k y W_v , de forma que los vectores de *key* y *value* son comunes entre todas ellas (lo que reduce la carga computacional), variando entre ellas únicamente W_q . [28]

Esto hace que, mientras que las matrices W_q se entrenan independientemente entre cabezas, las matrices W_k y W_v se entrenan para minimizar simultáneamente la pérdida de todas las cabezas. $\circ$

Siendo p el número de cabezas de atención, T el tamaño de la ventana de contexto, d_v la dimensión de los vectores de *key* y *value*, y d_q la dimensión de los vectores de *query*, esta sistemática reduce el número de parámetros en un bloque de atención de $p(2Td_v + Td_q)$ a $2Td_v + pTd_q$. Esta simplificación, pese a ser extremadamente efectiva en reducir el número de parámetros, conlleva un empeoramiento considerable de los resultados. [28]

Existe, sin embargo, un espectro entre la atención multi-query y la atención original, que podemos alcanzar agrupando cabezas de atención en bloques.

Definición (Atención de query agrupada): En esta alternativa, las diferentes cabezas de atención se dividen en grupos equinumerosos y se asocia a cada uno

de ellos sendas matrices W_k y W_q , teniendo cada cabeza individual su matriz W_q independiente. [28] ○

Así, siendo ν el número de grupos, esta simplificación reduce el número de parámetros del original $p(2Td_v + Td_q)$ a $\nu 2Td_v + pTd_q$. Esta opción se presenta como un punto intermedio variable tanto en eficiencia como en resultados entre multi-query attention y el bloque de atención completo. [28]

En el reporte técnico del DeepSeek-V2 se plantea un método más complejo donde se comparte parte de la información de las matrices, afectando mínimamente al resultado, empleando una representación latente de las matrices W_q, W_v, W_k común a todas las cabezas, y aprendiendo en su lugar matrices de desembebiendo. La técnica se conoce como *Multiheaded Latent Attention* (MLA). [29]

2.3.2.2. El bloque de autoatención del decoder

El bloque de atención antes descrito se corresponde con el empleado en el codificador. Sin embargo, puesto que la labor de este último no coincide con la del decodificador tampoco lo hacen sus bloques de atención. El decoder funciona de igual forma sobre todos los elementos de su entrada: a partir de k elementos trata de generar k predicciones, cada una correspondiéndose con el siguiente elemento de la secuencia dada por concatenar la entrada hasta su elemento.

Si la red tuviera, entonces, acceso a todos los elementos de la secuencia, la estructura óptima es equivalente a una matriz diagonal superior, a atender de forma desorbitada siempre al siguiente elemento en la secuencia. Naturalmente, este comportamiento falla tan pronto como «no existe» el siguiente elemento de la secuencia: fuera del preentrenamiento¹³, donde el modelo recibe grandes corpus textuales, es labor del modelo generar la siguiente palabra únicamente en base a las anteriores. En consecuencia, si únicamente ha aprendido a producir como resultado el siguiente token al que atendía, no será capaz de reproducirlo.

Para evitar esta convergencia, se fuerza al mecanismo de atención a actuar solamente de forma retrospectiva, impidiendo recibir información de elementos siguientes de la secuencia. Esto se consigue enmascarando la matriz de produc-

¹³Véase el [Anexo B](#) para detalles sobre este proceso.

tos escalares $R = QK^T$ de forma que, tras aplicar softmax la matriz resultante sea triangular inferior.

Así, en la fila i -ésima de AV , solamente hay presente información de las filas 1 hasta i de la matriz V . [3]

2.3.2.3. El bloque de atención cruzada

Además del bloque de autoatención, en el decoder hay presente otro bloque específico¹⁴, el *bloque de atención cruzada* [2]. Mientras que en un bloque de autoatención se calculan las tres matrices K , Q y V a partir de las entradas, en el bloque de atención cruzada se calculan los valores de Q a partir de las entradas del anterior bloque del decoder, pero las matrices K y V se generan a partir de la salida del encoder.

De esta forma se combina la información que llega del decoder y la extraída del encoder, de forma que se combinan. En esencia, se permite que cada elemento de la secuencia vea su significado modificado por el significado de la secuencia de entrada completa (la salida del encoder).

2.3.3. El encoder

El encoder es un conjunto de *encoder blocks* apilados, donde la información fluye de unos a otros de forma secuencial [2]. El número de estos bloques, al igual que muchos otros parámetros que diferencian estructuralmente los bloques entre sí, varían entre diferentes modelos concretos con esta arquitectura.

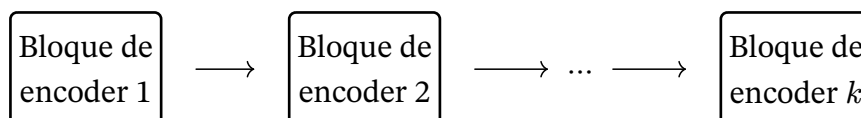


Figura 13. Diagrama del encoder como composición de bloques de encoder.

En el interior de un bloque de encoder podemos encontrar dos subbloques: uno de atención y una red feedforward, entre los cuales se encuentra una interfaz de adición residual y una normalización por lotes (*Layer normalization* o *Layer norm*).

¹⁴Por lo menos el presente en el *transformer* original.

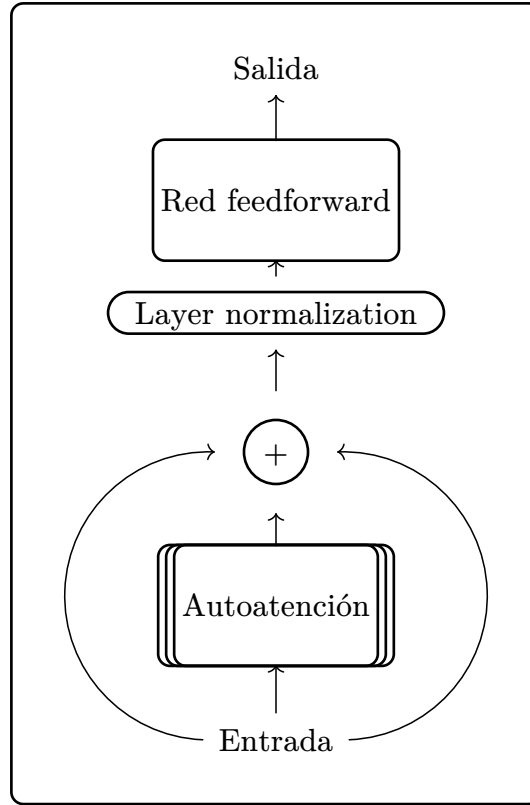


Figura 14. Diagrama de un bloque de encoder donde, mediante « $\oplus$ » estamos simbolizando la adición residual, en la que el resultado de la autoatención y su entrada, apropiadamente denotada «Entrada», se suman. Notemos que el bloque de atención puede tener múltiples cabezas. [2], [3]

Definición (Layer normalization): La *layer normalization* es una técnica de normalización diseñada para estabilizar el entrenamiento de redes neuronales profundas. Consiste en estandarizar las salidas de una capa de forma independiente para cada muestra, empleando los momentos estadísticos de las activaciones de esa misma capa. [30] Siendo $y = (y_1, \dots, y_d)$ el vector de activaciones de una capa con d dimensiones para una única entrada x , el resultado de aplicar *layer normalization* para cada componente i se define como: $\tilde{y}_i = \frac{y_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$, donde, μ y σ son, respectivamente, la media y la varianza muestrales calculadas sobre los d elementos del vector y .

El escalar $\varepsilon > 0$ es una constante añadida para garantizar la estabilidad numérica evitando divisiones por cero. Habitualmente, el vector normalizado se somete después a una transformación afín vectorizada $a_i \tilde{y}_i + b_i$, donde $a, b \in \mathbb{R}^d$ son parámetros entrenables del modelo. $\circ$

El resultado del encoder son dos matrices, $K_x, V_x \in \mathcal{M}_{n \times d_v}$, que se usarán en bloque de atención cruzada del decoder [2], [3].

2.3.4. El decoder

Al igual que el codificador, el decodificador es también una concatenación de bloques de decodificador [2].

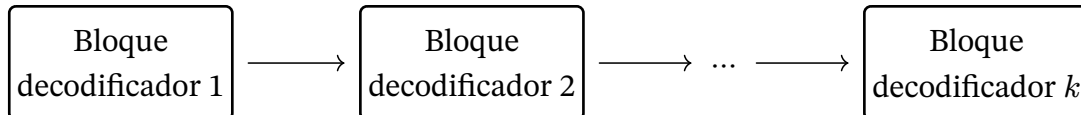


Figura 15. Diagrama del decodificador como composición de bloques decodificantes.

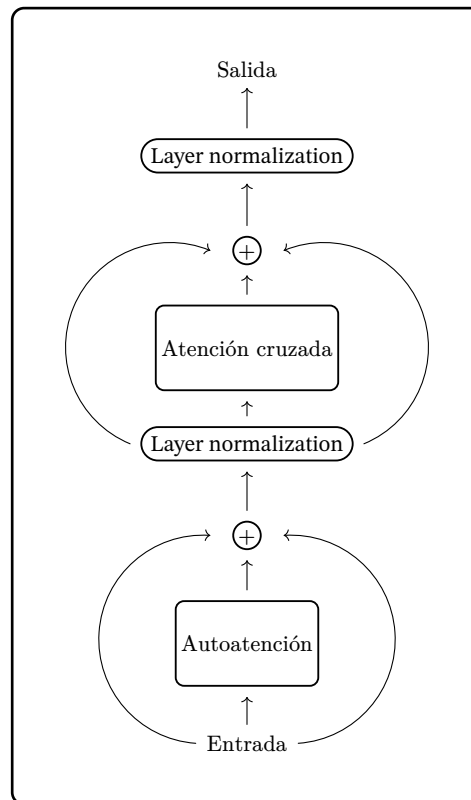


Figura 16. Diagrama del decodificador en un transformer. [2], [3]

Se dice que el decoder es un modelo «autorregresivo» porque recicla su salida de forma iterativa para generar nuevos resultados. Una vez ha procesado la entrada concreta, (en un inicio un token especial que marca el inicio), se obtiene como resultado una distribución de probabilidad sobre los tókenes. Esta distribución se muestrea para generar el siguiente token.

Este nuevo token sintético se introduce como siguiente a la entrada del decoder, concatenado a la ya presente, y se procesa como el resto de elementos para obtener una nueva distribución de probabilidad: la del siguiente elemento. Este proceso se repite hasta que se alcanza un token específico que denota el final del proceso de decodificación.

2.3.5. Arquitecturas encoder-only y decoder-only

Definición (Arquitectura encoder-decoder o seq2seq): Llamamos *arquitectura transformer encoder-decoder*, *transformer completo*, *transformer secuencia a secuencia* o, simplemente *transformer* a la arquitectura original presentada en *Attention is all you need*. [2] ○

La arquitectura original del transformer, sin embargo, ha sufrido modificaciones extensas con el paso del tiempo. La más importante de ellas es la escisión de sus dos bloques principales (codificador y decodificador) en dos arquitecturas independientes.

Definición (Arquitectura encoder-only): Llamamos *arquitectura encoder-only* a aquella red neuronal cuya arquitectura sea equivalente a la del codificador de un transformer, produciendo una representación latente V para una secuencia de entrada.

También se conoce a este tipo de modelos como *transformers autocodificantes* (*autoencoding transformers*). ○

Este tipo de redes neuronales se emplean habitualmente para obtener una representación «semántica» del contenido de una secuencia, por ejemplo para búsquedas en una base de datos a partir de una pregunta en lenguaje natural en un sistema RAG.

Habitualmente se entrenan en tareas de recuperación de contenido faltante: se «corrompe» parte de una secuencia con contenido aleatorio o un token especial, <mask>, y se calcula la pérdida como la diferencia entre la representación latente que el modelo genera y la del token faltante [31]. Puesto que no se cuenta con un decoder, sin embargo, es necesaria una matriz de desembebimiento que, a partir de las salidas de nuestro modelo, permita recuperar los tokens.

Definición (Arquitectura decoder-only): Llamamos *arquitectura decoder-only* a aquella red neuronal cuya arquitectura es similar a la del decodificador de un transformer sin tener presente un codificador. Así, esta red toma como entrada una secuencia de elementos y produce otra como salida de forma autorregresiva. A diferencia del decodificador presente en el transformer, sin embargo (puesto que no se cuenta con codificador) no hay bloques de atención cruzada, habiendo únicamente bloques de atención enmascarada.

Las redes neuronales que siguen este tipo de arquitecturas se conocen como *transformers autorregresivos* (*transformer autoregressors*). [32] ○

Mientras que en la arquitectura original había una diferenciación clara entre la «entrada» al modelo (la secuencia que se pasaba al encoder) y su respuesta (los tokens generados por el decoder), en los transformers autorregresivos esto se pierde. Puesto que no se cuenta con un encoder, la secuencia de entrada íntegra se embebe en un espacio latente token a token empleando una matriz de embebimiento. Tras esto, los tokens generados por el modelo se embeben con esta misma matriz y se concatenan a los vectores de la entrada, produciéndose un solo bloque de contenido para el modelo.

Es habitual, por tanto, construir en un LLM una diferenciación –más sugerida que impuesta– entre la entrada al modelo y su generación mediante la adición de tokens especiales¹⁵.

Una diferencia funcional entre las arquitecturas encoder-only y decoder-only es que, mientras que en las primeras todos los tokens pueden «atender» al resto (bidireccional), en las segundas este mecanismo solamente funciona hacia atrás: un token t solamente puede prestar atención a los tokens $1, 2, \dots, t - 1$ y, en ocasiones, también a sí mismo.

2.4. Entrenamiento por ajuste fino

Definición (Ajuste fino o *fine tuning*): Llamamos *ajuste fino* o *fine tuning* al proceso de tomar un modelo ya entrenado (habitualmente para una tarea

¹⁵La diferenciación entre roles es útil, por ejemplo, durante el *safety training* o para que el modelo aprenda la dinámica pregunta-respuesta propia de un *chatbot* asistencial. Véase el [Anexo B](#), sobre el entrenamiento de LLMs para este proceso y la tokenización de una conversación mediante tokens especiales.

general) y reentrenarlo con un nuevo conjunto de datos que, potencialmente, no estaban presentes en su entrenamiento. Mediante esto se pretende que el modelo adquiriera la capacidad de realizar una nueva tarea. [33] ○

El *fine tuning* se basa en un concepto clave que ha tomado especial importancia en el panorama de los grandes modelos de lenguaje y visión, entrenables para tareas muy generales: el aprendizaje por transferencia.

Definición (Aprendizaje por transferencia): En el contexto del *fine-tuning* (no necesariamente de LLM), se conoce como *aprendizaje por transferencia* (*transfer learning*) al proceso mediante el que un modelo diseñado para una tarea general es capaz de transferir parte de sus conocimientos sobre la lógica del mundo y su funcionamiento a una nueva tarea, más específica, para la que se le entrena mediante un ajuste fino. [34] ○

Se presupone, por tanto, que el modelo es capaz de: retener (al menos parcialmente) el conocimiento del mundo que posee capaz, de aplicar este conocimiento a la nueva tarea para la que es entrenado, y, además, de aprender mejor o más rápido del conjunto de ajuste fino.

Se asume entonces que la información añadida en forma de ajustes a los parámetros (la necesaria para mejorar en la tarea), puede de alguna manera *coexistir* con la información anterior sin destruirla, complementándola. Este concepto se exploraría en más detalle, entre otros, en [35].

El equilibrio, sin embargo, en esta adición de información durante el ajuste fino tiene sus riesgos.

Definición (Olvido catastrófico): Llamamos *olvido catastrófico* al proceso mediante el que un modelo pierde sus capacidades originales tras sufrir ajuste fino para una tarea concreta [36]. ○

Por ejemplo, un modelo ajustado para realizar razonamientos matemáticos entrenado con una instrucción (*prompt*) concreta puede comenzar a producir resultados incoherentes para preguntas con una instrucción diferente, o en temas no matemáticos.

Este problema no es exclusivo de LLM, sino que se trata de un problema común a muy diversas arquitecturas de redes neuronales y es conocido desde hace décadas [37].

2.4.1. Proceso general de ajuste fino

El proceso de ajuste fino requiere, igual que el proceso de entrenamiento, un conjunto de muestras X con sus respectivas etiquetas Y . En el caso de LLM, ambos son conjuntos de tókenes (ya sean tókenes de texto, de imagen, etc.).

Estos datos, habitualmente divididos en lotes (o *batches*) se emplean como si fueran el conjunto de entrenamiento completo para que calcular los gradientes de la función de coste no sea tan exigente computacionalmente. En el proceso de *fine-tuning* completo se emplean los datos para *actualizar* los valores de nuestras matrices de pesos. A partir de una cierta matriz de transición W (por ejemplo W_q , W_v , W_k o W_o), la operación a realizar durante la inferencia sería

$$\hat{Y} = X^{\{k\}} W$$

y es a partir de este valor y las etiquetas reales, Y , que se calcula un cierto error de nuestro modelo para el conjunto de entrenamiento suministrado, que denotaremos $J(\hat{Y}, Y)$ [2], [35]. En el caso de los LLM, esta función de error es la entropía cruzada, calculada entre la distribución de probabilidad que nos ofrece y la distribución de probabilidad poblacional que resulta de nuestro conjunto de datos:

Definición (Entropía cruzada): Se define la entropía cruzada entre dos distribuciones de probabilidad con funciones de masa respectivas f y g como

$$\mathcal{H}(f, g) = - \sum_{x \in \mathcal{X}} f(x) \log(g).$$

○

En nuestro caso una de las distribuciones de probabilidad vendrá condicionada (la que genera el modelo), por tanto $f(x) := f_t(x \mid x_1 \dots x_{t-1})$, y la otra será la distribución puntual $g(x) := g_t(x_{t+1}) = 1$, $g_t(y) = 0 \forall y \neq x_{t+1}$. Este error se calcula para todas las posiciones de nuestra secuencia, resultando en la siguiente expresión:

$$\mathcal{L} = \sum_{t=u}^T \mathcal{H}(f_t(\cdot \mid x_0 \dots x_{t-1}), g_t(\cdot)).$$

La función principal de la función de coste es darnos una métrica de cuánto error se está cometiendo de forma que sea posible buscar sus mínimos. En consecuencia, la función de error debe ser una fiel representación del «criterio humano», pues de ella se extraen las reglas de actualización de los parámetros.

Durante el proceso general de *fine-tuning*, sin embargo, la labor cambia, al menos en general, a la de *generar contenido en respuesta a un estímulo* (como responder una pregunta de un usuario o realizar una búsqueda a internet). En consecuencia, no queremos que el modelo aprenda a generar todas las partes de una conversación sino únicamente aquellas en las que es «su turno» para responder. Así, pese a que la función de error se calcula completa, los gradientes de esta función de error para actualizar sus parámetros se calculan únicamente con respecto a los tókenes del modelo¹⁶.

La información sobre los gradientes se suministra tras esto a un *optimizador*, cuya labor es actualizar los parámetros del modelo en consecuencia.

Definición (Optimizador): En el contexto del aprendizaje profundo, llamamos *optimizador* a una regla de actualización recursiva de los parámetros de un modelo para minimizar una cierta función de coste.

Un ejemplo es descenso de gradiente, el más sencillo de todos, y otro es Adam (*Adaptive Moment Estimation*), que actualiza los parámetros en base a una estimación de la media y de los cuadrados de los gradientes de la función de costes [38]. ○

Denotaremos Δ_W^t al valor del optimizador de los valores de W en el tiempo t , que posiblemente depende de los valores anteriores del gradiente, $dW^{t-1}, dW^{t-2}, \dots$, y de una serie de parámetros θ fijados,

$$\Delta_W^t := \Delta^t(\theta; dW^t, dW^{t-1}, \dots, dW^0)$$

calculándose entonces el nuevo valor de los parámetros como [35]

$$W^{t+1} = W^t - \Delta_W^t$$

Una vez calculado este nuevo valor, se recalculan los gradientes para el siguiente lote (el $X^{\{t\}}$) empleando el nuevo valor para W recién calculado, W^t . De esta forma, en un proceso análogo a descenso de gradiente con una elección de gradiente algo más informada, se buscan mínimos locales en la función de error.

Una vez ha terminado el proceso de entrenamiento, podemos expresar el valor final de W tras N actualizaciones de su valor como

$$W^N = W^0 + \underbrace{\sum_{t=0}^N \Delta_W^t}_{:= \Delta W}$$

¹⁶Véase la [Sección B.3](#) del [Anexo B](#).

Notemos que, en general, el rango de ΔW no está restringido de ninguna forma.

2.4.2. Adaptación de bajo rango (LoRA)

El problema con el ajuste fino «estándar» radica en que el conjunto de datos con el que se hace el *fine-tuning* suele ser mucho más pequeño que el original. En consecuencia, es fácil que nuestro modelo se sobreadapte a nuestros datos (pues los LLM tienen miles de millones de parámetros), perdiendo sus capacidades anteriores. A esto se suma que ajustar todos los parámetros de un modelo es extremadamente costoso, tanto en tiempo como en precio de cálculo [35].

Para tratar de solventar estas dos problemáticas, la comunidad ha diseñado ya varios esquemas de *ajuste fino eficiente en parámetros* (PEFT, por sus siglas en inglés *Parameter-Efficient Fine-Tuning*). De entre ellos, el más empleado es LoRA (*Low Rank Adaptation*).

El funcionamiento de este proceso es sencillo de interpretar: en lugar de calcularse el gradiente con respecto a todas las matrices de pesos completas, $W \in \mathbb{R}^{d_1 \times d_2}$, se calculan para una matriz de menor rango, llamada *adaptador LoRA* [35].

Sea $W \in \mathcal{M}_{d_1 \times d_2}$ una matriz de parámetros que queremos ajustar usando LoRA. Para construir el adaptador, se inicializan dos matrices, $A^0 \in \mathbb{R}^{r \times d_2}$ y $B^0 \in \mathbb{R}^{d_1 \times r}$, con $r \ll \min(d_1, d_2)$, de forma que $B^0 A^0 = \mathbf{0}_{d_1 \times d_2}$ ¹⁷. Se define entonces una nueva matriz $\widehat{W}^t = W + \frac{\alpha}{r} B^t A^t$, siendo α un hiperparámetro. En lugar de calcular el gradiente con respecto a la matriz W , se calcula con respecto a las dos nuevas matrices A^t y B^t , empleando como matriz de parámetros del modelo $\widehat{W}^t$.

$$A^{t+1} = A^t - \Delta_A^t$$

$$B^{t+1} = B^t - \Delta_B^t$$

Podemos visualizar el producto $B^t A^t$ como una aproximación de precisión variable de la matriz Δ_W^t . Escoger un menor r nos lleva a un ajuste fino más «suave», que añade poca información, y un rango mayor nos lleva a uno más «agresivo», más similar a un ajuste fino completo.

¹⁷En el artículo original se propone A^0 generada a partir de ruido gaussiano y B^0 idénticamente nula, sin embargo existen inicializaciones alternativas.

Una forma habitual de medir la «intensidad relativa» de este *fine-tuning* es usando un porcentaje:

$$\text{LoRA}\% = \frac{\# \text{ parámetros en LoRA}}{\# \text{ parámetros en el modelo completo}}.$$

Se suelen tomar rangos r tales que este porcentaje se sitúe entre un 0.05% y un 3% [35].

2.4.2.1. Aprendizaje en un subespacio

La base teórica de LoRA es que el aprendizaje para una tarea específica puede reducirse a un subespacio muy pequeño del espacio de parámetros [39]. Para una tarea concreta, la información añadida no requiere toda la dimensionalidad que existe.

Es posible explicar parte de este fenómeno teniendo en cuenta que las representaciones aprendidas de nuestro modelo (las diferentes representaciones latentes a diferentes profundidades) viven en una variedad de dimensión mucho menor a la del espacio ambiente en la que se encuentran [40], [41]. Esto se ve reflejado en que, realizando una descomposición en valores singulares de las matrices de atención, los valores ordenados siguen aproximadamente una exponencial [42], lo que hace que la acción de esta matriz sobre los datos de entrada esté dominada por unas pocas direcciones de máxima variación¹⁸. Así, mientras modificar todos los pesos permite cambiar (de forma posiblemente destructiva) esta variedad, restringir el rango distribuye de forma más homogénea las modificaciones entre todos los parámetros, haciendo el impacto sobre estas representaciones más sutil y aumentando las probabilidades de permanecer en la variedad donde el modelo funciona correctamente.

Esto puede reforzarse aún más implementando técnicas de regularización en los adaptadores LoRA añadidos, que penalicen modificaciones muy grandes a las matrices de parámetros.

2.4.3. Cuantización, QLoRA

Definición (Precisión de coma flotante): La representación de números reales en memoria emplea habitualmente el estándar de coma flotante. Este

¹⁸Este comportamiento no es exclusivo de LLM, pues ocurre en otro tipo de arquitecturas de deep learning. [43]

define diferentes *precisiones*, que varían en el número de bits que se dedican a cada número en memoria. Las más usadas, de mayor a menor precisión y espacio ocupado, son 64, 32 y 16 bits. ○

Emplear LoRA acelera el proceso de entrenamiento por necesitar calcularse menos gradientes, pero no elimina el problema del uso de memoria: para calcular estos gradientes es necesario tener acceso a todas las matrices de pesos del modelo, lo que puede ser prohibitivo en el caso de que sea un modelo suficientemente grande (entrenar un LLaMA 65B requiere 780 gigabytes de memoria en GPU [44]). La solución para esto es realizar un proceso de *cuantización*, y emplear un algoritmo de LoRA adaptado a esto. El más empleado en este contexto es el presentado en [44].

Definición (Cuantización): Llamamos *cuantización* (a menudo referido a los pesos de un modelo) al proceso de transformar un conjunto de datos pertenecientes de un espacio a otro que contiene menos información.

En este proceso a menudo se pierde precisión, pero, si los datos deben guardarse en memoria, emplear un espacio menos rico en información permite reducir su tamaño. [45] ○

El proceso de cuantización puede representarse, igual que LoRA con los diferentes rangos, como un espectro. Dependiendo de la cantidad de precisión¹⁹ que se esté dispuesto a perder en el proceso.

Definición (Cuantización entera): Llamamos *cuantización entera* (*integer quantization*) a un proceso de cuantización en el que se aplica una transformación afín a nuestros datos y se truncan a un intervalo dado.

Siendo X nuestro conjunto de datos y $q_{\min}, q_{\max} \in \mathbb{Z}$ los límites del espacio cuantizado (tendremos $q_{\max} - q_{\min}$ valores diferentes en el espacio cuantizado), se toma el conjunto de datos dado por:

$$q(x) = \text{trunc}_{q_{\min}}^{q_{\max}} \left(\left\lfloor \frac{x}{S} \right\rfloor + Z \right)$$

donde mediante $\lfloor x \rfloor$ denotamos el redondeo, y

$$\text{trunc}_a^b : \mathbb{Z} \rightarrow \mathbb{Z}, \quad \text{trunc}_a^b(x) = \min(\max(a, x), b).$$

¹⁹En ocasiones también se pierde velocidad de entrenamiento e inferencia.

El valor de S se escoge de forma que un rango determinado, por ejemplo $[x_{\min}, x_{\max}] \subset \mathbb{R}$, se transforme en nuestro espacio completo $[q_{\min}, q_{\max}] \subset \mathbb{Z}$, y Z se escoge de forma que el cero no sufra errores de redondeo:

$$S = \frac{x_{\max} - x_{\min}}{q_{\max} - q_{\min}}, \quad Z = \left\lceil -\frac{\alpha}{S} \right\rceil + q_{\min}.$$

Además de los valores $q(x)$ de un conjunto de pesos X se guardan también los valores de S y de Z para poder recuperar los datos originales. [45] ○

Además de los datos, se guardan también los valores de Z y S para poder realizar el proceso de *descuantización*. Este proceso resulta relevante puesto que los datos cuantizados no pueden usarse directamente durante inferencia o el entrenamiento por estar en la escala incorrecta, y requieren primero recuperar su escala original.

Definición (Descuantización): Llamamos *descuantización* de un modelo (o una parte de él) al proceso de obtener una aproximación de sus pesos $\hat{x}$ a partir de una cuantización concreta.

Llamamos *error de cuantización* de un peso x al error entre el peso original y su versión descuantizada, $|x - \hat{x}(q(x))|$. ○

Durante el proceso de inferencia y entrenamiento es necesario emplear un proceso de descuatización para hacer cálculos con los pesos. Así, un modelo cuantizado, pese a ocupar menos en memoria, requiere un proceso más complejo (de cuantización y descuantización constante) para poder emplearse.

El que los datos se encuentren en una precisión menor, sin embargo, permite una «pseudodescuantización» a un espacio de menor información al original pero que mantiene la misma escala. Así, las operaciones con los datos requieren simultáneamente menos memoria y capacidad de cálculo. Suele emplearse BF16, un formato de punto flotante que requiere un cuarto de la memoria y capacidad de cómputo para operar que el formato F64, en el que no indagaremos.

Decimos que la cuantización entera es *uniforme* en tanto que la preimagen de cada valor en el espacio cuantizado son intervalos de igual longitud (salvo quizás sus valores mínimo y máximo).

Este tipo de cuantizaciones, sin embargo, incurren en pérdidas de información innecesarias en datos que no están distribuidos uniformemente en el

intervalo $[x_{\min}, x_{\max}]$. Para obtener métodos de cuantización óptimos (o al menos mejores), es necesario emplear la estructura estadística del conjunto de pesos del modelo, que habitualmente se considera sigue una distribución normal.

Definición (Normal float 4): El proceso de cuantización y estructura de datos *normal float 4* (NF4) es un proceso de cuantización y estructura de datos ideado para ser entrópicamente óptimo en cuantización de datos distribuidos normalmente [44], con media 0 y varianza 1.

Divide los datos en 16 bloques equiprobables siguiendo la función de distribución gaussiana, convirtiendo cualquier peso que se encuentre en uno de estos intervalos en el representante de ese bloque. De igual forma que con la cuantización entera es necesario guardar valores sobre la distribución de los pesos, con la cuantización de tipo NF4 es necesario guardar la constante de normalización absoluta (el valor absoluto máximo de un bloque de pesos) para poder recuperar la escala original durante la descuantización. ○

Es el tipo de datos empleado en QLoRA, una adaptación de LoRA para modelos cuantizados. Además de implementar esta estructura de datos, QLoRA emplea doble cuantización (cuantización de los parámetros de cuantización) para conseguir una reducción aún mayor del tamaño en memoria del modelo.

2.5. Adaptadores de visión

Los LLM son modelos entrenados sobre lenguaje, lo que impide que procesen imágenes sin modificaciones a su estructura. Para dotarlos de la capacidad de procesar imágenes se les añade una capa de preprocesado de imágenes llamado adaptador de visión, que funciona como puente entre las imágenes y el espacio donde vive la entrada del modelo.

La función del adaptador visual es considerablemente diferente de la del tokenizador, por ser las secuencias de texto y las imágenes entidades de naturalezas muy diferentes. Mientras el texto es fácilmente compresible, las imágenes no lo son; mientras el texto plano se fragmenta en elementos discretos, utilizar este enfoque con imágenes requeriría dividir la imagen en fragmentos minúsculos que no tienen información contextual.

En LLM modernos (GPT-4, GPT-5, etc.), es habitual un vocabulario en el rango de 100 000 – 250 000 tókenes, aunque los modelos más simples (a nivel de

carácter) emplean un vocabulario de unos pocos cientos símbolos. En contraste, un solo píxel de una imagen en formato RGB de 32 bits toma un color de entre 2^{24} posibles, lo que, considerando también la capa de transparencia, asciende a 2^{32} valores posibles.

El tamaño de un teórico *vocabulario visual* análogo al vocabulario textual, incluso uno que surgiera de agrupar solo unos pocos píxeles, sería una potencia de este valor, lo que lo hace imposible en la práctica. Esta posibilidad se vuelve aún más improbable cuando tenemos en cuenta que las imágenes pueden contener hasta millones de píxeles, por lo que serían necesarios cientos, miles o millones de estos elementos de vocabulario visual, para procesar la imagen.

La única alternativa, por tanto, resulta no realizar una operación tan rígida como la tokenización y posterior embebimiento de estos tókenes. Para esto, se emplea un *Vision Transformer* (ViT) como capa de preprocesado de las imágenes [46]: un ViT divide las imágenes en fragmentos cuadrados (por ejemplo 16×16), y emplea la información de color de cada uno de ellos como entrada, proyectándolos linealmente y procesándolos mediante un transformer encoder-only [18].

El adaptador recibe la información del *Vision Transformer*, que no tiene por qué haberse entrenado en conjunto con el LLM, y tiene como función reconciliar las dimensiones de las representaciones ocultas de ambos modelos. Aunque depende del modelo LVLM que se esté empleando, habitualmente involucran operaciones matriciales y no lineales mezcladas, entrenándose para, entre otras, mantener relaciones de similitud y distancia entre el espacio de entrada (el de salida del ViT) y el del LLM [18].

Esto produce una serie de vectores continuos, posiblemente miles, de información visual. Puesto que la complejidad del mecanismo de atención de los LLM es cuadrático y son modelos considerablemente más grandes que los ViT, es necesario un paso de compresión y filtrado de estos vectores visuales para reducirlos en número.

Tras esto, existen varias alternativas posibles para introducir los vectores visuales en nuestro LLM. Entre ellas, las dos más naturales son:

1. Los vectores pueden concatenarse antepuestos a los vectores de embebimiento de los tókenes textuales (si es que hay alguno), y procesarse desde el principio.

2. Por otro lado, pueden introducirse de forma progresiva conforme la información textual fluye a lo largo del transformer, en los mecanismos de atención. En lugar de introducirse desde el principio, el modelo considera los vectores visuales únicamente para obtener vectores K y V , empleando los vectores Q del texto de entrada para realizar atención cruzada entre la información visual y la información textual. El proceso es análogo al proceso de atención cruzada con la información de un encoder hipotético, pero la interpretación de la información añadida durante la atención cruzada es completamente diferente, al estar incluyéndose únicamente la información visual en las representaciones intermedias de lo que originalmente era solo texto.

3.

El dataset

La efectividad de cualquier modelo de aprendizaje profundo está estrictamente supeditada a la calidad de los datos que procesa durante su entrenamiento. El presente trabajo se enfrenta a un corpus documental que no cuenta con un formato estandarizado asimilable por una red neuronal. El conjunto de datos base está constituido por documentos procedentes de dos fuentes de naturalezas dispares: por un lado *La longue marche à travers la théorie de Galois*, compuesto por notas manuscritas de Grothendieck centradas en la teoría de topos y categorías aplicada a la teoría de Galois; por otro, una recopilación de cartas manuscritas dirigidas a diferentes individuos.

La transición de estos documentos (históricos, ruidosos, irregulares, de caligrafía hostil y plagados de notación matemática abstracta) a un conjunto de datos estructurado exige primero reconciliar imágenes con su representación textual. El que la transcripción parcial existente de *La longue marche à travers la théorie de Galois* estuviera codificada en el TeX original de Knuth, empleando paquetes y macros ahora perdidos, añade complejidad a este proceso.

Este capítulo detalla el proceso seguido para sortear las múltiples patologías de nuestros datos crudos. Desglosamos secuencialmente el proceso de obtención del material crudo, el pretratamiento y traducción a LaTeX y el despliegue una arquitectura de etiquetado colaborativo a través de Label Studio. Finalmente, se exponen los fundamentos seguidos para el aumento de datos y posterior división entre entrenamiento y evaluación, evitando la contaminación cruzada de la división dadas las complejas dependencias estructurales del corpus.

La implementación y el detalle más preciso y exhaustivo del tratamiento y procesado de estos datos (cálculo del orden de lectura, división en párrafos, contextualización de muestras, etc.) se tratan en el [Anexo A](#).

3.1. Obtención

Las transcripciones de *La longue marche à travers la théorie de Galois* a las que tenemos acceso se solicitaron directamente a Jean Malgoire, discípulo de Grothendieck, heredero en vida de parte de sus escritos e impulsor de la transcripción y difusión de su obra tardía. Los escaneos de la obra se obtuvieron del [archivo Grothendieck de la Universidad de Montpellier](#), a la que J. Malgoire donó parte de los escritos que recibió de Grothendieck.

Las cartas se obtuvieron del [Grothendieck Circle](#), así como de la [Fundación Salvador Seguí](#) y del archivo privado de J. A. Navarro, requiriendo ser transcritas primero. Puesto que son contenido que Grothendieck producía para ser leído por un ajeno, sin embargo, la caligrafía es considerablemente mejor que la de *La longue marche à travers la théorie de Galois*, por lo que fue posible emplear herramientas de transcripción automática directamente sobre ellas, ahorrando trabajo humano.

3.2. Transcripción y pretratamiento

El primer paso en el caso de ambos grupos de documentos era claro, pues era necesaria una transcripción. En el caso de las cartas, transcribirlas no era un problema: puesto que Grothendieck las escribió con el propósito de ser leídas, la caligrafía es cuidada y limpia; en el caso de «*La longue marche à travers la théorie de Galois*», sin embargo, este no era el caso.

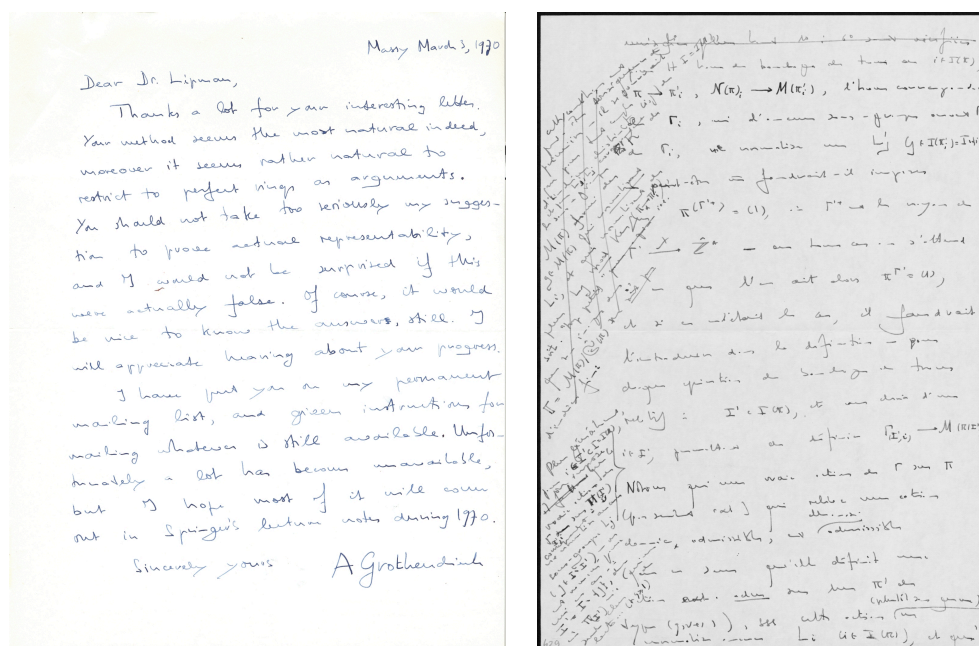


Figura 17. A la izquierda un ejemplo de una de las cartas, a la derecha una de las páginas de *La longue marche à travers la théorie de Galois*.

Debido a su dificultad y longitud, realizar una transcripción manual de *La longue marche à travers la théorie de Galois* es virtualmente imposible. Se partía en este documento, sin embargo, de una singular ventaja: parte de él ya había sido transcrito en torno al año 2007 por uno de los discípulos más cercanos a Grothendieck, Jean Malgoire [47]. Esta transcripción parcial, que había sido expandida en años siguientes por el propio J. Malgoire, presentaba, por otro lado, una problemática igualmente particular: estaba escrita en el TeX original de Knuth, y los paquetes y archivos de *macros* usados por Malgoire se han perdido para siempre. Para acceder a su contenido sería necesaria una fase de traducción a LaTeX y completar aquellos vacíos dejados por abreviaturas y comandos perdidos.

Esta transcripción parcial estaba agrupada en aproximadamente los mismos capítulos que el documento manuscrito, siendo cada uno un documento .tex independiente.

3.2.1. Modernización de *La longue marche*

La traducción de *La longue marche* a LaTeX resultó ser una labor compleja por la naturaleza de las abreviaturas (no pretenden dar contexto de lo que debería haber en su lugar, sino simplemente ser cortas), la heterogeneidad de su uso en

diferentes archivos, la complejidad de la caligrafía de Grothendieck y el nivel de abstracción de los temas tratados en el documento. Pese a que se contaba tanto con el contexto transcrito de muchas abreviaturas como la referencia manuscrita, era necesario un análisis bastante pormenorizado de cada caso individual para poder sustituir con certeza una abreviatura por contenido.

Tras la primera etapa de sustitución de abreviaturas, por otro lado, era necesario convertir el documento del TeX original a LaTeX moderno. Este proceso comenzó haciéndose de una forma parcialmente manual, descrita en el [Anexo A](#), en la [Sección A.1](#). Este proceso combinaba la transcripción y segmentación del texto a nivel de página, y se empleó para las páginas 1-300.

El proceso, sin embargo, era extremadamente lento e ineficiente, lo que motivaría un cambio de estrategia. Un primer paso fue considerar que la tarea de división en páginas y la traducción debían hacerse por separado, lo que permitió un proceso en dos etapas:

1. Traducción del contenido completo usando un LLM. Tras esto, el contenido se revisaba manualmente.
2. Adición de marcas de división de páginas en el documento traducido, tras lo que este se dividía automáticamente en documentos LaTeX mínimos correspondientes con cada página empleando un *script* de *Python*.

Con este proceso se transcribirían las páginas restantes, aproximadamente 300-605.

3.3. Etiquetado

Pese a que una segmentación a nivel de página puede parecer, a priori, un conjunto de datos razonable, debemos tener en cuenta la arquitectura de nuestros modelos.

Un LLM multimodal a menudo descompone una pequeña imagen en miles de tokens: si la imagen contiene mucha información (muchas palabras, para lo que requerimos muchos píxeles), la cantidad de tokens es enorme. Puesto que la capacidad de transcripción y atención se degrada conforme aumenta el número de tokens [48], esta opción pierde viabilidad en las complejas páginas de *La longue marche à travers la théorie de Galois*. A esto se suma que un modelo con millones de parámetros no tiene ninguna dificultad para memorizar 600 u 800 muestras, por lo que es necesario realizar una división más fina de nuestros datos para que el entrenamiento no resulte en un sobreajuste inmediato.

Resulta más sencillo, además, para un modelo realizar un aprendizaje progresivo: en primer lugar «aprende» a transcribir fragmentos pequeños de contenido, y, solo tras esto, comienza a transcribir fragmentos más grandes. De esta forma, aprende la caligrafía del individuo de una forma más suave y gradual, evitando enormes errores iniciales si se requiere que transcriba fragmentos demasiado grandes. Esta dinámica es un caso particular del llamado *aprendizaje por currículum* (o *curriculum learning*) [49], que se detalla en la [Subsección 4.2.2](#).

De forma algo heurística, se empleó como división objetivo del contenido la línea manuscrita, pues la división a nivel de párrafo es demasiado basta, y la división a nivel de palabra no permite usar información contextual que sí puede conseguir una línea. A esto se suma que, si la división a nivel de línea se hace correctamente, pueden componerse bloques más grandes, multilínea, a partir de líneas adyacentes concatenando sus transcripciones.

Nuestro conjunto de datos final contiene unas 11 500 líneas, lo que ejemplifica la imposibilidad de realizar la división mediante técnicas poco sistemáticas como recortar las imágenes a mano línea a línea, cortar y pegar sus transcripciones. El volumen de datos, además, apuntaba a que sería necesaria colaboración de voluntarios para conseguir el etiquetado a tiempo.

Tras barajar diversas opciones, se optó por emplear Label Studio, que se propone como un sistema de interfaz gráfica y gestión para anotación de conjuntos de datos, precisamente con el objetivo de conseguir muestras para entrenamiento de modelos. La versión gratuita, de la comunidad, permite configurar la interfaz ampliamente, colaboración entre múltiples etiquetadores y el exportado de los datos de etiquetado, así como una gestión de los mismos relativamente sencilla usando su API.

Label Studio se instala como un paquete de Python, y permite abrir un *host* local accesible mediante el navegador, dentro del cual podemos configurar la interfaz. Aquí, cada página manuscrita con su transcripción asociada se denomina «tarea», y, cada persona que accede para:

- recortar las imágenes en líneas,
- seleccionar el texto de cada línea,
- corregir el texto de aquellas líneas que estén mal, o
- realizar la asociación imagen-texto

se denomina «anotadora». Una vez configurada la interfaz y subidas las tareas, que se configuraron empleando un script de *Python*, la interfaz es la siguiente:

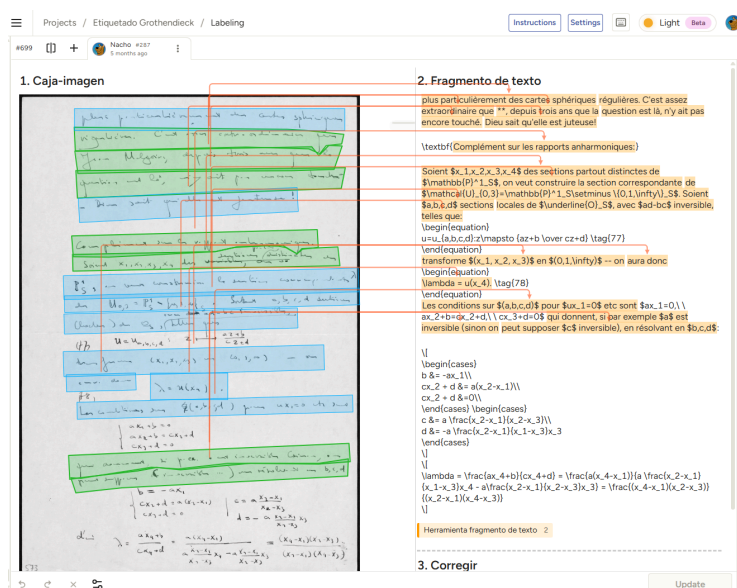


Figura 18. Interfaz configurada de Label Studio ante una tarea ya anotada. Captura tomada directamente del servidor de Label Studio, accesible en grothendieck.online.

Se configuraron para realizar las anotaciones 3 herramientas diferentes y una utilidad aparte:

1. Herramienta marco rectangular: permite realizar recortes cuadrados. Tras usarla, persiste sobre la imagen, en aquella región que se ha marcado, un rectángulo de las dimensiones dibujadas. Este rectángulo puede editarse en cualquiera de sus dimensiones e incluso rotarse tras haber sido dibujado, lo que permite ajustes finos.
2. Herramienta marco poligonal: permite dibujar polígonos clicando dondequiera que se desea un vértice sobre la imagen. Al igual que el marco rectangular, es posible modificarlo tras haberlo dibujado, quedando su forma dibujada sobre la imagen.
3. Herramienta fragmento de texto: permite dividir el texto en secciones. Estas herramientas sirven para segmentar las dos partes del contenido de cada tarea (la imagen y el texto), que luego pueden asociarse usando una «relación» de Label Studio por el propio usuario.

Además, existe la posibilidad de que el usuario corrija el texto presente en un fragmento concreto (delimitado por él) colocando el nuevo texto en una caja independiente

3. Corregir

Selecciona un fragmento de texto para poder corregirlo.

(Opcional) Escribe aquí el texto corregido...

Add

Figura 19. Campo de corrección en la interfaz de Label Studio

3.3.1. Colaboración, servidor en línea

Emplear Label Studio simplifica el proceso de realizar la anotación de una tarea cualquiera, pero la carga de trabajo sigue siendo sustancial. En consecuencia, se decidió que el trabajo de anotar todas las tareas debía repartirse entre voluntarios (previa instrucción de los resultados objetivo) para poder distribuir así la carga.

Una primera alternativa involucra distribuir diferentes subconjuntos del total de tareas (unas 700) entre todos los usuarios, como archivos ya listos para ser cargados en Label Studio. Tras esto, es posible instruirlos en cómo abrir su propia instancia de Label Studio, configurarla y cargarla con las tareas asignadas. Las dos ventajas de esta opción son que requiere una cantidad moderada de trabajo desplegarse (pero se repite para todos los usuarios) y no es necesaria más infraestructura que un ordenador por usuario (que es un requerimiento inevitable).

Las problemáticas son, sin embargo, múltiples y bastante decisivas:

1. Requiere un conocimiento técnico algo más detallado que el estrictamente necesario para etiquetar los datos: es necesario descargar Python, abrir la línea de comandos cada vez que se quiere abrir el servidor local, cargar el archivo de tareas y navegar algo más en detalle por las interfaces de Label Studio.
2. El etiquetado de tareas no está centralizado, sino que sería necesario que un usuario compartiese al finalizar el archivo exportado. Revisar las tareas es, entonces, una labor algo más compleja que si se encuentran todas en un mismo servidor.

3. La repartición de tareas es, en este caso, extremadamente inelástica. Un mismo usuario recibe un lote cerrado de tareas que se presupone va a completar, sin ser posible que estas se dejen «a medias» o que un tercero las termine.

Esta combinación de problemas hace bastante poco operativa esta solución, especialmente si, como es nuestro caso, queremos minimizar la fricción entre el voluntario y la tarea. Ya que nos legan su tiempo de forma desinteresada, lo mejor es facilitar esta tarea lo máximo posible.

Otra alternativa, la que finalmente se impondría, es la de abrir un servidor con alguna compañía de infraestructura en la nube y alojar allí una versión centralizada del servidor de Label Studio. Así, tras abrir los puertos del servidor para aceptar conexiones entrantes (de forma que no fuera un servidor de Label Studio «local» ejecutándose en una máquina ajena), sería accesible desde cualquier ordenador, mediante un navegador. Puede accederse a este servidor mediante grothendieck.online.

El servidor se alojó en una instancia de Oracle bajo un plan gratuito. Por las limitaciones de las instancias disponibles, fue necesario guardar las imágenes en un sistema de almacenamiento aparte, un «*bucket*», que también se alojaría en Oracle, también mediante el plan gratuito.

Así, el servidor de Label Studio accede de forma dinámica a las imágenes, cargándolas en memoria durante la anotación, y libera esa memoria cuando no se están anotando, lo que ahorra memoria física a la instancia de cómputo.

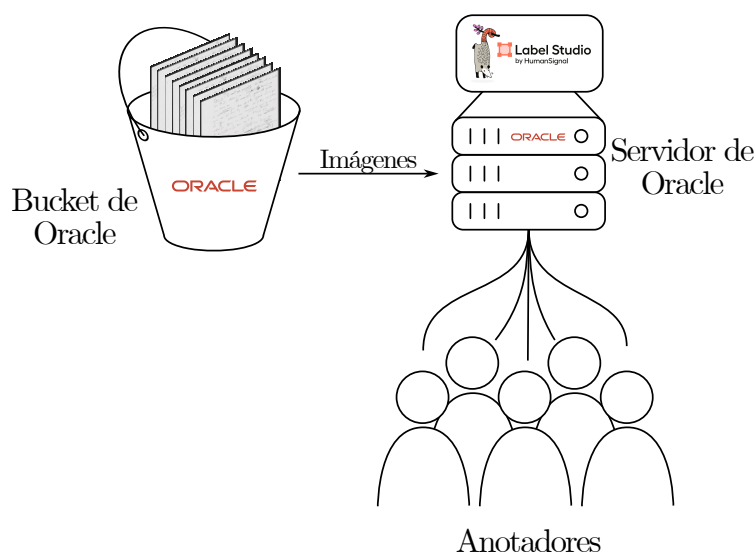


Figura 20. Esquema de la relación entre el *bucket*, la instancia de cómputo (el «servidor») y los anotadores.

3.3.2. Ingesta de los datos de Label Studio

Label Studio produce como salida un archivo, llamado *export*, en formato JSON que contiene la información de todas las anotaciones. Esta información es de la forma:

Fragmentos de texto Texto concreto que contiene.

Recorte rectangular Anchura, altura, rotación y coordenadas del centro del rectángulo. Estos valores se miden relativos al tamaño de la página, por lo que son invariantes por escala.

Recorte poligonal Coordenadas relativas al tamaño de la página de cada vértice.

Corrección El identificador único del texto que está corrigiendo y el texto corregido.

Asociación Contiene el identificador único de los dos objetos que asocia. De forma óptima, deberían ser siempre una imagen y un fragmento de texto.

Queremos, sin embargo, no información sobre las anotaciones como tal sino recortes y transcripciones ya realizadas, para poder proporcionárselas al LLM. Sería necesaria, entonces, una interfaz que realizase los recortes y asignase las transcripciones según la información dada en el *export* de Label Studio. Esta es precisamente la labor del submódulo *processing* del proyecto de *Python* descrito en [Sección A.3](#).

3.3.3. Homogeneización

El texto resultante de las anotaciones, aunque en muchas instancias corregido y depurado, hereda del original transcrito por Jean Malgoire las inconsistencias propias de la escritura humana en LaTeX. La presencia de espacios, la forma de escribir algunos comandos, el espaciado dado al texto (especialmente en ecuaciones), etc. hacen que una misma imagen pueda tener transcripciones que, aunque muy similares tras ser renderizadas, textualmente son muy diferentes.

Así, llamamos *homogeneidad* de nuestras transcripciones a cómo de consistentemente se transcribe cada fragmento de él: espaciado, símbolos, matemáticas en general, si se desabrevian o no abreviaturas presentes en el manuscrito, etc. Puesto que durante el entrenamiento estamos penalizando transcripciones según su texto plano (no su resultado visual tras renderizarse en LaTeX), el aprendizaje del modelo depende de la homogeneidad de las transcripciones.

Se debe, por tanto, adaptar el texto recibido para homogeneizar las transcripciones y reducir en la medida de lo posible las variaciones arbitrarias en él presentes. En concreto, esta labor recaería sobre la interfaz que actúa entre el *export* de Label Studio y la generación del dataset. Se buscarían manualmente estructuras propensas a escribirse de formas diversas y patrones que admitiesen regularización, y se sustituirían automáticamente antes de generarse el conjunto de datos. El proceso se desglosa con algo más de detalle en la [Sección A.3](#) del [Anexo A](#).

3.4. Aumentado de datos

Nuestra intención última es que el modelo sea capaz de transcribir textos arbitrarios²⁰, para lo que será necesario que, durante su entrenamiento, se haya enfrentado tanto a muestras con una sola línea como muestras con más.

En algunos modelos de transcripción, es posible emplear como muestras de entrenamiento la concatenación de cualquier par de muestras arbitrario. El nuestro, por otro lado, puesto que conoce la estructura estadística del lenguaje, se ve beneficiado si aquellas líneas que se concatenan para producir muestras de entrenamiento multilínea son adyacentes en el texto original. Esto garantiza que su transcripción concatenada posee una correcta estructura gramatical y

²⁰Con posibles matices, véase la [Sección 5.2](#) sobre trabajos futuros.

semántica, lo que simplifica la tarea del modelo y refuerza su conocimiento del lenguaje específico del texto.

La tarea de inferir el orden en el que deben leerse las líneas a partir de las anotaciones resultó un problema no trivial, y su elección se discuten en la [Sección A.3](#) del [Anexo A](#). Una vez que se posee este orden, es posible generar muestras *artificiales* a partir de las líneas individuales (cuyas muestras correspondientes llamaremos *naturales*) concatenando sus textos relativos y colocando los recortes en sus posiciones originales en la imagen. Tras esto, y para ahorrar espacio, la imagen se recorta siguiendo los ejes para delimitar de la forma más estricta posible los recortes en ella presentes, y se rellena el espacio vacío con el color promedio de la página.

3.4.1. Contextualización

Empleamos un LLM precisamente por sus capacidades para generar texto coherente, lo que nos es útil en la tarea de transcribir un documento completo página a página. Para poder aprovechar esta información contextual, sin embargo, requerimos que el modelo reciba, además del recorte concreto, información sobre el texto que precede al que va a transcribir.

Una vez que tenemos asentado el orden de lectura, nos basta con introducirle al modelo parte del texto que le precede en su página o, si se encuentra al principio de la página, de la página anterior si existiera.

Este fragmento de texto anterior, el «contexto», se guarda en cada muestra. Es posible contextualizarlas por palabras completas o por caracteres a partir de este corpus.

3.5. División entrenamiento/validación

El proceso de división entre validación y entrenamiento responde a una tensión entre ambos grupos:

1. Se quiere hacer el conjunto de entrenamiento lo más grande posible, de forma que el modelo tenga todos los datos posibles para el entrenamiento.
2. El conjunto de validación debe ser lo más diverso posible, y asemejarse lo máximo posible al caso de uso real, de forma que la métrica de precisión final sea lo más cercana posible al despliegue del modelo.

Además, existe la problemática de que ambos conjuntos deben ser completamente disjuntos entre sí, pues la mejora en las muestras del conjunto de entrenamiento está garantizada, al menos para modelos suficientemente complejos.

Definición (División contaminada): Decimos que un conjunto de *test* (o de *train*) está *contaminado* cuando presentan muestras comunes, o parcialmente comunes. Este comportamiento es indeseado ya que enmascara las métricas de precisión obtenidas, al haberse entrenado el modelo en parte del conjunto de *test*. ○

Nuestro conjunto de datos tiene, además, la problemática añadida del aumento de las muestras y la contextualización. Una división de las muestras *naïf* tiene los dos siguientes problemas:

1. alguna de las muestras multilínea del conjunto de entrenamiento pueden tener alguna de sus líneas componentes en el conjunto de evaluación. Esto supone una contaminación parcial de nuestro conjunto de datos.
2. Si una línea pertenece al conjunto de entrenamiento, pero la anterior en su misma página pertenece al conjunto de evaluación, cuando el modelo recibe el contexto recibe información sobre el contenido de la línea en evaluación. De nuevo, esto supone una contaminación parcial.

La solución al primer problema pasa por tener en cuenta que las muestras multilínea siempre contienen líneas de una misma página. En consecuencia, si realizamos la división entre *train* y *test* a nivel de página, no hay contaminación posible entre conjuntos. Puesto que existe una gran disparidad en número de líneas entre diferentes páginas, pese a hacerse considerando las páginas bloques indivisibles, la división se hace intentando balancear las páginas de forma que la proporción entre líneas en cada conjunto sea la buscada (por ejemplo 95% o 99%). Asimismo, puesto que tenemos muestras de dos orígenes muy diferentes, las cartas y LaLoMa, se realizan por separado las divisiones, de forma que podemos asegurar una proporción muy similar de líneas procedentes de cada fuente en entrenamiento y evaluación.

Ante el segundo problema, sin embargo, la única solución aparente que maximiza la cantidad de contexto que podemos aportar (hacer que las páginas correspondientes a cada conjunto sean contiguas), no nos es válida. Puesto que queremos que el conjunto de validación sea una muestra representativa y

variada de nuestro conjunto de entrenamiento, considerar páginas subsecuentes sesga enormemente el vocabulario usado a aquel perteneciente a ese capítulo.

La solución, sin embargo, no involucra ningún cambio al modelo, sino una correcta implementación de la contextualización. El modelo recibe, cuando se le proporciona contexto, un bloque de contenido de la forma:

```
<|system> You are an expert transcripator [...]
<|user> [Tókenes imagen]
<|user> For reference, this is the previous context: [Contexto]
<|agent> [Transcripción del modelo]
```

Naturalmente, el modelo solamente debe entrenarse con los tókenes que suceden a <|agent>, no con el corpus completo (al contrario que durante el preentrenamiento). En consecuencia, puesto que el modelo no ve sus pesos modificados para favorecer la generación del contexto, sino la del texto siguiente en base a este contexto, no hay problema en proporcionarle contexto del conjunto de evaluación. En nuestra implementación, esta restricción al entrenamiento (muy común en ajuste fino) se consigue con un ajuste sencillo «train_on_responses_answers».

3.6. Sistema de evaluación

La métrica de error estándar dentro de la industria de la transcripción es el CER, o *Character Error Rate*, se define en base a la distancia de Levenshtein.

Definición (Distancia de Levenshtein): Sea $\mathcal{W}$ un conjunto de símbolos, se define en base a él la distancia de Levenshtein, $\text{Lev} : \mathcal{W}^* \times \mathcal{W}^* \rightarrow \mathbb{N}$, como el mínimo número de deleciones, inserciones o sustituciones a cualquiera de las dos secuencias para transformarla en la otra. $\circ$

Ejemplo (De distancia de Levenshtein): Sea $\mathcal{W} = \{1, 2, 3\}$ el conjunto de símbolos, y sean $A = 123123$ y $B = 32123$ dos secuencias. La mínima composición de inserciones, deleciones y sustituciones que puede realizarse para llegar de una a otra es:

$$A = 123123 \sim 12 \underset{3}{1}23 \sim \underset{1}{3}2123 = B,$$

luego $\text{Lev}(A, B) = 2$. $\circ$

Definición (CER): Se define el CER (*Character Error Rate*) entre una cadena de símbolos de referencia R y otra, candidata a transcripción, C , como la distancia de Levenshtein relativa:

$$\text{CER}(R, C) = \frac{\text{Lev}(R, C)}{|R|}.$$

○

A menudo se considera el CER como el «porcentaje de error» del modelo: si contamos con un CER del 50%, a grandes rasgos quiere decir que el modelo se equivoca en uno de cada 2 caracteres que transcribe. Puesto que se calcula como un porcentaje con respecto a la longitud de la secuencia de referencia, podemos tener un CER mayor al 100% en casos donde la transcripción sea de muy baja calidad.

Dependiendo de la elección del vocabulario $\mathcal{W}$ es posible definir una métrica para cada modelo, alternativa al CER aun siendo un caso particular del mismo, el TER.

Definición (TER): Se define el TER (*Token Error Rate*) a partir de un LLM como el CER que resulta de tomar $\mathcal{W}$, el conjunto de símbolos, igual al vocabulario del LLM, $\mathcal{X}$.

○

El TER penaliza de forma homogénea cada generación incorrecta en lugar de carácter incorrecto. El problema de este tipo de métricas *ad hoc* es que los tokenizadores son, en general y por motivos heurísticos, procesos estocásticos [50], y es además mucho más representativo y sencillo para un ser humano interpretar el CER que el TER.

La elección de conjunto de símbolos, aunque formalmente sea el abecedario de la lengua a transcribir junto a números y símbolos, a menudo de forma práctica es el conjunto de caracteres representables en una codificación concreta, como UTF-8.

Nuestro caso, por otro lado, no es una tarea de transcripción genérica, sino que queremos que nuestro modelo transcriba a LaTeX. En consecuencia, podría ser coherente alejarnos del estándar e incluir en el vocabulario algunos símbolos extra²¹. Esto, sin embargo, pues nos aleja del estándar y disminuye nuestro

²¹Por ejemplo el alfabeto griego en comandos de LaTeX ($\backslash\sigma$, $\backslash\delta$, etc.) como símbolos individuales, ya que, por ejemplo, confundir una instancia de « σ » con « s » conlleva una penalización de 5 puntos al CER pues $\text{Lev}("s", "\sigma") = 5$

error, tiene una motivación cuestionable, razón por la que nos limitamos a la convención de usar el CER «tradicional» en la mayoría de métricas.

El CER tiene el problema de no ser diferenciable, luego, aunque lo empleemos como métrica de desempeño del modelo, no podemos usarlo como función de coste a minimizar durante el entrenamiento (con los problemas de alineamiento que ello conlleva). La función de costes será, como es habitual en LLM, la entropía cruzada.

3.6.1. Generalizaciones del CER a varias muestras

Puesto que nuestro conjunto de validación contiene varias muestras, necesitamos una generalización del CER que considere varias muestras. Emplearemos dos métricas diferentes: por un lado, la media del CER sobre muestras individuales, la métrica natural

$$\overline{\text{CER}}(Y, \hat{Y}) = \frac{1}{|Y|} \sum_{i=1}^{|Y|} \text{CER}(y, \hat{y}) = \frac{1}{|Y|} \sum_{i=1}^{|Y|} \frac{\text{Lev}(y, \hat{y})}{|y|},$$

así como el CER pesado por la longitud de la secuencia a generar, dado por

$$\text{WCER}(Y, \hat{Y}) = \frac{1}{\sum_{y \in Y} |y|} \sum_{i=1}^{|Y|} \text{Lev}(y, \hat{y}).$$

Esta última métrica, el WCER, aunque similar a realizar el CER de la concatenación de todas las secuencias, resulta siempre mayor que este al dividir las secuencias en bloques. En nuestro caso, se emplea como indicador de la «densidad de error por carácter» real que comete el modelo, a diferencia del $\overline{\text{CER}}$, que penaliza de forma homogénea un error alto en líneas individuales como en pares o tríos de líneas.

4.

Modelo y entrenamiento

La transcripción automática de manuscritos históricos con alta densidad de notación matemática y caligrafías extremadamente complejas, como la obra tardía de Alexander Grothendieck, escapa por completo a las capacidades de los sistemas de OCR convencionales y tradicionales. Para superar estas limitaciones y descifrar un texto de caligrafía hostil y composición irregular, resulta indispensable utilizar modelos de aprendizaje profundo que no solo analicen las características visuales del trazo, sino que incorporen un profundo conocimiento contextual y estadístico del lenguaje. Los VLM de pesos abiertos ofrecen el paradigma ideal para esta tarea, permitiendo además un despliegue y entrenamiento estrictamente local que garantiza la confidencialidad de los datos frente a soluciones comerciales privativas basadas en API externas.

Este capítulo expone tanto el modelo concreto escogido y sus características diferenciadoras como el núcleo técnico del ajuste fino realizado. Con el fin de hacer viable el ajuste fino local bajo severas restricciones de memoria gráfica, el proceso de optimización se realizó mediante QLoRA sobre la librería *Unsloth*, lo que permite congelar los parámetros del codificador de imágenes para prevenir olvido catastrófico y concentrar la actualización del gradiente exclusivamente en las capas de lenguaje.

4.1. Modelo empleado

El modelo escogido es [Qwen3-VL 8B.](#), presentado el 23 de septiembre de 2025 y basado en la arquitectura [Qwen3 8B](#). La familia de modelos de visión, entrenados en base a sus contrapartes de lenguaje puro, hereda de estos las decisiones estructurales y arquitectónicas, pero incorporando adaptadores visuales. [21]

Tanto en las versiones de visión como de lenguaje el modelo 8B (8 mil millones de parámetros) se enmarca dentro de su familia en el punto medio entre tamaño y calidad, pues fue concebido para la eficiencia. La arquitectura base del Qwen3-8B, reflejando esto último, es la de un LLM moderno y denso, pero de tamaño moderado: cuenta con 36 capas de *decoder*, con 8 grupos de 4 cabezas de atención grouped-query cada capa, y tiene un vocabulario de 151 669 tokens.

4.1.1. Atención de doble bloque

Los modelos de lenguaje únicamente son capaces de «prestar atención» a contenido que quepa en su ventana de contexto que, recordemos, escala cuadráticamente en complejidad con el número de tokens. Mientras modelos enormes (como los modelos frontera de OpenAI, Anthropic o Google) pueden permitirse matrices de atención descomunales durante inferencia, los modelos orientados a la eficiencia y la portabilidad no. Estos últimos modelos suelen entrenarse en versiones optimizadas de los *patrones de atención*, que ofrecen mejoras en complejidad frente a la atención «estándar».

En esta atención estándar un token «atiende» a todos aquellos que lo preceden, mientras que los patrones de atención optimizada pretenden reducir este número, habitualmente a un «entorno» de los tokens anteriores [51], intentando impactar lo menos posible la calidad de los resultados. Existen variedad de patrones de atención optimizada, empleando la familia Qwen3 un patrón de atención por bloques. En este tipo de patrones se dividen los tokens en bloques, empleándose atención estándar dentro de cada bloque. Para permitir la comunicación entre bloques, se realizan adiciones a estos patrones de atención, y es precisamente en su estructura donde reside la diferencia con otros modelos [5].

Para aprender las matrices de *key*, *query* y *value*, el modelo se entrena en dos fases: una primera, similar a un preentrenamiento²², en la que trata secuencias

²²Véase el [Anexo B](#).

de 4096 tókenes (o menos) de longitud; y otra posterior en la que se extiende la longitud de las secuencias hasta los 128000. Es durante esta segunda en la que el modelo aprende a propagar información entre bloques, cada uno de 4096 tókenes de longitud.

El proceso de enmascarado debe, entonces, realizar una división en bloques (*chunks*), replicando la atención de decoder intra-bloque y empleando nuevos patrones de atención inter-bloque, que permiten que diferentes bloques se comuniquen entre ellos.

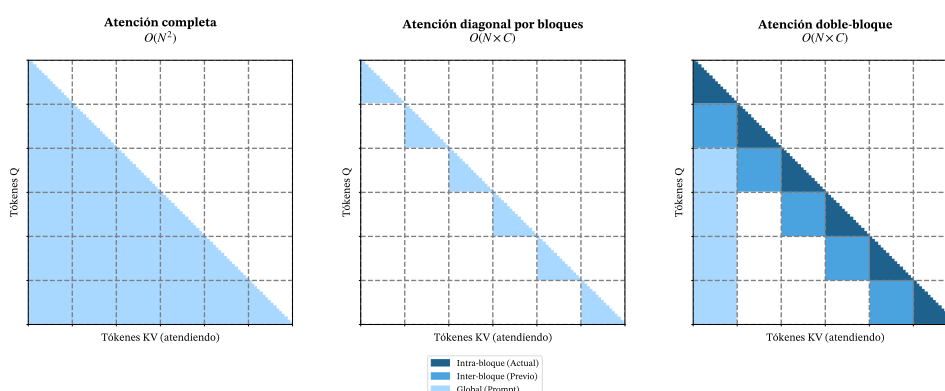


Figura 21. Representación de los patrones de atención. A la izquierda la atención estándar; en el centro un patrón de atención diagonal por bloques, notemos que no permite a la información propagarse entre bloques; a la derecha el patrón de doble bloque, empleado por Qwen3, mediante el que se generalizan los patrones de atención de 4096 tókenes a mayores longitudes. [4], [5]

El patrón de atención empleado, *atención en doble bloque (Double Chunk Attention)*, está fuertemente orientado a la tarea asistencial: los bloques subdia-gonales permiten la propagación de la información de bloques cercanos hacia bloques siguientes, así como se permite a todos los tókenes atender al contenido inicial, el primer bloque, donde se presupone se encuentra la instrucción del asistente. [5]

4.1.2. El adaptador visual concreto

El adaptador visual de la familia Qwen3-VL es una solución innovadora que se presenta en su propio reporte técnico [21]. Trata de resolver simultáneamente varios problemas relativos a la eficiencia, el más relevante de ellos la explosión de complejidad que supone añadir tókenes a diferentes niveles de «enriquecimiento» a partir de un ViT. Mientras que una aproximación ingenua, basada en extraer representaciones intermedias de los vectores visuales, supondría multi-

plicar la complejidad computacional²³ por tantos grupos de representaciones queramos extraer, la alternativa que proponen consigue hacerlo con complejidad lineal.

El adaptador visual cuenta con dos partes principales: un ViT que actúa como codificador visual y un *fusor* (*merger*) que adapta estas representaciones. Cuando se introduce una petición que contiene tanto texto como imágenes, en primer lugar se preprocesa la imagen:

1. La imagen se divide en parches cuadrados y se introduce en un ViT, en concreto SigLIP-2. Llamemos D_V a la dimensión de las representaciones que produce, que permanece invariante en todas las capas.
2. Conforme se procesa la imagen, se extraen tres representaciones intermedias de cada parche: uno inicial, otro intermedio y otro final. Puesto que el contexto se propaga conforme aumenta la profundidad, si llamamos V_1, V_2, V_3 a las representaciones a estas tres alturas de todos los parches, la información contextual presente en cada uno crece con el índice.
3. El ViT genera su representación final de cada uno de los parches de entrada. Llamamos a este grupo de representaciones V_4 . La función del codificador de visión termina aquí.

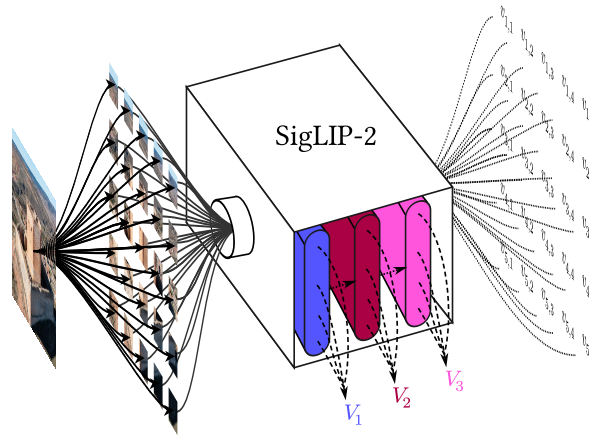


Figura 22. Diagrama de la obtención de las representaciones intermedias V_1, V_2, V_3 , así como de las finales, V_4 , a partir del ViT SigLIP2.

Tras obtener estas representaciones, comienza la actividad de fusor, que actúa dentro de cada grupo V_i de forma independiente:

²³Con el aumento cuadrático de la complejidad que ello supone.

1. Se agrupan las representaciones en aquellas que corresponden a grupos 2×2 de parches adyacentes en la imagen original, sin superposición entre grupos. Esto resulta en subgrupos de 4 vectores,

$$\{v_{2a,2b}^i, v_{2a+1,2b}^i, v_{2a,2b+1}^i, v_{2a+1,2b+1}^i\} \subseteq V_i;$$

que se concatenan formando un único vector de dimensión $4D_V$.

$$\text{Proy} : \mathbb{R}^{2 \times 2 \times D_V} \longrightarrow \mathbb{R}^{4D_V}$$

$$\begin{pmatrix} \vec{v}_1 & \vec{v}_2 \\ \vec{v}_3 & \vec{v}_4 \end{pmatrix} \mapsto \begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \vec{v}_4 \end{pmatrix}.$$

2. Se reduce la dimensionalidad de este vector empleando un proceso análogo a un perceptrón multicapa con dos capas: primero se proyecta linealmente a una dimensión intermedia, $\tilde{D}$, se aplica una transformación no lineal y, finalmente, se proyecta a la dimensión del LLM:

$$\mathbb{R}^{4D_V} \xrightarrow[\text{compresión}]{W_{P,i}^1} \mathbb{R}^{\tilde{D}} \xrightarrow{\text{(no lineal)}} \mathbb{R}^{\tilde{D}} \xrightarrow[\text{proyección}]{W_{P,i}^2} \mathbb{R}^d.$$

Nótese que cada grupo de representaciones V_i tienen sus matrices de proyección independientes, con parámetros distintos gracias al entrenamiento.

Denotaremos V'_1, V'_2, V'_3 y V'_4 a las versiones proyectadas de V_1, V_2, V_3, V_4 .

Se obtiene tras este proceso, por tanto, un nuevo vector de cada grupo V'_i para cada vecindario 2×2 sin superposición de parches adyacentes de la imagen original.

Una vez se han conseguido esta representación de los vectores, se toma la más enriquecida, V_4 , y se emplean sus vectores como entrada visual para el LLM, lo que se consigue introduciendo estos vectores visuales y, concatenados, los embebimientos de los tókenes textuales. El mecanismo de atención procede ahora de manera ligeramente diferente sobre este contenido visual que sobre el textual, limitándose las diferencias a las 3 primeras capas.

Además de un embebimiento posicional bidimensional RoPE-2D aplicado únicamente a las representaciones visuales, que difiere del RoPE unidimensional usado en las representaciones textuales, tras el paso por el t -ésimo bloque decodificante se suma a la representación intermedia de cada parche (originalmente de V'_4) su vector correspondiente en V'_t .

estos parámetros, primero debemos decidir sobre cuáles de estos grupos (y, en consecuencia, qué parámetros parámetros) queremos afectar.

Para esto, primero debemos tener en cuenta que modelos como SigLIP-2 se entrenan de formas complejas (aprendizaje por contraste) pues son solo codificadores sin un decodificador. Durante un entrenamiento que comprende miles de millones de pares imagen-texto, el modelo aprende una representación extremadamente generalizable de las imágenes que recibe: inferimos de esto, por tanto, que las características recibidas por el LLM son suficientemente ricas para una amplia gama de tareas, entre ellas transcripción [52]. A esto se suma que nuestro conjunto de datos es relativamente modesto: realizar un *fine-tuning* sobre el adaptador requeriría una sistemática completamente diferente, una precisión mucho mayor en la elección de parámetros, y aun de esta forma, el riesgo de sobreajuste es alto. [53]

El *fine-tuning* de capas visuales se ajusta a un problema donde la detección de características del *vision encoder* es insuficiente. Es el caso cuando los patrones visuales son extremadamente complejos, sutiles o se alejan demasiado del conjunto de entrenamiento del modelo (imágenes de biopsias, TAC, análisis de defectos industriales). Dado que nuestro objetivo es la transcripción de texto manuscrito, asumimos que el codificador preentrenado captura correctamente la abstracción de las características visuales del trazo manuscrito. Siguiendo [19], optamos por mantener el codificador de imágenes congelado para reducir el coste computacional y evitar el olvido catastrófico.

En base a esto, podemos afirmar que el objeto de nuestro ajuste fino es, por tanto, únicamente de alineamiento multimodal en nuestra tarea concreta, es decir, el modelo recibe correctamente patrones visuales, pero no sabe interpretarlos y componerlos en una transcripción: no reconoce las letras, no sabe cómo componer el texto a partir de ellas, no está familiarizado con el lenguaje... lo que sí motiva un *fine-tuning* sobre las capas de lenguaje, dejando congeladas las correspondientes al ViT y el adaptador visual.

4.2.2. Entrenamiento por currículum

Es un fenómeno algo singular que un conjunto de datos se preste a definir una métrica de «dificultad» de sus muestras, pues en muchos casos se reducen a un enorme grupo de datos tabulares. En estos casos, en tareas de clasificación

o predicción, incluso definir una métrica de dificultad basada en nuestra intuición humana puede no transportarse bien a la forma de aprender del modelo, volviéndola inútil.

Algunos conjuntos de entrenamiento, sin embargo, sí se prestan bien a definir una métrica de estas características. Este es el caso, por ejemplo, cuando las muestras pueden encajarse unas en otras, de forma que una predicción correcta de la mayor de ellas necesariamente implique también un buen rendimiento sobre las menores. En estas situaciones, puede ser beneficioso para la estabilidad del entrenamiento introduzcan las muestras progresivamente, de menor a mayor dificultad [49]. En el caso de un modelo de transcripción, más allá de definir métricas heurísticas que no tienen por qué corresponderse con la dificultad percibida por el modelo (claridad de la imagen, rotación, superposición entre líneas, frecuencia relativa de las palabras empleadas, etc.), nuestro conjunto de datos sí tiene una característica evidente: resulta más difícil transcribir bien dos líneas que una sola.

Definición (Aprendizaje por currículum): Llamamos *aprendizaje por currículum* (*curriculum learning*) al proceso de entrenamiento que, a partir de un conjunto de datos X y una medida de complejidad $P : X \rightarrow [0, 1]$, entrena al modelo en subconjuntos progresivamente más complejos según P [49]. A cada una de las partes del entrenamiento donde cambia el subconjunto empleado la llamaremos «etapa».

Es importante que el cambio de subconjunto sea siempre aditivo, es decir, si X_τ es el subconjunto en la etapa τ -ésima, dado por

$$X_\tau = \{x \in X : a_\tau \leq P(x) \leq b_\tau\}$$

se requiere que $X_\tau \subseteq X_{\tau+1} \forall \tau$. ○

Un modelo mal adaptado a las características visuales recibe tókenes visuales que no puede procesar, inundándose de una información que no reconoce y volviendo las predicciones esencialmente aleatorias, *alucinaciones*²⁴. Mediante el aprendizaje por currículum se pretende aislar diferentes fases del proceso de aprendizaje, estabilizándolo y facilitando la abstracción de las características de los datos: en nuestro conjunto de datos, comenzar con líneas individuales supone que el modelo comience aprendiendo únicamente la tarea de transcrip-

²⁴Véase la [Sección 4.3](#), sobre resultados.

ción en su caso más sencillo, dejando la composición de textos multilínea para etapas posteriores.

Una vez el modelo ha aprendido parcialmente a transcribir líneas individuales, la siguiente etapa parte de un punto en el paisaje de pérdida que está lejos de ser aleatorio. La tarea para el modelo se transforma ahora en saber cómo componer pares de líneas concretas y cómo interpretar las características visuales de dos líneas diferentes. Puesto que los subconjuntos de diferentes etapas están encajados, incluso en etapas posteriores el modelo continúa incentivado a transcribir correctamente las líneas individuales.

4.2.2.1. Currículum escogido

El aprendizaje por currículum supone, sin embargo, un enorme riesgo de sobreajuste: puesto que las muestras de líneas individuales se conservan en todas las etapas del currículum, el modelo emplea estas muestras tantas veces como épocas realiza. En un conjunto de datos de tamaño moderado, como el nuestro, el riesgo de sobreajuste es, en consecuencia, alto si realizamos demasiadas iteraciones.

Nuestro currículum, por tanto, refleja tanto la necesidad de entrenar al modelo con muestras de diferentes tamaños, para conseguir que el modelo aprenda correctamente a transcribir muestras multilínea, como la moderación en épocas para evitar el sobreajuste. Concretamente, representando en orden las épocas a realizar y aquellas líneas a emplear, el currículum de datos empleado es el siguiente:

- 3 épocas con una sola línea.
- 3 épocas con una y dos líneas.
- 3 épocas con una, dos y tres líneas.

4.2.3. Contextualización estocástica

La intención con el entrenamiento es conseguir un modelo relativamente versátil, aunque enfocado a la escritura de Grothendieck. En concreto, queremos que el modelo tenga la capacidad de transcribir una imagen tanto habiendo recibido el texto anterior (contextualizada) como cuando no (sin contextualizar). Para esto podrían entrenarse dos modelos diferentes, pero consideramos que ambas tareas son suficientemente similares para que el modelo no solo aprenda a

realizar ambas, sino que su rendimiento en ellas sea equiparable al de cualquier modelo entrenado en una sola. Esto se debe, claro, a que las tareas son complementarias y, hasta cierto punto, mutuamente necesarias.

Una posibilidad para incorporar este tipo de entrenamiento es realizar un entrenamiento alternante: realizar unas pocas iteraciones o una época con muestras contextualizadas, continuar con muestras sin contexto, volver después a aquellas con contexto... Si se hace por épocas, el contenido de los conjuntos de entrenamiento es demasiado diferente entre ellas, lo que llevaría a inestabilidad en el entrenamiento. Por otro lado, si se hace por iteraciones, sería necesaria una repartición manual de las muestras para asegurarnos de que todas se emplean como muestras con y sin contexto.

Finalmente se optó por contextualizar de forma aleatoria cada muestra individual durante el entrenamiento. Esto quiere decir que las muestras de datos se guardan siempre en su forma original, y no es hasta que el entrenamiento las requiere (en cada época) que emiten una copia de sí mismas transformada, estructurada como conversación, y contextualizada aleatoriamente²⁵.

Asimismo, para evitar que el modelo se sobreadapte al texto presente en líneas anteriores y memorice la distribución concreta, la cantidad de contexto añadida también se aleatoriza. Cuando se estructura una conversación a contextualizar, se toma la cadena de contexto completa y se recorta su longitud aleatoriamente entre un valor $c_{\min}$ y $c_{\max}$.

4.2.4. Hardware y librerías

Un modelo cuantizado en precisión de 4 bits requiere aproximadamente de tantos gigabytes de VRAM como miles de millones de parámetros tenga. En entrenamientos puramente textuales, la mayor parte de la carga de memoria es el modelo; en nuestro caso, sin embargo, es el procesamiento de las imágenes. Puesto que empleamos imágenes de calidades relativamente altas en todos los casos, los requerimientos de memoria para acelerar el entrenamiento se disparan, siendo muy superiores a los $\approx 9\text{GB}$ que indica el tamaño del modelo.

Trabajamos con un ordenador con Windows Server, trabajando en Linux (forzosamente, por las librerías empleadas) mediante WSL. El sistema cuenta con 256 GB de RAM habitual, un AMD EPYC 740P de 24 núcleos a una

²⁵Véase la [Subsección A.4.1](#), sobre la implementación de transformaciones *on-the-fly*.

frecuencia base de 2GHz ampliable en *boost* hasta los 3GHz. La capacidad gráfica se encuentra en dos tarjetas NVIDIA A6000 de la serie *Ampere* con 48 GB de VRAM cada una. Por las librerías empleadas, sin embargo, y debido a la situación de utilización del sistema, únicamente se tenía acceso real a una de estas dos tarjetas gráficas durante el entrenamiento e inferencia. Parte del entrenamiento e inferencia se realizaron también en los cuadernillos *online* de Kaggle, así como los de Google Colab.

La librería empleada fue *Unsloth* (*fine-tuning* con QLoRA), que se integra a su vez con otras librerías como *transformers* (LLM), *bitsandbytes* (cuantización) y *torch* (aprendizaje automático general).

4.3. Resultados obtenidos

Nuestros resultados se basan en una evaluación sobre un 5% del total de muestras presentes en nuestro conjunto base, habiendo diseñado la partición de forma que se incluya esta proporción tanto de muestras de *La longue marche à travers la théorie de Galois* como de cartas, resultando en 31 páginas. Naturalmente este subconjunto de nuestras muestras no se incluyeron en el conjunto de entrenamiento, de forma que sean realmente representativas de la capacidad de nuestro modelo.

Puesto que existen diferentes métodos de inferencia en un solo modelo, se emplearán varias de ellas, escogiéndose siempre aquella que obtenga un mejor desempeño para realizar comparaciones entre modelos. Bajo este estándar, empleando un rango de LoRA de 8 solo en las capas de lenguaje y usando de forma exclusiva nuestro conjunto de entrenamiento, nuestro modelo obtiene una mejora en el CER de un 62% con respecto al modelo base (Qwen3-VL-8B-Instruct).

4.3.1. Modelo base

El modelo de partida tiene capacidad nativa para transcripción, puesto que es una de las tareas para las que fue entrenado [21]. Para obtener una métrica de esta capacidad de transcripción nativa, realizamos inferencia sobre una serie de muestras sintéticas de texto plano y texto matemático con LaTeX, estructuradas en dos conjuntos de datos sintéticos:

Intro-text Un conjunto de datos de texto plano renderizado en LaTeX, de longitud variable²⁶.

en el mundo y su experiencia con él

toma la decisión de cortar todo lazo con la sociedad: limpió el que había sido su hogar, quemó gran

puede ser más o menos preciso, pero también el problema del

Figura 24. Tres de los 303 ejemplos con los que cuenta este pequeño conjunto de datos de evaluación.

Mixed-math Otro conjunto de datos, con texto y matemáticas varias (aunque sin contenido semántico ni continuidad), con su equivalente en LaTeX.²⁷

Entonces $|a_n - L| < \varepsilon$ para $n > N$.

Sea $\mathcal{H}$ un espacio de Hilbert.

Tomamos $\varphi \in \mathcal{H}$ con $\|\varphi\| = 1$.

Figura 25. Tres de los 50 ejemplos con los que cuenta el conjunto de datos de evaluación de matemáticas.

Los resultados del modelo base, empleando generación mediante 5 haces, son:

Dataset	$\overline{\text{CER}}$	WCER
Texto plano	0.0033	0.0037
Texto mixto	0.1190	0.1162

Tabla 1. Resultados del modelo base sobre los conjuntos de datos sintéticos.

Debido a la altísima precisión del modelo base en el texto plano, resulta ilustrativo evaluar únicamente aquellos errores que comete sobre el conjunto de texto mixto. Denotando por **R** el texto de referencia y **P** a la predicción del modelo, tenemos:

R Sea $\Gamma^2_1 + \delta = \mathcal{M}$.

P Sea $\Gamma_1^2 + \delta = \mathcal{M}$.

R Definimos $\varphi_n(x) = \sum_{k=0}^n \alpha_k x^k$.

P Definimos $\varphi_n(x) = \sum_{k=0}^n \alpha_k x^k$.

²⁶[Enlace al dataset completo de texto plano.](#)

²⁷[Enlace al dataset completo de texto mixto..](#)

R En particular, $\Delta = \beta^2 - 4\alpha\gamma$.

P En particular, $\Delta = \beta^2 - 4\alpha\gamma$.

Notamos, además, que el error aparente de 11.90% en $\overline{\text{CER}}$ y 11.62% en WCER es, en realidad, un espejismo causado por la falta de alineación entre la métrica base, CER, y la sintaxis de LaTeX²⁸. El resultado que produce el modelo es, de hecho, una reproducción visual exacta del original:

Referencia Sea $\Gamma_1^2 + \delta = \mathcal{M}$.

Predicción Sea $\Gamma_1^2 + \delta = \mathcal{M}$.

Referencia Definimos $\varphi_n(x) = \sum_{k=0}^n \alpha_k x^k$.

Predicción Definimos $\varphi_n(x) = \sum_{k=0}^n \alpha_k x^k$.

Referencia En particular, $\Delta = \beta^2 - 4\alpha\gamma$.

Predicción En particular, $\Delta = \beta^2 - 4\alpha\gamma$.

Partimos, por tanto, de un modelo cuya capacidad de transcripción de texto matemático y texto plano es ya excelente.

En base a esto, y para partir de una métrica aproximada de la dificultad de la tarea que enfrentamos para los modelos, calculamos el CER de las predicciones de nuestro modelo sobre el conjunto de evaluación. Está compuesto de páginas manuscritas de Grothendieck, y, en concreto, estas métricas se calcularán únicamente sobre las muestras de 1, 2 y 3 líneas.

²⁸Se discute este hecho en el [Apartado 5.2.1.2](#) de las conclusiones.

Haces, Contexto	$\overline{\text{CER}}$	WCER
Sí, sí	0.6265	0.6201
Sí, no	0.5631	0.5825
No, sí	0.8810	0.7617
No, no	0.6297	0.6357

Tabla 2. Resultados del modelo base sobre nuestro conjunto de evaluación.

Observamos que los resultados se degradan enormemente con respecto al conjunto de datos sintéticos, siendo claro un enorme aumento en la dificultad de la tarea propuesta.

4.3.2. Modelo entrenado

Habiendo realizado el entrenamiento correspondiente, siguiendo las decisiones expuestas con anterioridad, se tomó el modelo con mejores resultados en el conjunto de evaluación de la última etapa del currículum. Realizando inferencia sobre el mismo conjunto de evaluación, los resultados obtenidos son los siguientes:

Haces, Contexto	$\overline{\text{CER}}$	WCER
Sí, sí	0.2236	0.2493
Sí, no	0.2157	0.2325
No, sí	0.2425	0.2610
No, no	0.2483	0.2594

Tabla 3. Resultados del modelo extraído del *checkpoint* 1200 sobre nuestro conjunto de evaluación.

Las mejoras tanto en $\overline{\text{CER}}$ como en WCER son evidentes en todos los casos: la mejor configuración (generación de 5 haces sin contexto) consigue un CER de 0.2157, lo que supone una reducción del error de un 61.69%. En el caso del WCER, la mejora es del 60.09%.

En estos resultados queda patente, además, otra característica del modelo: pese a que durante el entrenamiento obtenemos pérdidas más bajas en las muestras contextualizadas que en aquellas sin contextualizar, la inferencia no sigue este patrón. El modelo obtiene resultados que no solo son equiparables, sino peores, si se contextualizan con contenido textual las muestras. Esto puede deberse a que el modelo está basando su respuesta *demasiado* en esta evidencia textual, obviando parcialmente la información visual, en un proceso de dilución de la atención [54]. Es un fenómeno conocido en LLM que, en ocasiones, los modelos priorizan la generación de texto coherente a la transcripción, lo que en este caso se materializa como el modelo alucinando contenido, probablemente debido a la dificultad del texto [55].

Extraemos de la comparativa en resultados, además, que la búsqueda en haces demuestra ser indispensable para estabilizar la transcripción, mitigando las divergencias observadas en estrategias puramente codiciosas.

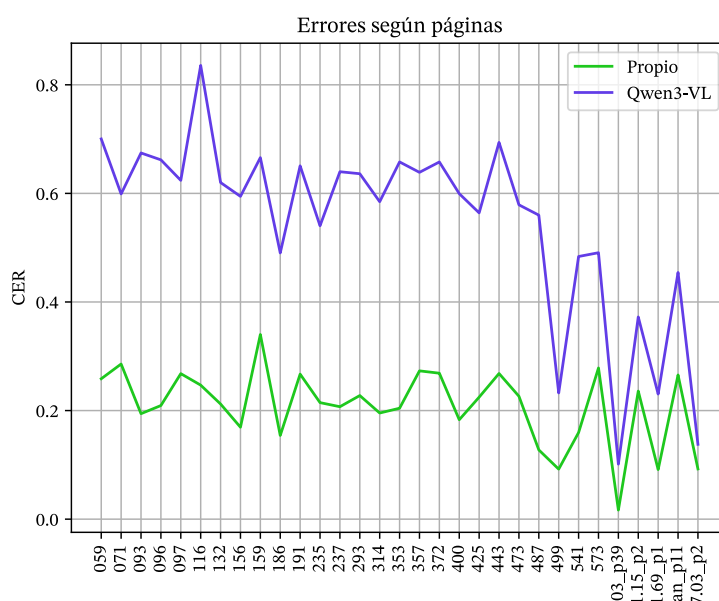


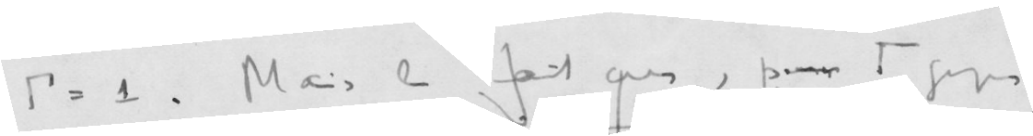
Figura 26. CER promedio del modelo base y el modelo entrenado en cada página del conjunto de validación.

Analizando los errores que comete el modelo en cada página, y comparándolos con los del modelo base, podemos notar además una alta correlación en la forma en que se distribuyen los errores: el modelo original y el modelo afinado parecen coincidir en aquellas páginas que consideran «fáciles» o «difíciles» en comparación al resto. Esto evidencia que nuestro proceso de ajuste fino verdaderamente

ha conseguido realizar *transfer learning*: el modelo ha aprendido, mejorando su interpretación de las características visuales de forma aproximadamente homogénea en todas las páginas, manteniendo las páginas su dificultad relativa para ambos modelos.

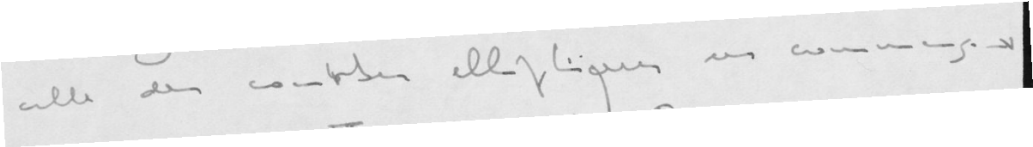
4.3.2.1. Ejemplos de generación

De forma ilustrativa, como método de comparación del modelo, se muestran a continuación predicciones de 3 muestras concretas monolínea. Denotando por **R** al texto de referencia (las transcripciones correctas) y como **P** a la predicción del modelo:



R $\Gamma = 1$. Mais le fait que, pour Γ groupe

P $\Gamma = 1$. Mais le fait que Γ soit



R celui des courbes elliptiques en commençant

P celle des courbes elliptiques en commutant



R de $\mathcal{M}_{1,1}'$, $\mathcal{T}_{1,1}'$

P des $\mathcal{M}^{!}_{g,n}$, $\hat{\mathcal{T}}^{!}$

Los resultados siguen a grandes rasgos lo esperable: el modelo es más propenso a cometer errores en muestras con rotaciones altas, en muestras con muchas matemáticas, emborronadas o, en general, poco legibles.

4.3.2.2. Ejemplos de generación sin matemáticas

Pese a que entre sus documentos en Lasserre hay presente texto matemático, nuestro documento objetivo, *El problema del Mal*, es un texto de otra índole y, por tanto, el modelo se estaría probando en una tarea más sencilla. Debido a la complejidad añadida que supone evaluar el modelo con contenido matemático, una métrica más fiable de la capacidad de transcripción de nuestro modelo en este caso de uso concreto es comprobar su efectividad sobre aquellas muestras de nuestro conjunto de evaluación que no presentan matemáticas.

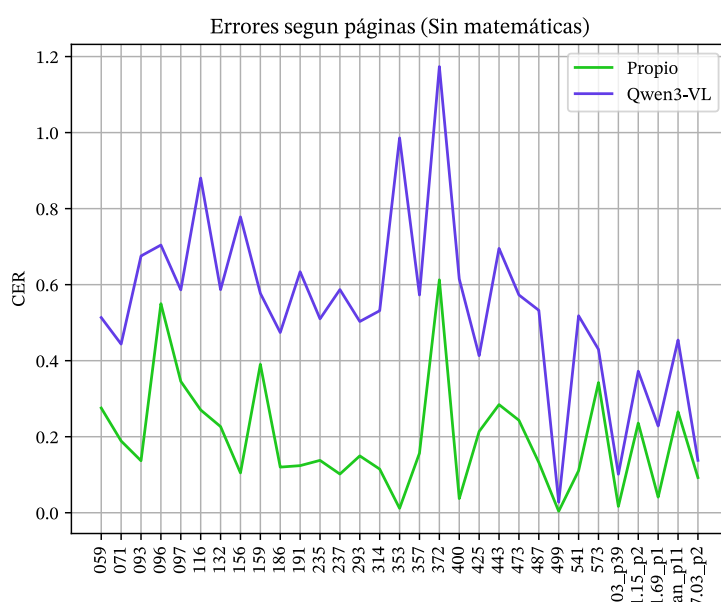


Figura 30. CER promedio del modelo base y el modelo entrenado en cada página del conjunto de validación, retirando las muestras con matemáticas.

En este caso, el $\overline{\text{CER}}$ se reduce hasta 0.178, lo que supone una reducción del 58.89% frente al modelo base, demostrando una superioridad sustancial frente al modelo base también en esta tarea, más cercana al problema del mal.

Sabemos, además, que las cartas no son una muestra representativa del tipo de texto al que se enfrentará nuestro modelo. Restringiendo aún más nuestro conjunto de validación a aquellas muestras donde la línea no procede de una

carta, llegamos a un WCER de un 20.65%, mientras el del modelo base es de un 56.92% sobre estas mismas muestras. Esta diferencia supone una mejora de un 63.72% frente al modelo base..

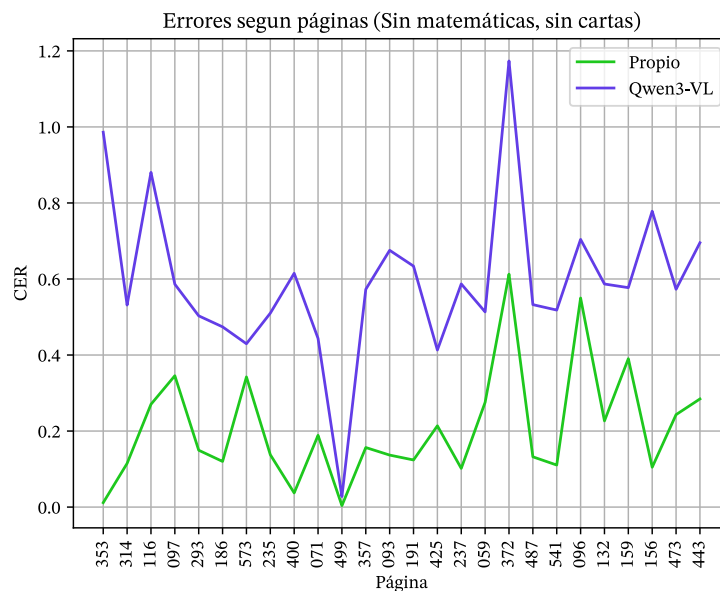
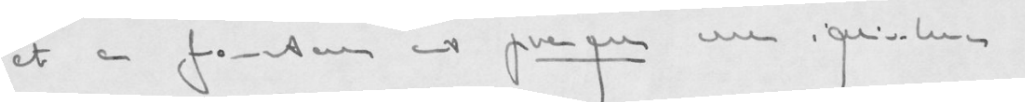


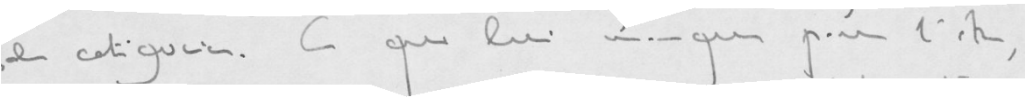
Figura 31. CER promedio del modelo base y el modelo entrenado en cada página del conjunto de validación, retirando las muestras con matemáticas y las cartas.

Se presenta a continuación la transcripción de todas las líneas no matemáticas de una página completa, escogida por ser la de mayor longitud:



R et ce foncteur est presque une équivalence

P et ce foncteur est quelque une équivalence



R de catégories. Ce qui lui manque pour l'être,

P de catégorie. Ce qui lui manque pour l'être,

trivial - opère trivialement sur l'objet corres

R trivial - opère trivialement sur l'objet corres

P trivial - opère trivialement sur l'objet corres

celle des courbes elliptiques en commençant

R celui des courbes elliptiques en commençant

P celle des courbes elliptiques en commutant

puis on prendra le champ associé à un

R puis on prendra le champ associé à un

P puis en prenant le champ associé à ce

"courbes elliptiques relatives à symétrie près sur

R "courbes elliptiques relatives à symétrie près sur

P courbes elliptiques tel E_i sur $\mathbb{F}_i$ symétries près sur

ayant comme groupe

R ayant comme groupe

P ayant comme groupe

ment, d'identité des p.c. des courbes

R ment, déduit par exemple des variétés

P ment, l'identité des pôles des courbes

modulaires : rigidification de Jacobi

R modulaires à rigidification de Jacobi

P modulaires à rigidification de Jacobi

5.

Conclusiones y trabajo futuro

La digitalización y preservación de la obra tardía de Alexander Grothendieck constituye un colosal desafío paleográfico y técnico. La naturaleza inextricable de su caligrafía manuscrita, combinada con una elevada densidad conceptual, en una obra donde se funde reflexión filosófica con matemáticas puras, sitúa este corpus fuera del alcance de las soluciones de OCR convencionales y de los modelos comerciales generalistas. A este obstáculo se añade un estricto imperativo legal y ético: la imposibilidad de transferir las imágenes del manuscrito a servidores externos privativos, obligando al despliegue de una arquitectura computacional estrictamente local.

El presente Trabajo Fin de Grado ha demostrado la viabilidad técnica de salvar esta brecha mediante el ajuste fino de grandes modelos de lenguaje visual de pesos abiertos. A través del diseño e implementación de un flujo integral (que abarca desde la segmentación y homogeneización de un *dataset* de 11 500 líneas hasta la articulación de infraestructura y estrategias de adaptación de bajo rango y aprendizaje por currículum) se ha logrado dotar a una red neuronal de la capacidad de descifrar este dominio caligráfico tan altamente especializado. En este capítulo pretendemos sintetizar los hallazgos empíricos más importantes de la evaluación del modelo, así como postular líneas de investigación futuras para avanzar hacia la digitalización sistemática de la obra de Lasserre.

5.1. Conclusiones generales y análisis del rendimiento

La evaluación cruzada entre el modelo base y el ajustado proporciona información fundamental sobre la dinámica de adaptación a datos que, como es nuestro caso, son de naturaleza multimodal.

El modelo base exhibe una alta competencia para comprender y estructurar la sintaxis de LaTeX sobre nuestros datos sintéticos (texto sencillo y claro), registrando un $\overline{\text{CER}}$ de 0.003 en texto plano y 0.119 en texto mixto. Sin embargo, este rendimiento colapsa al enfrentarse a los manuscritos reales de Grothendieck, obteniéndose un $\overline{\text{CER}}$ óptimo de 0.5631.

El proceso de ajuste fino sobre las capas de lenguaje mediante un rango 8 en LoRA ha demostrado ser una estrategia efectiva para solventar la falta de alineamiento multimodal. El mejor modelo alcanzado durante los entrenamientos alcanza un $\overline{\text{CER}}$ de 0.2157 y un WCER de 0.2325 (generación basada en búsqueda de 5 haces sin contextualización previa). Esto representa una mejora de un 61.69% para el $\overline{\text{CER}}$ y del 60.09% para el WCER.

Estos resultados, aunque lejos de perfectos, validan la hipótesis de partida de este trabajo: el tipo de entrenamiento usado, realizado sobre el conjunto de datos confeccionado y empleando la sistemática expuesta es viable técnicamente y consigue mejorar considerablemente los resultados. Esto demuestra empíricamente la apertura de una nueva vía hacia la digitalización de la obra de Lasserre, con multitud de opciones disponibles para continuar avanzando.

Cabe destacar, además, que, como ejemplifica la evaluación en datos sintéticos, el *error visual real* tiende a ser considerablemente inferior al CER obtenido. Esto se debe a la combinación de la ineficiente sintaxis de LaTeX con la distancia de Levenshtein. Esta distancia (ideada para códigos y óptima en el caso de transcripciones carácter a carácter de texto plano) penaliza muy severamente incluso pequeñas discrepancias estilísticas inherentes a la sintaxis de LaTeX: el uso de espacios en entornos matemáticos, delimitadores equivalentes o comandos diferentes que producen resultados iguales. Así, emplear esta distancia lleva a un proceso de acumulación de penalizaciones que no se corresponden con alteraciones en la semántica o en el resultado visual final de la transcripción.

5.1.1. Aplicabilidad

El desglose del desempeño de nuestro modelo ajustado, realizado empleando subconjuntos de datos, nos informa de la posible utilidad práctica en el corpus objetivo. En primer lugar, al evaluar nuestro modelo ajustado sobre el subconjunto de muestras exentas de notación matemática, el error desciende hasta un 17.80%, desde un 43.33%. Dado que la obra monumental de Lasserre, *El problema del Mal* (40 000 páginas), carece de matemáticas, esta métrica sitúa al modelo en un umbral de precisión funcional para la automatización de su digitalización. Por otro lado, al restringir la validación a líneas que no procedían de correspondencia privada, el modelo afinado presenta un CER de un 20.65%, lo que supone una mejora de un 63.72% frente al modelo base.

Finalmente, y de forma crucial, se obtienen del modelo transcripciones extremadamente correctas de páginas de este tipo, pudiendo servir como pre-transcripción. Esto es apreciable en la transcripción de todas las líneas de una página presentadas en [Apartado 4.3.2.2](#). Queda patente, una vez más, que la red ha abstraído patrones gráficos generales de su caligrafía y no únicamente regularidades superficiales de escritos más cuidados, además de la viabilidad de estos esquemas de entrenamiento para este caso de uso concreto.

5.1.2. Hallazgos para el proceso de inferencia

Los resultados muestran, en primer lugar, la necesidad de emplear búsqueda en haces: resulta un componente en las mejores métricas de nuestros modelos, estabilizando el proceso de inferencia y evitando alucinaciones y errores en cadena en nuestro conjunto de validación. Supera en todas las métricas a la estrategia codiciosa empleada, que tiende a propagar errores a lo largo de la cadena de texto de LaTeX.

Resulta contraintuitiva, al menos *a priori*, la inutilidad de la contextualización (el CER promedio empeora un 0.79% en búsqueda en haces con respecto a no contextualizar la muestra). Esta diferencia sugiere dos posibles interpretaciones:

1. La introducción de contenido textual sesga nuestro modelo y provoca un fenómeno de dilución de la atención hacia los tokens visuales, donde el modelo prioriza contenido que considera más «correcto» semánticamente que la adecuación al contexto visual. [\[54\]](#), [\[55\]](#)

2. La sistemática de entrenamiento basada en contexto no se ha calibrado correctamente (por ejemplo con el parámetro de probabilidad), lo que ha llevado a que el modelo no abstraiga correctamente los patrones de este tipo de muestras.

5.2. Posibles trabajos futuros

La mayor restricción durante el desarrollo del presente Trabajo Fin de Grado ha sido temporal. Debido al enorme volumen de tareas, la necesidad de desplegar proyectos en servidores externos, compatibilizar flujos de trabajo entre diferentes sistemas (cuadernillos de Kaggle, Colab y el servidor de entrenamiento) y la cantidad de tiempo requerido para el *fine-tuning* del modelo, muchas de las ideas que surgieron durante su desarrollo no pudieron implementarse.

A continuación, se exponen algunas de ellas, surgidas en diferentes estadios del proceso, relativas al entrenamiento, la infraestructura, el paradigma de inferencia, el procesado de las anotaciones y otras partes del proceso.

5.2.1. Modelo, paradigma y evaluación

Nuestro modelo base, la métrica que empleamos para escoger el mejor de ellos y los paradigmas de entrenamiento empleados son piezas fundamentales para la mejora del modelo.

5.2.1.1. Aprendizaje por refuerzo mediante GRPO

La entropía cruzada, métrica empleada durante el entrenamiento, no representa correctamente el *error visual* o de transcripción. Esto hace que nuestro modelo se entrene en una representación demasiado rígida en la que se penaliza de igual forma confundir dos tókenes visualmente muy similares que dos tókenes visualmente muy diferentes.

El problema de las métricas visuales es que no son diferenciables, y nuestro modelo ya ha sido entrenado empleando entropía cruzada. En consecuencia, para poder implementarlas requeriríamos otro paradigma de entrenamiento, compatible con este tipo de modelos.

Una alternativa es el entrenamiento por refuerzo usando *Group Relative Policy Optimization*. En este tipo de entrenamiento, el modelo genera múltiples

transcripciones para una misma imagen, reforzándose activaciones que llevan a buenas transcripciones y penalizándose el resto.

La métrica mediante las que se separan las «buenas» transcripciones de las «malas» puede ser extremadamente rica, pudiendo tener en cuenta factores como:

- Corrección visual de la transcripción del texto (mediante CER).
- Corrección visual de las fórmulas (mediante algún sistema basado en compilación del LaTeX o basado en el árbol de sintaxis abstracta).
- Corrección sintáctica del contenido de LaTeX (todos los comandos compilan, se cierran los entornos que se abren).
- Se infiere correctamente el número de elementos de matemáticas presente (mismo número de \$ en la transcripción que el original).

Y otras adaptaciones a la métrica, lo que conecta con el siguiente apartado de trabajo futuro, [Apartado 5.2.1.2](#), sobre elección de métricas.

Además, usar GRPO solventa otra de las deficiencias inherentes a los ajustes finos: el modelo no se expone a su propia generación, por lo que no tiene la posibilidad de arrastrar sus errores, lo que lleva a que este tipo de comportamientos no se corrijan. Así, mediante este tipo de aprendizaje por refuerzo el modelo aprende a «recuperarse» de fallos anteriores que él mismo ha cometido, favoreciendo transcripciones más limpias y con menos errores catastróficos.

Este tipo de aprendizaje por refuerzo mejora significativamente el resultado de los modelos [55], [56], pero debe partir de un modelo con una capacidad de transcripción aceptable. Este paradigma, por tanto, no sustituye el ajuste fino, sino que lo complementa: sería necesario partir de un modelo como el nuestro para poder realizarlo, potenciándose su capacidad de transcripción.

5.2.1.2. Alternativas al CER

Ya hemos discutido en [Sección 3.6](#) la decisión de tomar como métrica el CER habitual en lugar del TER (*Token Error Rate*) dado por nuestro modelo. Nuestro conjunto de datos, sin embargo, por contar con tan abundante proporción de matemáticas, podría no prestarse correctamente a esta métrica.

Nuestro modelo puede haber aprendido correctamente la representación,

(Manuscrito $\rightarrow$ Transcripción)

$\varepsilon \rightarrow \backslash\text{varepsilon}$

$\rho \rightarrow \backslash\text{rho}$

$\kappa \rightarrow \backslash\text{kappa}$,

sin embargo es habitual, debido a la caligrafía de Grothendieck, que el símbolo concreto sea dudoso. Pueden por tanto confundirse estos símbolos multicarácter con otros (en los ejemplos anteriores, con E, p, k) con relativa facilidad. En estos casos estamos sobrestimando el «error visual» cometido por el modelo por mucho: en lugar de considerar que se ha equivocado interpretando un símbolo, penalizamos su predicción como si hubiera interpretado un solo símbolo como una palabra completa. Este tipo de errores, que serían marcadores de una codificación inaceptable por parte de un modelo de transcripción, proceden realmente de las particularidades sintácticas de la escritura a LaTeX, y se corresponden más con una falta de alineamiento entre métrica y lenguaje que estamos arrastrando a nuestros resultados que con fallos del modelo.

Podría diseñarse, por tanto, un CER calibrado en el que todo comando íntegro de LaTeX que renderice como un solo símbolo se contabilice como un solo símbolo ($\backslash\text{varepsilon}$, $\overline{C}$ ²⁹, $\uparrow$). Este «CER tras renderizar a LaTeX» sería, en base a lo expuesto, una métrica de mayor calidad que el CER habitual.

Esto podría complementarse con un cambio en el tipo de tokenizador empleado, para favorecer el aprendizaje de la sintaxis de LaTeX en el caso concreto de los documentos de Grothendieck.

Alternativamente, en línea con sistemas de evaluación más típicos (pero también menos especializados), se podrían incorporar métricas de error más avanzadas, como combinaciones entre el CER, WER (*word error rate*), BoW-WER (*bag of words-word error rate*), intentando modular cómo se combinan para paliar la penalización excesiva al orden que se obtiene con CER [57]. Estas alternativas, sin embargo, serán siempre incompletas en esta tarea específica, por las limitaciones ya expuestas en torno a la sintaxis de LaTeX.

²⁹En este caso concreto podría contar como dos diferentes. Tal vez $\overline{C}$ y el argumento C o $\{C\}$, pues es visualmente comprensible que el modelo interprete correctamente la presencia de la C y no vea el gorro, y este caso concreto tal vez debería contabilizarse diferente de si el modelo ha, por ejemplo, predicho una T donde hay una $\hat{C}$.

Otra alternativa posible es definir el error como una ponderación³⁰ de dos tipos de errores: el error textual (que podría mantenerse como el CER) y el error en fórmulas, que podría conseguirse a partir de un árbol de sintaxis abstracta realizado sobre el código LaTeX, o basado en el renderizado de las fórmulas, como se realiza en otros trabajos de transcripción matemática [55], [58].

5.2.1.3. Preentrenamiento con textos de Grothendieck

Nuestro modelo aprende a realizar simultáneamente dos tareas diferentes durante su entrenamiento: por un lado, el modelo aprende a transcribir la caligrafía de Grothendieck y, por otro, el modelo se acostumbra a la temática que trata y su forma particular de escribir. Puesto que nuestros modelos de lenguaje son probabilísticos, el modelo puede estar sobreadaptándose al vocabulario presente en el conjunto de entrenamiento, tras lo que podría no generalizar bien a otros documentos con otras temáticas.

Para paliar esto, una posible solución es separar estos dos aprendizajes de forma más específica: en lugar de hacer un fine-tuning puramente visual, este puede acompañarse de uno textual que parta de los textos de Grothendieck, en el que el modelo aprenda sus formas características de expresarse y aumente su conocimiento sobre los temas que trata en su obra. Esto puede realizarse de dos formas:

- Incluir que cada lote del entrenamiento contenga un porcentaje de muestras de tipo textual. Así, el modelo se acostumbra simultáneamente a ambas tareas.
- Realizar un fine-tuning puramente textual y, a partir de este modelo *grothendieckificado*, ajustarlo para transcripción. De esta forma el modelo «base» de este segundo fine-tuning ya está familiarizado con la escritura de Grothendieck y puede limitarse a alinear su interpretación de las características visuales.

El conjunto de datos para el entrenamiento textual se extraería, naturalmente, de su vasta obra pública (*Récoltes et Semailles*, *La Clef des Songes*, *Les Dérivateurs*, sus artículos y conferencias, etc.). El contenido textual podría extraerse o bien del LaTeX (si es accesible, como es relativamente habitual en *arXiv*) o

³⁰Esta ponderación podría ser estática o dinámica: o bien se asigna una cierta puntuación fija a ambos pesos (por ejemplo 50-50), o se calcula en función del peso visual de cada tipo de bloque en las muestras.

de los PDFs de estos enormes documentos. Esta última tarea podría realizarse empleado un OCR para documentos mecanografiados (como OlmoOCR-2, excelente en estos casos) o bien una herramienta como [Markitdown](#), dependiendo de si el PDF contiene o no información textual.

Este es un campo, además, poco explorado en la literatura del OCR³¹. En general, en OCR mediante LLM se asume (aunque sea de forma implícita) que el texto presente en los documentos sobre a transcribir sigue una distribución muy similar a la del texto de preentrenamiento del modelo, pues se pretende un producto de OCR «generalista». Nuestro caso, por otro lado, es más similar a un proceso industrial altamente especializado, pues el lenguaje presente en estos documentos es bastante concreto, reducido y poco común. Se trata de una tarea poco explorada, en parte porque rara vez existe justificación para entrenar un modelo en la transcripción de documentos de un único individuo. Incluso cuando es el caso, es también poco común tener acceso a un corpus textual suficientemente grande como para poder entrenar al modelo en la forma concreta de expresarse y el lenguaje preferido del susodicho individuo.

5.2.1.4. Contextualización con imágenes

Nuestra contextualización estocástica se ha realizado puramente con información textual, lo que resulta útil para [Apartado 5.2.4.1](#), pero su aplicación sin esta parte del trabajo pasa necesariamente por introducir (semi)manualmente en el modelo el contexto que ha producido con anterioridad.

Este tipo de contextualización hace además al modelo, hasta cierto punto, más propenso a errores en cadena. Si comienza a desviarse demasiado del contenido, un texto anterior incoherente o completamente desconectado del contenido real de una línea impacta en la generación de líneas siguientes. Este, sin embargo, no es el caso si la contextualización está plenamente asida en la realidad del manuscrito, como puede ser contextualizar al modelo empleando información visual de líneas anteriores.

Sean I_t y T_t respectivamente la imagen y la transcripción de la línea t -ésima, y sea $(X : Y)$ un par donde X es la información (con algún tipo de formato) suministrada al modelo e Y la predicción querida. Proponemos, entonces, en lugar de entrenar al modelo con muestras del tipo $(I_t : T_t)$ e $(I_t, T_{t-1} : T_t)$,

³¹Aunque podría considerarse un caso especial de *domain-adaptive pre-training*.

añadir también muestras del tipo $(I_t, I_{t-1} : T_t)$ o incluso $(I_t, I_{t-1}, T_{t-1} : T_t)$ haciendo que el modelo se acostumbre a extraer información contextual visual³², haciéndolo más robusto.

Este tipo de contextualización es similar a la que se realiza en OlmoOCR-2, donde se añade información extraída de la propia página mediante un preprocesado [56]. Ambas se diferencian de la contextualización mediante texto en que el modelo se instruye específicamente para «tener en cuenta» esta información, no para continuar el texto. Esto puede ayudar en el proceso de dilución de la atención observado cuando las imágenes se contextualizan con el texto anterior.

5.2.1.5. Cambio de modelo

Los modelos de lenguaje de pesos abiertos evolucionan increíblemente rápido. Solo desde el inicio de este Trabajo Fin de Grado, cuando se escogió uno de los modelos estado del arte, han surgido dos nuevos de su misma serie, a saber Qwen3.5 y Qwen3.6. Es natural, por tanto, esperar que el desempeño de estos dos modelos sea superior³³ al de este modelo, precisamente por su novedad.

De forma similar a nuestro enfoque, olmoOCR se presenta como una variante de Qwen2-VL (o Qwen2.5-VL, dependiendo de la versión de olmoOCR) entrenada con el propósito específico de transcribir documentos. Resulta, por tanto, razonable asumir que un *fine-tuning* sobre este modelo en lugar de un Qwen base podría resultar en mejores resultados.

Existen otros muchos modelos de visión que podrían haberse empleado en lugar de Qwen3-VL, que se escogió en base a un estudio provisional inicial sobre precisión en el modelo base. Como candidatos se emplearon modelos de [Llama](#), [Mistral](#) y otros Qwen anteriores, pero existen multitud de otros modelos.

5.2.1.6. Parámetros

Debido a las limitaciones temporales, no se ha podido estudiar con detenimiento el plan óptimo de entrenamiento. En consecuencia, existe posibilidad de mejora en muchos de los parámetros:

³²El modelo, hasta cierto punto, ya está realizando este proceso, pero hacia delante. Esto se debe a que el modelo atiende a toda la información visual que recibe, de la línea completa, ya para generar el primer token de la transcripción.

³³Con los ajustes necesarios al entrenamiento.

- Rango de los adaptadores y penalización de pesos de LoRA.
- Ratio de aprendizaje, esquema de penalización de la tasa de aprendizaje.
- Decaimiento de pesos.
- Regularización, etc.

5.2.2. Tratamiento de las anotaciones

La interfaz y el método de anotación se desarrollaron y pulieron de forma progresiva mediante la retroalimentación de los anotadores y necesidades del modelo y código de ingesta. Debido a este crecimiento orgánico, algunas de las ideas propuestas no pudieron implementarse, o se consideraron poco rentables en tiempo. Se exponen en esta sección algunas de las ideas relativas a esta parte del desarrollo de este Trabajo Fin de Grado.

5.2.2.1. Separación automática de párrafos entrelazados

Se discute el planteamiento de este problema en el [Apartado A.3.4.3](#).

En la implementación actual, cuando no es posible separar dos párrafos que se encuentran entrelazados durante la anotación se deben dividir en dos anotaciones independientes. Este proceso, aunque sencillo de realizar, dificulta ligeramente la labor del anotador y los revisores.

Para resolverse, podría realizarse un procesado de los párrafos en el que, empleando algoritmos de *clustering* usando la rotación de cada línea y sus coordenadas mínimas en x (dónde empieza horizontalmente la línea), se obtenga una puntuación (por ejemplo la de silueta) para diferente número de grupos. Tomando el agrupamiento que produzca una mejor puntuación, se ejecuta una separación automática de este párrafo original en varios «párrafos efectivos» (cada grupo de líneas dado por el *clustering*) para la generación.

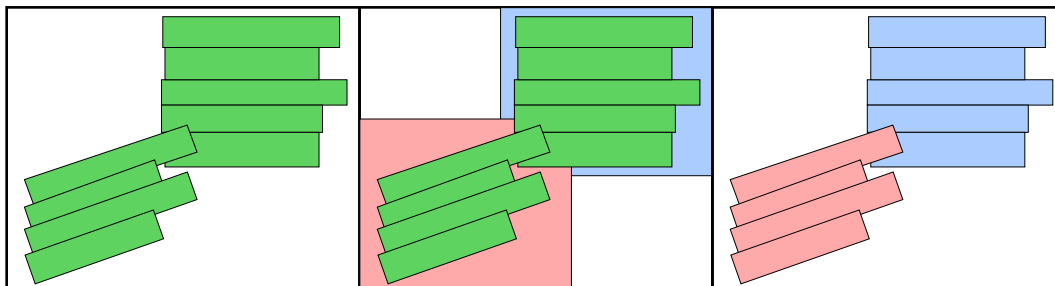


Figura 32. Separación de un párrafo único en dos párrafos efectivos mediante un algoritmo de clustering. Se calcularía la calidad de diferentes agrupamientos, cambiando el número de grupos, para decidir si se divide o no el párrafo, y en qué subpárrafos.

Esto permitiría circunvalar la separación en tareas diferentes, realizada actualmente, y permitiría un orden de lectura más próximo al pretendido por el propio Grothendieck.

5.2.2.2. Semiautomatización de la expansión del dataset

Existen diferentes algoritmos de segmentación de imágenes (como YOLO [59], Surya, Docling [60]...), permitiendo incluso algunos de ellos un ajuste fino de sus parámetros para ajustarlos a nuestra tarea concreta. Este es el caso con muchas versiones de YOLO, que podríamos ajustar empleando las anotaciones visuales que ya tenemos en nuestro conjunto de datos³⁴, de forma que el criterio se ajuste a la composición de textos concreta de Grothendieck y nuestra intención al anotar. Empleando tales modelos como *presegmentadores* de las imágenes, la tarea de los anotadores en una hipotética expansión del conjunto de datos se simplificaría enormemente. En lugar de partir de la imagen plana, podrían partir de una (buena) aproximación de las cajas-imagen.

Por otro lado, para la transcripción sería posible emplear el modelo ya entrenado para generar una *pretranscripción*. El anotador tendría entonces la tarea de corregir los cambios necesarios en transcripción y segmentación, lo que se supone más rápido que realizar la anotación y transcripción desde cero. Así, el proceso de expansión del conjunto de datos resultaría considerablemente más sencillo, pudiendo extenderse a la totalidad de *La longue marche à travers la théorie de Galois* hasta completar la transcripción de toda la obra, y mejorando futuros entrenamientos en consecuencia.

³⁴De forma relativamente barata y rápida, pues son modelos medianos.

5.2.3. Los datos y transformaciones

La variedad en el conjunto de datos es una de las características más importantes para conseguir un buen ajuste fino. Nuestro conjunto de datos, aunque razonablemente variado de base, implementa una amplia variedad de transformaciones posibles a las imágenes, admitiendo cada una de estas transformaciones una serie de parámetros que determinan su *intensidad*.

Estos parámetros, así como otros tipos de transformaciones aplicables al conjunto de entrenamiento, son un campo por explorar, y se discuten a continuación algunas de las opciones abiertas. Resulta de interés, en general, emplear más variedad (por ejemplo con transformaciones más agresivas y datos en entrenamiento más «ruidosos») en aras de evitar el sobreajuste.

5.2.3.1. Variaciones en imágenes, prompt y contexto

En imágenes ya contamos con transformaciones como zooms, traslaciones, deformaciones y rotaciones. Mientras que en OCR habituales estos parámetros se escogen de forma que las transformaciones aplicadas sean relativamente suaves, nuestro objetivo es un texto profundamente distorsionado y rotado. En consecuencia, el modelo debe ser extremadamente resistente a estas deformaciones.

En concreto, la implementación de rotaciones más agresivas es una alternativa que parece especialmente evidente. Al existir líneas con rotaciones extremas, un modelo bien entrenado y generalizado debería ser capaz de leer estas líneas en toda una variedad de ángulos. Se plantea, en consecuencia, un sistema de rotación que lo tenga en cuenta: sean $r_{\min}$ y $r_{\max}$ dos parámetros,

```
procedure get_rotation_intervals(sample,  $r_{\min} > 0$ ,  $r_{\max} > 0$ ):
1   $\theta \leftarrow \text{sample.avg\_rot}$ 
2   $m \leftarrow \text{abs}(\theta)$ 
3  if  $m < r_{\min}$  then:
4    return  $-r_{\min}, r_{\min}$ 
5  elif  $\theta < r_{\max}/2$  then:
6    if  $\theta < 0$  then:
7      return  $0, 2m$ 
8    else:
9      return  $-2m, 0$ 
```

```

10 | else:
11 |   if  $\theta < 0$  then:
12 |     | return  $0, 2r_{\max}$ 
13 |   else:
14 |     | return  $-2r_{\max}, 0$ 

```

Esto permitiría introducir transformaciones más agresivas, respetando la rotación que ya presentan. Con esta introducción de transformaciones más agresivas, sin embargo, sería necesario realizar también un estudio de la cantidad de ruido que añade la interpolación (que, en nuestro código, es bicúbica).³⁵

Otra de las ideas para la transformación del conjunto de datos durante el entrenamiento fue de *descanonización* del texto, de corrupción controlada, para ofrecer más variedad. Se podrían, por ejemplo, sustituir expresiones por otras, cambiar algunos símbolos por otros parecidos (`\to` por `\rightarrow` o `\longrightarrow`). Esto añadiría más variedad a nuestro texto, lo que tal vez permitiese realizar más iteraciones sin sobreajuste.

Finalmente, aunque implementado parcialmente en la adición de contexto, modificar el prompt usado entre ejemplos de entrenamiento contribuiría a la robustez del modelo, pues lo forzaría a funcionar con instrucciones similares en contenido pero no en forma, haciendo que aprenda más fielmente la tarea manteniendo sus capacidades originales menos degradadas.

5.2.4. Infraestructura y modelo resultante

El modelo resultante podría cargarse y emplearse dentro de un cuadernillo de Jupyter, por ejemplo el usado para realizar inferencia para obtener las métricas de CER del modelo sobre el conjunto de test. Este formato, sin embargo, requiere algo de conocimiento de Python y no es del todo amigable con el usuario, especialmente para trabajar sobre documentos nuevos, pues requerirían un pretratamiento que puede no ser trivial de conseguir. La utilidad del modelo, además, se ve limitada a:

³⁵Este procedimiento de elección de las rotaciones ya se encuentra implementado en nuestro código de transformación, accesible mediante [este repositorio de GitHub](#), sin embargo no llegó a usarse durante el entrenamiento ni, por tanto, tampoco a medirse su impacto.

- que sea capaz de transcribir imágenes completas, las introducidas por los usuarios;
- que sea capaz de transcribir párrafos o unidades más pequeñas, pero que los usuarios deban recortarlas o hacer las fotos en consecuencia;
- o, en caso de no ser capaz, cometer muchos errores.

Se proponen en esta sección diferentes complementos a la estructura del modelo, o estudios realizables a partir del modelo ya entrenado.

5.2.4.1. Segmentación y contextualización automática

De igual forma que hemos explicado en [Apartado 5.2.2.2](#), sería posible entrenar un segmentador para que divida automáticamente la imagen en líneas como las que presenta nuestro dataset. Esto, además, podría servir para un posible proceso de inferencia encadenado más eficiente y con menos propensión a errores.

Este sistema de inferencia tendría entonces dos partes, con 2 conectores:

- Un segmentador entrenado con la forma de segmentar en líneas los documentos.
- El modelo, que recibiría las imágenes línea a línea.

El flujo de información sería el siguiente:

1. Se recibe la imagen en el conector de entrada.
2. Esta imagen pasa por el segmentador, que la divide en líneas.
 - De forma opcional, los polígonos dibujados por el segmentador (a menudo a nivel de píxel) se simplifican en polígonos con menos vértices empleando, por ejemplo, el algoritmo de Ramer-Douglas-Peucker o una simple envolvente convexa.
3. Se calcula el orden de lectura como en base a las regiones generadas por el segmentador [Apartado A.3.4.2](#).
4. Se toman las líneas siguiendo este orden de lectura, suministrando al modelo no solo la línea a transcribir sino también la transcripción de la línea o líneas anteriores, mejorando su precisión y creando un contexto actualizado de forma procedural.
 - Opcionalmente y si la segmentación es suficientemente buena (que es probable), podría emplearse el mismo algoritmo empleado en [Apartado A.3.2.1](#) para calcular la rotación de estas líneas y corregirla antes de la inferencia. Esto previsiblemente aumentaría la precisión del modelo.

5. Se guarda la transcripción generada por el modelo y se repite el punto anterior hasta terminar con la página, tras lo que se procesaría la siguiente.

5.2.4.2. Interfaz de inferencia con variantes

Para simplificar y mejorar la interacción de un usuario con el modelo, se propone desarrollar una interfaz de inferencia más completa, que permita introducir las imágenes de forma menos programática, realice automáticamente el preprocesado, le de formato de conversación y la envíe al modelo.

Esta opción añade, además, la posibilidad a que la tarea del transcriptor se vuelva aproximadamente colaborativa con la del modelo. Existen varias opciones:

1. El transcriptor recibe del modelo una transcripción completa (ya sea hecha línea por línea, empleando el proceso descrito en el apartado anterior, o de la página completa) y se basa en ella para generar la suya propia.
2. El transcriptor produce su transcripción y, cuando duda de una palabra, introduce al modelo el contenido anterior (transcrito por él) así como una imagen de la línea para recibir una sugerencia informada del contenido.
3. El transcriptor y el modelo trabajan simultáneamente: el transcriptor envía una imagen al modelo, que en lugar de comenzar a transcribir envía como resultado aquellos tokens con mayor probabilidad. Tras esto, el anotador escoge el token que considera sigue a la secuencia anterior, basándose en la información visual que también tiene.
4. Similar a la solución anterior, pero el modelo transcribe de forma continua, con la posibilidad de que el transcriptor interrumpa el proceso para añadir una corrección, que el modelo asimila y e incorpora para no arrastrar el error en siguientes inferencias.

5.2.4.3. Recuperación de textos de Grothendieck a partir del modelo

Un fenómeno inherente al entrenamiento de redes neuronales con un número masivo de parámetros sobre conjuntos de datos de tamaño modesto, como es nuestro caso, es la fuerte propensión a la memorización. Como se remarcó durante la [Sección 3.3](#), un modelo de estas dimensiones no tiene dificultad para memorizar de manera exacta cientos o miles de muestras textuales si se

expone a ellas repetidamente. Aunque el objetivo principal del ajuste fino ha sido dotar al modelo de la capacidad de transcribir nuevas imágenes basándose en características visuales, evitando todo lo posible la memorización, los pesos de la red inevitablemente han absorbido parte de esta información (una nueva distribución sobre el léxico, la sintaxis y las manías tipográficas particulares de la escritura de Grothendieck).

Se plantea, por tanto, como una vía de trabajo futuro más exploratoria y teórica, estudiar la capacidad de la red neuronal para reproducir, incluso parcialmente, *La longue marche à travers la théorie de Galois* o las cartas. Consistiría en incitar el proceso de regurgitación del modelo, evaluando si es posible recuperar fragmentos del conjunto de entrenamiento en ausencia del estímulo visual correspondiente. Se podría, por ejemplo, fomentar estos comportamientos introduciendo imágenes en blanco, ruido estructurado³⁶ o evitando el adaptador y usando unos pocos tókenes textuales de contexto a modo germinal.

Más allá de su utilidad evidente como métrica empírica para cuantificar el grado exacto de sobreajuste y contaminación adquirida durante el aprendizaje por currículum, este experimento presenta también interés técnico singular, pues permitiría comprobar si el modelo ha trascendido una labor estricta de OCR y medir la eficacia de las técnicas empleadas para evitar memorización.

³⁶En ocasiones ruido «bien escogido» puede llevar a comportamientos anómalos en algunos modelos de aprendizaje profundo [61]. Los adaptadores visuales, pues son transformers, suelen ser algo más robustos ante este tipo de perturbaciones que otros modelos.

A.

Implementación

El desarrollo práctico de este Trabajo Fin de Grado ha requerido la construcción de una infraestructura de software dedicada a solventar las necesidades técnicas tanto de la obtención del conjunto de datos como del ajuste fino. El susodicho código se divide en dos bloques principales: un conjunto de cuadernillos experimentales (aunque fundamentales) y un módulo estructurado de *Python*, llamado *croppen* [repositorio de GitHub](#), que centraliza la lógica de granularización, aumentado y soporte para el entrenamiento.

Este anexo desglosa, por tanto, las soluciones de software elaboradas. En primer lugar, se expone el pretratamiento aplicado a imágenes y transcripciones de *La longue marche à travers la théorie de Galois*, detallando la traducción de TeX a LaTeX y el reconocimiento y posterior recorte automático de imágenes empleando filtros morfológicos. Tras esto se presenta el despliegue y gestión del servidor centralizado de Label Studio³⁷, describiendo las herramientas de anotación y los algoritmos empleados para calcular durante el procesado de las anotaciones tanto la rotación como el orden de lectura de los párrafos etiquetados. Por último, se examina la lógica del aumentado de datos mediante extracción de subgrafos y se concluye con el desarrollo de herramientas personalizadas para la librería *Unslloth*, tales como *Collators*, *Callbacks* de limpieza de memoria gráfica y evaluadores duales de métricas.

³⁷Accesible en <https://grothendieck.online>.

A.1. Pretratamiento de *La longue marche*

En esta sección se discute el procesado que sufrieron las imágenes y transcripciones de *La longue marche à travers la théorie de Galois* antes de poder ser usadas, así como la traducción de TeX a LaTeX.

A.1.1. Extracción y recorte de las imágenes

El documento original de *La longue marche à travers la théorie de Galois* estaba en formato PDF, siendo un escaneo crudo conforme se extrajo del escáner. En consecuencia, las imágenes presentan todas un borde negro y un marco propio de una plantilla institucional:

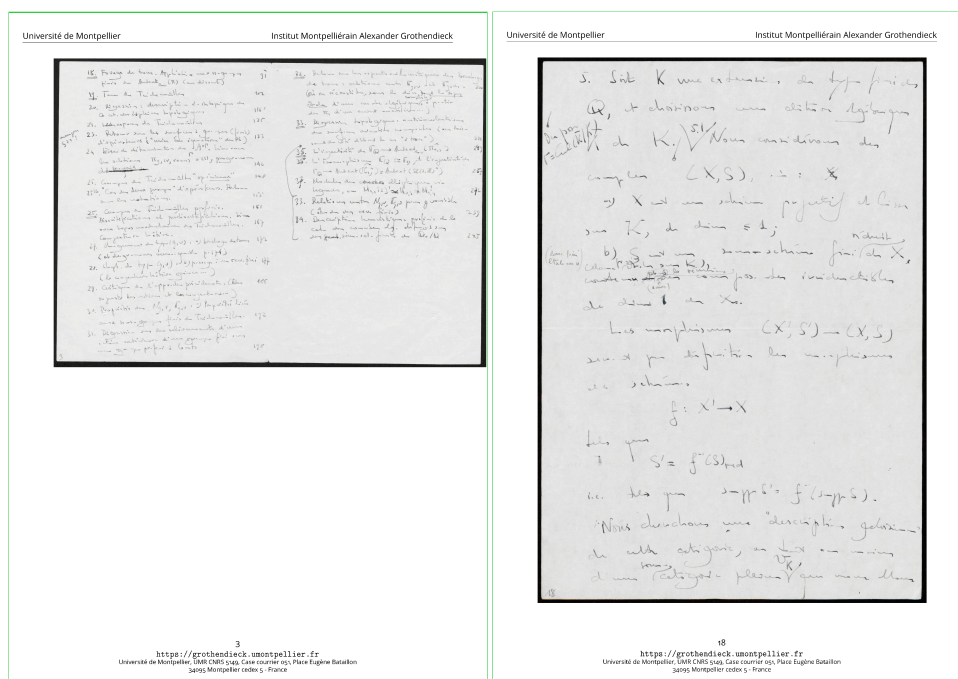


Figura 33. Dos páginas del documento original descargado de la web de la Universidad de Montpellier. Los bordes verdes exteriores marcan los bordes de página.

Puesto que el documento presenta escaneos de diferentes tamaños, formas y posiciones relativas dentro de la página, la extracción de estos escaneos no es directa. Analizando todas las páginas del documento, observamos el siguiente patrón común:

1. Un fondo blanco con texto de la Universidad de Montpellier que detalla información de la página. Para nuestro uso, es superfluo.

2. Dentro de este marco, de posición y proporciones variables, un marco negro rectangular. Este marco es el fondo del escáner, por lo que no es siempre de un negro uniforme sino solamente muy oscuro.
3. Dentro de este recuadro negro, el escaneo de la página, de proporciones variables, pero rectangular a grandes rasgos.

Nuestra estrategia, por tanto, va a hacer uso del patrón más regular dentro de las páginas y el más fácilmente explotable para extraerlas, el borde oscuro. Antes de exponer la sistemática de extracción, sin embargo, son necesarias dos definiciones relativas a teoría de tratamiento de imágenes para describir el proceso de filtrado que realizamos.

Definición (Dilatación morfológica en imágenes binarias): Sea $A \in \mathcal{M}_{n \times n}(\mathbb{Z}_2)$ una imagen binaria y $B \in \mathcal{M}_{m \times m}(\mathbb{Z}_2)$ otra, a la que llamamos *filtro*, tal que $m < n$ y $2 \nmid m$. Llamamos *dilatación de A con respecto a B*, denotado $A \oplus B$, a la imagen binaria $n \times n$ que resulta de aplicar el siguiente algoritmo:

Se recorren las coordenadas de A , digamos el índice i . Si A_i es nulo, se asigna a $(A \oplus B)_i$ el valor 0. En caso contrario, se superpone B a A centrando B para que coincida con A_i . Si algún valor no nulo de la versión de B superpuesta coincide con uno de A , se asigna un valor de 1 a $(A \oplus B)_i$; en otro caso, se le asigna el valor 0. ○

Definición (Erosión binaria en imágenes binarias): Sean A y B tal como en la definición anterior. Llamamos *erosión de A con respecto a B*, denotado $A \ominus B$, a la imagen binaria $n \times n$ que resulta de aplicar el siguiente algoritmo:

Se recorren las coordenadas de A , digamos el índice i . Se superpone B centrándola al valor A_i y, si todos los valores no nulos de B están superpuestos a valores no nulos de A , se asocia un uno a esa posición en $(A \ominus B)_i$. En caso contrario, se le asocia un 0. ○

Con esto en mente, el procedimiento es el siguiente:

1. Se fusionan los tres canales de color para generar una imagen en blanco y negro.
2. Se calcula una máscara binaria de la imagen que asigna a todo píxel con intensidad menor a un cierto umbral el valor 255, y 0 al resto.
3. Se aplica un kernel de tipo mediana 5×5 para suavizar detalles (por ejemplo el texto contiene zonas oscuras, la tinta, pero está rodeado de regiones más claras, el papel).

4. Se realiza la clausura morfológica de la imagen, que suaviza los detalles. La clausura morfológica es una técnica en procesamiento de imagen que, para imágenes binarias, se reduce a aplicar de forma encadenada una dilatación y una erosión.
5. Se emplea un algoritmo de búsqueda de contornos proporcionado por la librería cv2, que nos devuelve un conjunto de contornos (polígonos) basados en la imagen binaria. Llamamos C a este conjunto de contornos, de los que se descartan inmediatamente aquellos cuyo área encerrada sea menor a un 10% del total.
6. Tras esto, para cada contorno $c_i \in C$ se calcula el rectángulo de área mínima alineado con los ejes que lo contiene, al que llamamos r_i , y $R := \{r_i : c_i \in C\}$. Se calcula una puntuación heurística para escoger cada contorno como imagen, empleando la siguiente fórmula:

$$s_i = \frac{\text{Area}(c_i)}{\text{Area}(r_i) + \varepsilon_{\text{err}}} \cdot \frac{\text{Area}(c_i)}{A_{\text{img}}} \quad s_i \in [0, 1]$$

donde A_{img} es el área total de la imagen. Esta puntuación premia tanto la *rectangularidad* de la región delimitada por el contorno como el área total que ocupa.

7. Se escoge el rectángulo con mejor puntuación, tras lo que se reduce ligeramente (por un factor de 2 píxeles o un 0.2% de la imagen, lo que sea mayor, que es un mero heurístico) para recortar sin excedernos el borde negro.

A.1.2. Traducción de *La longue marche*

Definición (Fuzzy matching): Llamamos *fuzzy matching*, o *correspondencia difusa*, especialmente referido a cadenas de caracteres, a establecer una puntuación de similitud entre dos secuencias de caracteres. Dependiendo del algoritmo empleado, estas correspondencias pueden ser:

- Independientes o dependientes del orden.
- Considerar subcadenas o no considerarlas.
- Realizar la división a nivel de carácter o a niveles mayores.
- Ignorar o no ignorar caracteres especiales (como puntuación).

Siendo cada caso de uso el que define el tipo de algoritmo concreto a emplear.○

Una métrica empleada habitualmente en correspondencia difusa es la distancia de Levenshtein, que mide diferencias entre muestras y se ha definido ya en el documento principal, en la [Sección 3.6](#).

Como se detalla en el [Capítulo 3](#), fue necesario un proceso de sustitución de abreviaturas, uno de traducción de TeX a LaTeX y una segmentación de estos capítulos en páginas. Habiendo quedado el de sustitución de abreviaturas suficientemente cubierto en el susodicho capítulo, expandiremos en la traducción y segmentación.

Hacia el final del proceso se cambiaría el método por un enfoque sencillo (traducir primero los capítulos completos con un LLM y luego segmentar a mano este documento), mientras que el proceso original, que vamos a detallar, era más complejo:

1. Se toma una «buena» aproximación del fragmento de TeX que contiene la página cuya transcripción queremos obtener en LaTeX moderno. La aproximación deberá ser por exceso, pero todo lo ajustada posible.
2. Esta ventana de texto se pasa a GPT-5 mini (mediante una llamada a la API de OpenAI [\[62\]](#), empleándose también la de Gemini durante pruebas [\[63\]](#)) para que convierta la página a LaTeX moderno.
3. Se revisa manualmente el texto transcrito y la imagen de la página original, y se elimina cualquier parte (del principio o fin) que no se encuentre en la página.

Este último paso produce la página transcrita, con aportación manual, pero requiere un paso extra: la aproximación de la ventana del TeX en la que se encuentra el próximo archivo. Para hallarla, basta con tener una aproximación de la posición final de la página anterior y otra de la longitud de la página que queremos transcribir.

Puesto que es posible que el modelo modifique el texto traducido, o se eliminaran partes mediante revisión humana, era necesario inferir la ubicación del final en el TeX original a partir de la transcripción final, a LaTeX. Puesto que el segundo está modernizado y corregido, para esta tarea se emplean dos técnicas diferentes:

1. Se emplea *fuzzy matching*, puesto que las coincidencias no son exactas.
2. Se eliminan los bloques de matemáticas (contenidos entre dólares o entornos de ecuación), pues son aquellos que contienen mayores disparidades.

Esta deletación, sin embargo, no es permanente: la búsqueda difusa se realiza sobre este texto reducido, pero se conserva el original y una correspondencia entre los índices en cada uno de ellos, de forma que, tras la búsqueda, pueda hallarse la posición en el texto sin deletaciones. El *fuzzy matching* se realiza entre la parte final del texto producido por el modelo y el documento en TeX (comenzando en la posición de inicio de la página que se acaba de transcribir). Se realiza mediante ventana deslizante, aplicando búsqueda difusa de subcadenas (*partial_ratio* según su implementación en *fuzzywuzzy*), donde la subcadena es el final del LaTeX recién generado.

La aproximación de la longitud se hacía a partir de un valor medio en caracteres heurístico con ajustes manuales *ad hoc*: el cuadernillo de *Jupyter* empleado para este proceso imprimía tanto la página a generar como el texto a pasar al modelo, con una ventana interactiva que permitía aumentar o disminuir el tamaño de la ventana para comprender el texto necesario.

A.2. Label Studio, servidor de Oracle

El servidor se alojó en una instancia de Oracle bajo un plan gratuito, ejecutando Ubuntu canónico. A este servidor se accedía mediante SSH para realizar las configuraciones. Se instaló Python y Label Studio, abriendo un servidor, que se alojaría en el puerto 8080, estableciéndose el despliegue del servidor como un servicio del sistema³⁸. Tras configurar una red virtual en la nube de oracle, se *abrieron los puertos* necesarios para la conexión externa a este servidor.

Por las limitaciones de las instancias disponibles, fue necesario guardar las imágenes en un sistema de almacenamiento aparte, un «*bucket*», que también se alojaría en Oracle, también mediante el plan gratuito. Así, el servidor de Label Studio accede de forma dinámica a las imágenes, cargándolas en memoria durante la anotación, y libera esa memoria cuando no se están anotando, lo que ahorra memoria física a la instancia de cómputo.

Aunque opcional, para hacer las conexiones al servidor más sencillas para los anotadores, se adquiriría una licencia de un año para un dominio (grothendieck.online) de forma que no fuese necesario memorizar la dirección IP.

³⁸Así, el servidor se levanta y es accesible después de reinicios de la instancia de cómputo de forma automática

A.2.1. La interfaz y herramientas

La interfaz se divide en dos secciones principales: la imagen a la izquierda y la transcripción a la derecha, como puede verse en la [Figura 18](#). Los anotadores cuentan con 4 herramientas diferentes:

Caja-imagen (RectangleLabels) Permite dibujar rectángulos (que se pueden mover, redimensionar y rotar) alrededor de una línea manuscrita completa para definir su región visual.

Polígono (PolygonLabels) Permite trazar regiones irregulares punto por punto (añadiendo vértices) para separar limpiamente líneas de texto manuscrito que estén muy entrelazadas.

Fragmento de texto (Labels) Sirve para seleccionar y resaltar cadenas de caracteres o palabras dentro de la transcripción de texto para luego vincularlas a su respectiva imagen.

Corregir (TextArea) Es un cuadro de texto editable que permite modificar fragmentos de la transcripción original cuando se detectan erratas, omisiones o cambios de formato. Para editar un fragmento, este debe haberse seleccionado, creado un «Fragmento de texto», después seleccionar este objeto y añadir en este campo el texto con el que se quiere sustituir el que contiene.

Además, es posible asociar (mediante una relación de Label Studio, objetos que no son herramientas como tal)

A.3. Ingesta de datos de Label Studio

Como se ha expuesto en la introducción a este anexo, el proyecto de Python originalmente contenía únicamente la lógica necesaria para ingerir los datos recibidos de Label Studio y realizar el aumentado de datos, que es la parte discutida en esta sección. Este proyecto, elaborado en *Python* en torno al paradigma de programación orientada a objetos, trata de compartimentar las diferentes secciones que de él forman parte para que haya las mínimas interacciones posibles y reducir la complejidad. Cuenta con las siguientes partes:

Shared Partes comunes a la mayor parte del código. Entre ellas, la clase que almacena la información sobre los directorios o la estructura de los datos conforme se reciben de los distintos servidores³⁹.

External Interfaces Aquellas secciones del código que se comunican con servidores externos, como pueden ser Label Studio o el *bucket* de Oracle donde se guardaban las imágenes y las transcripciones.

Processing Primera parte del procesado, era la encargada de producir los objetos identificados con las unidades textuales y las unidades de imagen producidas por los etiquetadores, y producir tanto los recortes como sus transcripciones. También era esta la parte encargada de generar los recortes *aumentados*, compuestos de varios recortes generados por los usuarios.

Splitter Parte final del código, encargada de transformar los resultados obtenidos por el código anterior (guardados en imágenes, archivos de texto y un archivo que los relacionaba entre sí) y transformarlos a una tipología que otros módulos (*pytorch*, *huggingface_hub*, *unsloth*, etc.) pudieran manejar, realizando la división en conjuntos de entrenamiento, evaluación y test.

De entre ellas, nos centraremos en el procesado (con especial énfasis en la función *augment_data*), la contextualización, y la división entre entrenamiento y test, realizándose estas dos últimas tareas en **Splitter**.

El submódulo *Processing* convierte los *resultados*⁴⁰ recibidos de Label Studio en 4 objetos diferentes, dependiendo de su naturaleza. Estos, clases intrínsecas al proyecto de código, están divididos según su función y contenido:

ImageBox Almacena la información sobre las cajas-imagen (tanto poligonales como rectangulares) hechas por los etiquetadores.

TextFragment Almacena la información sobre las regiones de texto resaltadas por los etiquetadores, y hechas corresponder con las cajas-imagen.

Paragraph Almacena (parte de) la información sobre las adyacencias en imágenes, de forma que sea más sencillo dividir en bloques contextuales.

AnnotatedPage Contiene información sobre la anotación completa: guarda los párrafos en orden de lectura, así como referencias a las cajas-imagen y los fragmentos de texto, y emplea el orden de los párrafos para componer el texto.

³⁹Habría sido más seguro y estable limitar el alcance de un potencial cambio en las estructuras «crudas» recibidas de estos servidores externos solo a «External Interfaces». Esto, sin embargo, no resultó un problema durante la realización del presente trabajo, y no se hizo.

⁴⁰Mediante «resultado» aquí nos referimos a la unidad genérica mínima de anotación: regiones de texto (*TextFragment*) y cajas-imagen (*ImageBox*).

A.3.1. External interfaces

El submódulo de interfaces externas media entre la API de Label Studio (que se comunica con la instancia de Label Studio en el servidor), así como con el Bucket de Oracle para descargar las imágenes. A partir de un único objeto (`PathBundle`, de *Shared*), que determina el directorio maestro donde se colocarán los archivos necesarios, se descarga tanto una versión actualizada del *export* de Label Studio como las imágenes necesarias para realizar los recortes.

Así, se realiza una actualización de los datos local antes de la generación, abstrayendo la complejidad de este proceso.

A.3.2. ImageBox y TextFragment

En una correcta anotación existe una correspondencia biyectiva entre las instancias de `ImageBox` y las de `TextFragment`⁴¹, pues son, respectivamente, las regiones de la imagen delimitadas y sus correspondientes transcripciones.

Las instancias de `ImageBox` contienen, entre otra información, su recorte de imagen, información sobre el polígono que es su región de la imagen y el ángulo de rotación del polígono (se discute más adelante cómo llegar a este valor). Además, contienen información de la tarea a la que pertenecen y un identificador único que viene dado por Label Studio.

Por otro lado, cada instancia de `TextFragment` contiene una copia del texto al que referencia, así como un identificador único e información sobre la tarea de la que procede.

De esta forma, cada una de las muestras que proceden de una sola línea se corresponde con una pareja (`ImageBox`, `TextFragment`).

Además de estos dos objetos, los anotadores podían corregir el texto presente en un fragmento. Estas correcciones están presentes en el *export* como entidades del mismo nivel que las dos recién expuestas. Para incorporar estas correcciones se realiza un primer paso de simplificación del *export*, en el que se buscan estas correcciones en el *export* y se aplican a las líneas correspondientes. Esta versión del *export* donde la información de las correcciones está incorporada (y ya no tiene objetos «corrección») se guarda aparte como un *export simplificado*, que es con el que se trabaja de ahora en adelante.

⁴¹Existe un *test* en el código que se encarga de comprobar, entre otras cosas, esto mismo (`test_audit_annotations`).

A.3.2.1. Cálculo de la rotación de ImageBox

Para el cálculo del orden de lectura implementado (véase [Apartado A.3.4.2](#)) resultó necesario una medida de la rotación de cada recorte. En el caso de las cajas cuadradas, basta con tomar el valor de rotación proporcionado por Label Studio. Para cajas poligonales, sin embargo, no existe esta noción de rotación, puesto que el objeto no está alineado con los ejes por defecto.

Sin embargo la noción de línea, que es, en definitiva, lo que pretende encapsular el recorte, sí tiene definida una orientación. Esto nos motiva a asociarle a cada recorte poligonal un valor análogo al que presentan las cajas rectangulares. Para ello hacemos uso de que todas las cajas poligonales tienen forma generalmente oblonga, y son aproximables mediante un rectángulo. Con esto en mente, podemos calcular el rectángulo rotado mínimo⁴² y asignarle a este polígono la rotación de este rectángulo. En concreto, se calcula la rotación del rectángulo considerando su lado de mayor longitud (pues en las líneas manuscritas el lado más largo suele corresponderse con la dirección de escritura), y calculando la inclinación de esta arista. Así, un polígono obtiene un valor de rotación coherente con el texto que contiene sin perder ni ganar contenido, pues el recorte permanece intacto.

Todas las rotaciones se miden con respecto a la horizontal, de forma que el texto perfectamente escrito en esta dirección contaba con una rotación de 0°.

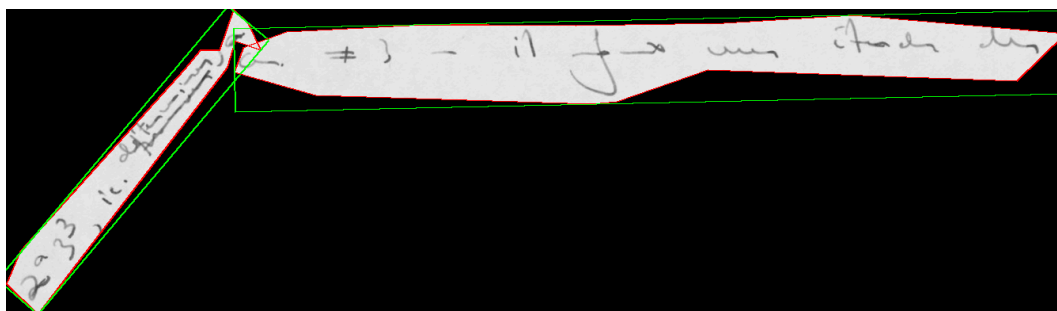


Figura 34. Imagen de dos cajas-imagen poligonales con rotaciones muy diferentes, sus respectivos polígonos (rojo) y sus rectángulos de área mínima (verde) sobre un fondo negro, en sus posiciones relativas en la imagen original.

⁴²El rectángulo rotado mínimo es el rectángulo rotado de área mínima que contiene una cierta figura geométrica. En nuestro caso, los polígonos.

A.3.2.2. Homogeneización del texto

A la hora de elaborar el escrito, Grothendieck fue bastante consistente en qué abreviaturas empleaba. No puede decirse lo mismo, sin embargo, de Jean Malgoire en los capítulos transcritos. La ambigüedad en algunos comandos de LaTeX y TeX, y su permisividad en el espaciado, hacen que sea posible transcribir un mismo pasaje de muy diversas formas. Por ejemplo:



Figura 35. Un fragmento de una de las páginas manuscritas

Admite simultáneamente toda una gama de transcripciones que producen un resultado equivalente, o igualmente aceptable, en LaTeX o TeX. Por ejemplo:

- `\mathcal{E}_{\mathcal{U}}, \Gamma \rightarrow \Gamma_{\mathcal{U}}`
- `\{\cal E\}_{\{\cal U\}}, \Gamma \rightarrow \Gamma_{\{\cal U\}}`

Con todas las variantes intermedias entre ellas, la sustitución de algunos comandos por sus equivalentes, la adición de más espacios o la adición de comandos de espaciado («`\ll`», «`\,`», «`\;`», etc.) para que el resultado sea más limpio... Todas estas decisiones son, hasta cierto punto, arbitrarias y (puesto que funcionan al nivel de token) son resultados completamente diferentes para un LLM⁴³.

El modelo requiere un criterio consistente por penalizarse transcripciones diferentes, incluso cuando producen el mismo resultado en LaTeX. Para conseguirlo, el texto de las líneas se preprocesa para homogeneizar las transcripciones de Malgoire y las posibles transcripciones de los diferentes anotadores. Entre otras, empleando expresiones regulares y otros formatos de sustitución hechos *ad hoc*:

1. Se eliminaron todos los espacios múltiples, sustituyéndolos por espaciado simple.
2. Se sustituyeron los subrayados y negritas del texto plano, pues Malgoire los empleaba de forma inconsistente para enfatizar partes que él consideraba relevantes.
3. Se normalizaron diversas notaciones (por ejemplo «`\{\cal \dots\}`» pasó a ser «`\mathcal{\dots}`», pues estaban presentes ambas variantes).

⁴³Si tokenizamos ambas secuencias con el tokenizador de GPT-4, por ejemplo, obtenemos dos secuencias: una de longitud 21 y otra de longitud 29, que tienen únicamente 5 elementos en común, y ninguno coincide en posición.

4. Se eliminaron algunos errores recurrentes o notación que Malgoire había modificado del manuscrito original, ya sea para modernizarla o corregirla.
5. Se eliminaron algunos caracteres poco habituales, como las dobles comillas estilizadas (un carácter especial para la de apertura y otro diferente para la de cierre), sustituyéndolos por la comilla inglesa estándar.

Estas sustituciones se realizaban durante la generación de la instancia de `AnnotatedPage` (que genera todas las demás) de forma que no fuese posible que contenido sin homogeneizar se «filtrase» a etapas posteriores del procesado.

Una alternativa aparente a esta sistemática habría sido, en lugar de estandarizar el texto de entrada, aplicar la estandarización tanto al resultado de la inferencia como a la transcripción y calcular sobre ello el CER. De esta manera, sin embargo, el modelo se entrena sobre un criterio variable (el resultado no homogeneizado), y, puesto que no emplea el CER como métrica de entrenamiento sino la entropía cruzada, el modelo aprende más lentamente.

A.3.3. AnnotatedPage

La clase `AnnotatedPage` sirve como contenedor de los recortes y las transcripciones, representados los dos tipos de objetos como instancias de las clases antes descritas. Además, contiene información global de la anotación (identificador de Label Studio, página concreta, si es o no una tarea, si se han modificado las imágenes de los recortes...), siendo la más importante de ellas el grafo de intersección de las imágenes.

En este grafo $G = (V, E)$, el conjunto de vértices viene dado por las cajas-imagen (sus identificadores), que serán adyacentes cuando sus polígonos respectivos solapen, aunque sea mínimamente. La decisión de emplear el criterio de que *las cajas deben superponerse* para considerarse que son adyacentes fue técnica. Así nos es posible separar cajas que, aunque cercanas, pertenecen a diferentes párrafos. Si, en su lugar, hubiéramos tomado un umbral (por ejemplo 1mm) a partir del cual asumimos adyacencia, habríamos perdido este control fino sobre las divisiones semánticas y estructurales del texto.

Originalmente no se realizaban asunciones sobre la estructura del grafo de cada anotación salvo que no era un pseudografo. Esto complicaba excesivamente la generación de los datos aumentados y, para resolverlo, fue necesaria una cuidadosa simplificación de las anotaciones. Esta modificación consistiría

en realizar aquellos cambios necesarios para transformar todas las componentes conexas de nuestra anotaciones en grafos camino: se crearía entonces la clase *Paragraph* para representar cada una de estas componentes conexas. La simplificación requirió aglutinar en una sola anotación líneas partidas en dos cajas, solapamientos entre líneas de un párrafo habitual y una nota al margen y otros casos particulares.

A.3.4. Paragraph

La clase *Paragraph* representa componentes conexas del grafo dado por las cajas-imagen y, en consecuencia, era poco más que un contenedor de estas (la información sobre adyacencias ya se guardaba en la estructura de grafo, correspondiente a *AnnotatedPage*). La implementación de mayor interés dentro de esta clase es la definición y cálculo del «orden de lectura». Con esto nos referimos al orden en el que deben concatenarse los textos de dos cajas-imagen adyacentes de forma coherente.

En un texto escrito perfectamente dentro de los márgenes y completamente en horizontal, esta tarea es tan sencilla como leer el texto verticalmente de arriba a abajo. Cuando a esto se añaden líneas auxiliares añadidas *a posteriori*, aclaraciones, notas al margen, texto en diagonal, etc., la tarea de decidir qué parte del texto va primero se vuelve algo más complicada, especialmente programáticamente.

A.3.4.1. Alternativas para orden de lectura

Si tratamos de buscar una ordenación consistente en el sentido izquierda-derecha arriba-abajo, dos posibles opciones son las siguientes:

1. Orden lexicográfico en las coordenadas mínimas o en el centroide:⁴⁴ Esta ordenación se reduce, puesto que las anotaciones son manuales y cajas diferentes nunca coinciden en ninguna de las dos coordenadas, a una ordenación en la primera coordenada.
2. Combinación de las coordenadas: Otra alternativa es escoger alguna forma de pesar las coordenadas de las cajas (centroide, cada uno de los cuatro vértices), y emplear el resultado para ordenarlas.

⁴⁴Asumiendo, como era el caso para los datos que recibíamos, que una menor coordenada x se correspondía con estar más a la izquierda en la página, y una menor coordenada y se correspondía con estar más arriba.

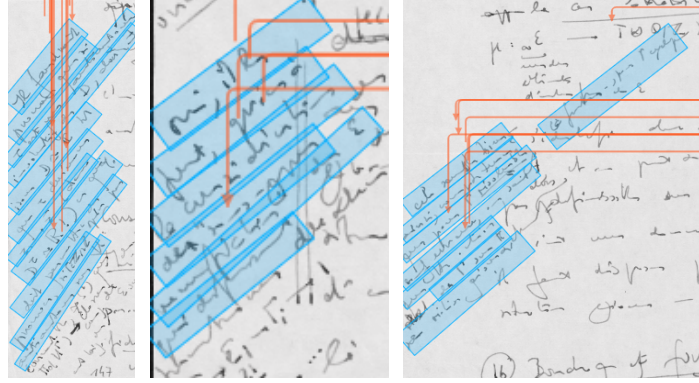


Figura 36. Diferentes grupos de líneas, ya sea entre páginas o dentro de una misma página, presentan muy diferentes inclinaciones. Esto descarta el orden lexicográfico en cualquiera de sus alternativas, y dificulta (o imposibilita también) escoger una expresión para la combinación de coordenadas que funcione en todos los casos.

El primero de los dos órdenes falla en general para algunos párrafos concretos, mientras que es extremadamente difícil hallar una expresión que funcione para todas las páginas, párrafos e inclinaciones de nuestras muestras.

A.3.4.2. Orden de lectura escogido

La solución a este problema es considerar el orden de lectura de la misma forma que leemos los humanos: un párrafo se lee siempre de principio a fin. En consecuencia, basta con dividir en párrafos la anotación (componentes conexas del grafo de intersecciones) y considerar el orden de lectura en cada caso.

Dentro de un mismo párrafo no existe tanta variabilidad en cuanto a las inclinaciones, otra que la natural al escribir a mano y anotar a mano. Con esto en mente, aplicamos la siguiente sistemática dentro de un párrafo cualquiera P :

1. Se obtiene el ángulo de cada caja $c_i \in P$, a saber θ_i , que hemos obtenido en las cajas poligonales mediante el sistema antes expuesto. Esta rotación vendrá dada por un vector unitario de la forma $\text{Rot}(c) = (\cos(\theta_i), \sin(\theta_i))$.
2. Se calcula el vector de rotación promedio del párrafo ponderando estos vectores por el área de cada caja,

$$(a_P, b_P) := \frac{1}{\text{Area}(P_k)} \sum_{c \in P_k} \text{Rot}(c) \cdot \text{Area}(c).$$

3. Se calcula el ángulo de rotación promedio del párrafo como

$$\bar{\theta} = \arctan\left(\frac{b_P}{a_P}\right)$$

4. Se calcula un pseudocentroide⁴⁵ del párrafo ponderando los centroides de cada caja de la misma forma que con los vectores angulares.
5. Se *enderezan* las cajas haciendo una rotación de centro este pseudocentroide y ángulo $-\bar{\theta}$. Estos valores se calculan y se guardan, pero no sustituyen los valores de rotación de las cajas en sus instancias, pues la información sobre la rotación original nos hará falta para la generación del conjunto de datos.
6. Se ordenan entonces las cajas-imagen por su punto más alto en este espacio transformado.

Este proceso produce buenos resultados en general, pero admite un paso extra que asegura un funcionamiento mucho más robusto en el caso en el que el párrafo tenga grafo asociado isomorfo a un grafo camino. En este caso, tras ordenarse los párrafos se toman los nodos terminales y se escoge cuál es el inicial (aquel que está más alto), siendo el restante el nodo terminal. Se asigna entonces como orden de lectura el dado por transitar el grafo del nodo inicial al terminal.

A.3.4.3. Entrelazamiento de párrafos

Pese a que la herramienta de recorte poligonal presente en Label Studio permite un control muy fino sobre el solapamiento de líneas, algunos párrafos que se encuentran muy entrelazados no permiten dividirse. Este es el caso, por ejemplo, cuando dos palabras de dos párrafos distintos solapan entre sí, estando una escrita completamente por encima de la otra.

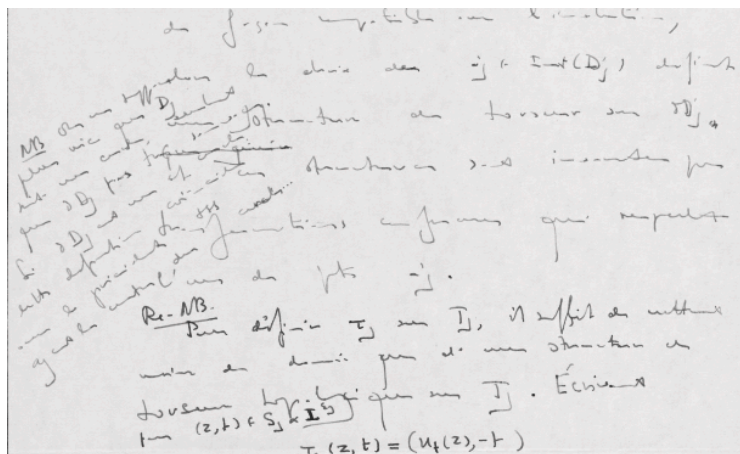


Figura 37. Dos párrafos que solapan completamente. Puesto que hay palabras escritas sobre otras, no es posible recuadrar las líneas dividiendo limpiamente en dos párrafos en una misma anotación.

⁴⁵No es un centroide real, puesto que las cajas pueden solapar.

En estas situaciones, la solución es relativamente sencilla: tanto las transcripciones como los recortes marcados por el anotador se trasladan (manualmente, juzgando en cada caso) de una misma anotación a dos diferentes (de la misma tarea) en Label Studio. Puesto que el cálculo de los párrafos se elabora dentro de cada instancia de `AnnotatedPage`, que procesa cada anotación individualmente (no cada tarea, que contiene varias anotaciones), esto separa completamente ambos párrafos pese a que ocupen unas mismas coordenadas en la página.

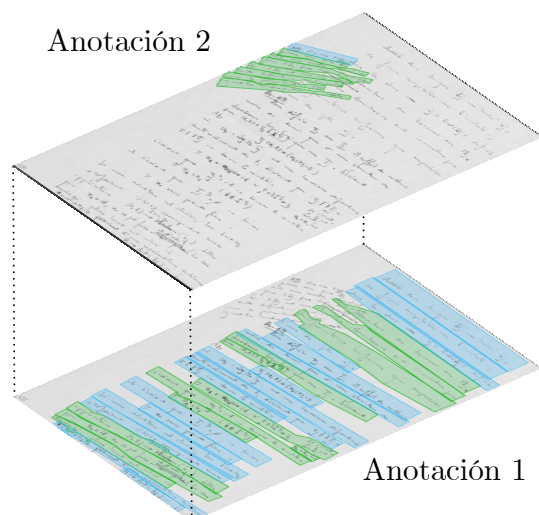


Figura 38. Diagrama de la división en dos tareas para separación de párrafos entrelazados.

Este proceso podría automatizarse, haciendo innecesaria la intervención de dividir manualmente en dos tareas. Una sistemática sería emplear un algoritmo de *clustering* (por ejemplo empleando la mínima coordenada horizontal y la rotación), agrupando en nuevos subpárrafos cada párrafo. Se juzgaría la calidad de esta división empleando alguna puntuación (como la de silueta) y se dividiría en tantos párrafos como se indique si esta puntuación supera un umbral. Así, sería posible separar estas notas al margen del texto «normal» de forma automática, ajustando correctamente los parámetros del clustering.

Sin embargo, puesto que dividir en varias tareas no es complicado ni trabajoso en Label Studio, siendo además un problema no muy extendido, este proceso nunca llegaría a implementarse.

A.3.5. Función *augment_data*

La función *augment_data* es la función principal del proyecto: carga los recortes, aglutina varias líneas para conseguir datos sintéticos, compone sus transcripciones y guarda los datos.

La generación de las muestras aumentadas se realiza siempre con líneas que pertenecen a un mismo párrafo. En primer lugar se toman los recortes de cada línea considerada y se colocan, montando un *collage* de recortes de líneas, en sus posiciones originales en la página. Tras esto, se considera el texto resultante de concatenar sus transcripciones en el orden de lectura, y se recorta la página a las dimensiones del collage interior para no desperdiciar memoria en disco.

Puesto que las formas de los recortes no tienen por qué llenar perfectamente una imagen (que siempre son cuadradas), el espacio vacío se rellena con el color promedio de la página de la que provienen las líneas. Se emplea este color en lugar de blanco puro para evitar un posible sesgo del modelo al color blanco.

A.3.5.1. Generación de subgrafos

La creación de datos sintéticos necesariamente tenía que venir informada por la estructura de nuestro modelo y la tarea en la que queremos que mejore. Puesto que queremos que el modelo aprenda a transcribir textos coherentes, no tendría sentido aglutinar líneas que no guardan relación semántica y emplear como ejemplo de entrenamiento la concatenación de sus transcripciones. Nuestra intención con el aumento de los datos, por tanto, es conservar la semántica todo lo posible.

El aumentado de datos sufrió durante la evolución del proyecto un cambio fundamental. En un principio, como se ha discutido anteriormente, se consideraban grafo generales, para terminar considerándose grafos unión disjunta de grafos camino. Discutimos ambas generaciones a continuación.

Una primera implementación empleaba un algoritmo general de generación de subgrafos conexos sensible al orden de los vértices. Así, para un mismo grafo $A - B - C - D$, repeticiones como $A - B - C$ y $C - B - A$ se generaban por separado. Esto, naturalmente, no es admisible pues las muestras asociadas a ambos subgrafos son idénticas en transcripción y recorte.

Para evitar duplicados, se convertía entonces la secuencia de identificadores de los nodos en un `set`⁴⁶ de Python, y se guardaban las secuencias generadas. Antes de generar los datos correspondientes a una nueva secuencia, se comprobaba si ya se había generado una permutación equivalente a ella, evitando generar duplicados.

Tras la transición hacia grafos camino, finalmente, fue posible explotar la estructura de estos grafos para simplificar la generación. Recorriendo cada subgrafo (correspondiente a un párrafo, luego conexo), bastaba con ordenar los vértices en el orden de lectura y generar subsecuencias ininterrumpidas de esta lista. Por supuesto para muestras de ℓ líneas, se tomarían subsecuencias de longitud ℓ , siendo necesariamente ℓ menor a la longitud en líneas del grafo.

procedure subgraphs(graph):

```

1  subgraphs = list
2  for k from 1 to graph.order do:
3      subgraphs_order_k ← list
4      for i from 0 to graph.order do:
5          subgraph ← vertex_list[from i to i + k]
6          insert subgraph in subgraphs_order_k
7      subgraphs ← subgraphs  $\sqcup$  subgraphs_order_k
8  return subgraphs

```

A.3.5.2. Generación parcial

La generación de los párrafos en primera instancia pasó por generar solamente una pequeña muestra de los mismos: para páginas con muchas líneas, por ejemplo 30, la generación de todos los subgrafos (conexos) de las mismas resulta prohibitivamente difícil computacionalmente para el caso general.

Esta generación se realizaba empleando una técnica conocida como *generación con reservorio*:

1. Se llena una lista (el reservorio) de subgrafos hasta llegar a una cierta cantidad predefinida (por ejemplo k). El reservorio está lleno en este punto, y no crecerá más en longitud.

⁴⁶Realmente un `frozenset`, puesto que al ser los conjuntos de Python mutables no es posible aplicarles una función hash.

2. Se generan más subgrafos y, con probabilidad p , se sustituye uno de los presentes en el reservorio (elegido también aleatoriamente) por este nuevo subgrafo.

La ventaja de emplear esta técnica frente a generar todos los grafos y, a partir de esta secuencia, elegir aleatoriamente, es que no tenemos la necesidad de tener cargados en memoria simultáneamente todos los identificadores de los subgrafos.

Con la transición hacia subgrafos tipo camino se eliminaría esta parte del código por no ser necesaria al ser la generación de subgrafos en este caso es mucho más sencilla. Para un párrafo de orden K solo existen $K - \ell + 1$ subgrafos posibles de orden ℓ , que se corresponden de forma natural con los números del 0 al $K - \ell$, luego nos basta con escoger tantos números como grafos se quieran generar empleando, por ejemplo, `random.randint`.

A.3.6. Adición de contexto

Un LLM es un sistema probabilístico que trabaja mejor cuanto más información tenga de una oración: es mucho más probable que la siguiente palabra sea «*canonique*» si el texto precedente es «*il existe un isomorphisme*». En consecuencia, resulta de interés intentar entrenar a nuestros modelos no solamente sin esta información, sino también añadiéndosela (de forma que sea versátil en cuanto a la presencia o no de contexto).

Nuestro texto tenía el problema, sin embargo, de estar «agujereado» por partes textuales que no se consideraban en las transcripciones presentes en el conjunto de datos final, así como no contener las correcciones hechas por los anotadores. Necesitábamos, por tanto, para contextualizar cada muestra, una transcripción del contenido de la página que se había anotado.

Este proceso pasa primero por ordenar los párrafos según sus respectivos pseudocentroides antes descritos, tras lo que se genera la transcripción completa leyendo las líneas de cada párrafo en orden, leyendo los párrafos en orden. Durante este proceso se asigna, además, a cada línea manuscrita (instancia de `TextFragment`) un *índice de inicio*, que se corresponde con su posición dentro de esta transcripción de la página. Puesto que el índice se mide en caracteres, es posible añadir a cada línea tantos caracteres de contexto como índice mínimo tenga. Para una muestra que aglutine varias líneas, lo mismo puede decirse considerando el menor de los índices de inicio de las líneas que la conforman.

Partiendo de un umbral de contexto mínimo aceptable, si el contexto posible dentro de su página tenía la suficiente longitud (ya sea en palabras o en caracteres) el proceso terminaba aquí. Si no, se buscaba la transcripción de la página anterior⁴⁷ y el contenido faltante hasta completar los caracteres necesarios de contexto se tomaba del final de esta.

Este contexto se añade como un `str` a todas las muestras, de forma que sea posible decidir durante el proceso de entrenamiento (incluso de forma aleatoria en cada muestra) si se añade o no, y cuánto, de este contexto.

A.3.7. División entre train y test

En lugar de hacerse la separación entre el conjunto de entrenamiento y el de validación antes de cada entrenamiento (lo que podría llevar a diferencias entre estos conjuntos, y por tanto diferencias en los criterios para elegir uno sobre otro), este se realizaba durante la fase de aumentado de datos.

Tras el aumentado de datos se escoge una proporción del total de líneas que terminarían en train, valor al que llamamos p . Para evitar contaminación cruzada, la asignación de muestras entre conjuntos no se realizó a nivel de línea, sino a nivel de página: todas las líneas de una página van o bien a un conjunto, o bien al otro, garantizándose la ausencia de solapamiento entre ambos.

La asignación se realizó mediante un algoritmo *greedy* que, teniendo dos conjuntos A (equivalente a *train*) y B (equivalente a *test*), asigna de manera secuencial páginas a cada uno de los dos conjuntos. Denotamos:

- A_t y B_t los conjuntos de líneas individuales antes de asignarse la página t -ésima.
- T_A y T_B las páginas correspondientes a cada conjunto.
- Y_t el conjunto de líneas de la página t .
- $P(A_t, B_t) = \frac{|A_t|}{|A_t \cup B_t|}$

Entonces el algoritmo procede de la siguiente manera:

procedure greedy_pagewise_split(p):

```

1 |  $A, B$  : set
2 | for  $t$  from 1 to  $T$  do:
```

⁴⁷Si la hubiera; si no, no era posible añadir más contexto. En caso de existir varias anotaciones de la página anterior (véase [Apartado A.3.4.3](#) sobre el entrelazamiento de párrafos), se tomaba la de mayor longitud.

```

3   |  $A^* \leftarrow A \sqcup Y_t$ 
4   |  $B^* \leftarrow B \sqcup Y_t$ 
5   | if  $|P(A^*, B) - p| \leq |P(A, B^*) - p|$  then:
6   |   |  $A \leftarrow A^*$ 
7   |   |  $T_A \leftarrow T_A \sqcup \{t\}$ 
8   | else:
9   |   |  $B \leftarrow B^*$ 
10  |   |  $T_B \leftarrow T_B \sqcup \{t\}$ 
11  | return  $T_A, T_B$ 

```

El conjunto de entrenamiento será el dado por filtrar el conjunto total por las páginas en T_A , y el de test el resultante al hacerlo por T_B . Nótese que la proporción p entre elementos de entrenamiento y validación solamente puede garantizarse (aproximadamente) en líneas individuales: puesto que no empleamos muestras de mayor orden para el cálculo de P , tras añadirlas esta proporción cambia. Suponemos, sin embargo, que la variación no es significativa, siendo nuestro interés principal que las muestras de orden 1 (con las que comienza a entrenarse nuestro modelo) estén distribuidas según p .

Como conjunto de desarrollo se tomaron aleatoriamente 200 de las muestras presentes en el conjunto de test, de forma que, pues train y test son mutuamente estancos, podemos asegurar que no hay contaminación entre el conjunto de desarrollo y el de entrenamiento.

Entre todos los entrenamientos se comparte este mismo conjunto, pues no se realiza la división durante el entrenamiento sino durante la generación del conjunto de datos, que se sube a Huggingface ya segregado.

A.3.8. Estructura final

Los campos del conjunto de datos final eran los siguientes:

- **image:** Imagen del recorte, en formato RGB.
- **text:** Transcripción de la imagen.
- **context:** Texto anterior a la transcripción de la imagen. La longitud máxima de este contexto era modulable.
- **ann_id:** ID único de la anotación concreta.
- **order:** número de líneas que incluía la muestra.

- `is_letter`: True en caso de ser una muestra procedente de una carta, False en caso contrario.
- `avg_color`: Color promedio de la página, para rellenar el fondo de la imagen en caso de ser necesario.
- `avg_rotation`: Rotación promedio de las líneas que incluye la muestra. Este valor podría emplearse durante el entrenamiento para escoger dinámicamente la rotación aplicada, por ejemplo para conseguir una muestra aproximadamente horizontal.

Esta es la estructura que se sube a Huggingface, sin tener formato de conversación. Durante la instanciación como muestras para ser suministradas al modelo únicamente se emplean los tres primeros campos (`image`, `text`, `context`), siendo el resto para extraer métricas de rendimiento o para estructurar el aprendizaje por currículum.

A.4. Data collators, Unsloth

La librería empleada para el *fine-tuning* usando QLoRA es *Unsloth*. Nos permite cargar modelos cuantizados directamente de *Huggingface*, el repositorio estándar en la industria para modelos *open-weight*. Puesto que la elección de repositorio se realizó desde el principio, las decisiones de formato para el conjunto de datos se tomaron ya con esto en mente.

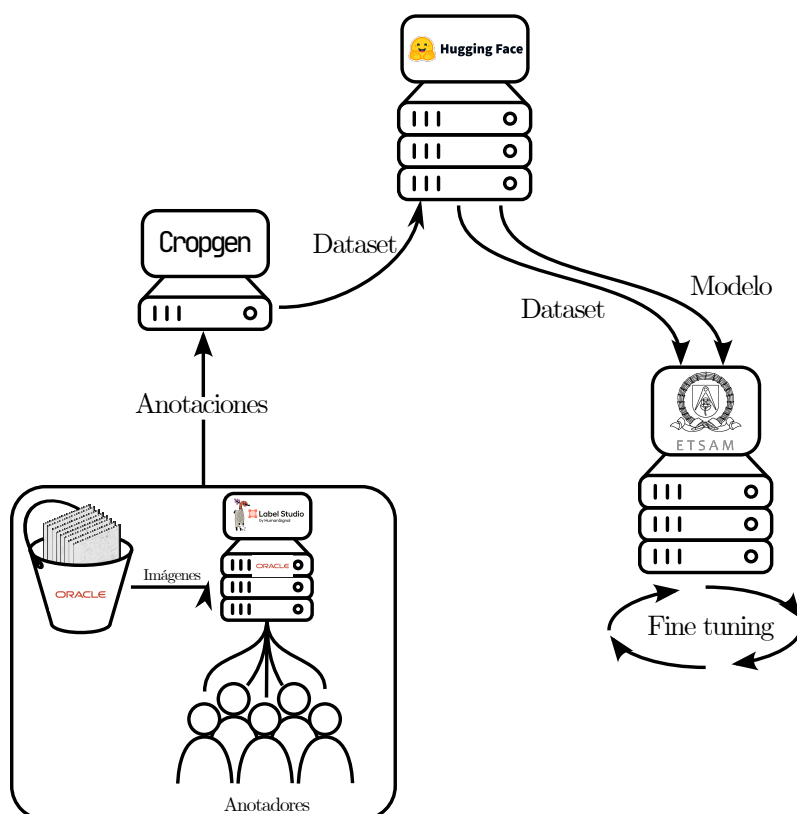


Figura 39. Diagrama de la generación del dataset, su subida a *Huggingface* y su posterior carga, de forma paralela al modelo, en el equipo donde se ejecuta el *fine-tuning*.

Por la naturaleza de los LLM, estos no se cargan solos desde Huggingface, sino que vienen acompañados de su tokenizador. Puesto que cada línea de modelos sigue una sintaxis y estructuras ligeramente diferentes, Huggingface se encarga de envolver ambos objetos en clases estandarizadas y homogéneas, que permiten interactuar de la misma manera con diferentes LLM, lo que hace especialmente sencillo cambiar de un modelo a otro.

En Unsloth, el entrenamiento se configura empleando un entrenador (*trainer*), que recibe como argumentos los valores de los parámetros como la penalización a los pesos, el número de épocas, cada cuántas épocas realizar un guardado de los parámetros, cuál es el conjunto de entrenamiento y de evaluación, la métrica que emplear en evaluación, etc. Siendo bastante modular, es también posible personalizar muchas de sus partes, por ejemplo, introduciendo funciones que deben ejecutarse al producirse algunos eventos en el entrenamiento o evaluación.

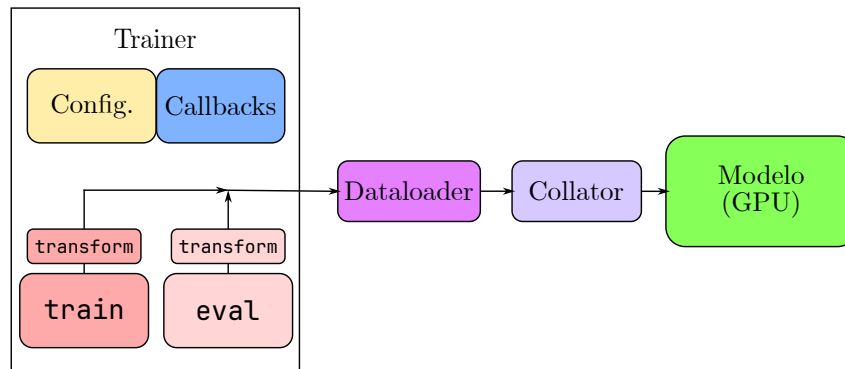


Figura 40. Diagrama de funcionamiento simplificado de las partes de *Unsloth* que intervienen en nuestro entrenamiento y hemos personalizado.

A.4.1. Transformaciones del conjunto de datos

Un LLM no tiene forma de interpretar un conjunto de datos como el descrito en [Subsección A.3.8](#). Estos esperan una estructura de conversación, que, a partir de nuestros datos, podemos conseguir de forma relativamente sencilla. Guardar directamente nuestro conjunto de datos como conversación es extremadamente ineficiente en espacio (el *prompt* se repite entre todas), además de que no queremos perder aquellos campos descritos en la sección anterior que, aunque no están presentes durante el entrenamiento, pueden ser útiles para obtener estadísticas.

Puesto que operamos con una instancia de `datasets.Dataset`, podemos emplear una de las características de esta clase, las transformaciones *on-the-fly*. Estas son transformaciones que se aplican a cada muestra individual cada vez que se quiere acceder a ella. Por ejemplo, de igual forma que en una lista accedemos a un elemento indiándolo:

```
>>> A = [1,2,3]
>>> A[1]
```

Cuando en *Python* se ejecuta una llamada con esta sintaxis sobre un objeto object de una clase Class (`object[i]`), el intérprete ejecuta internamente `Class.__getitem__(object,i)`. Así, una transformación *on-the-fly* de una instancia de `datasets.Dataset` se obtiene ejecutando

```
>>> X.set_transform(f)
```

y modifica el comportamiento de `datasets.Dataset.__getitem__(X, i)` para la instancia concreta *X*. En lugar de devolver la fila *i*-ésima del conjunto de datos *X*, a saber X_i , cada vez que se indiza este valor se devuelve $f(X_i)$. La ventaja

de esto es que el valor *interno* no se modifica ni sustituye (aunque permanece inaccesible, devolviéndose su valor transformado por f , hasta que no se cambie o elimine la transformación asignada).

Así, las funciones de transformación que hemos empleado tienen entonces dos funciones:

1. Dar a las muestras su estructura de conversación.
2. Realizar un segundo aumentado de datos: se añaden pequeñas rotaciones, distorsiones, movimientos, zooms, etc. a las imágenes; se decide si se añade contexto a la imagen según una cierta probabilidad p_{ctxt} , se decide *cuánto* contexto añadir (en caracteres), y se añade a la conversación en caso de haber. Los parámetros para estas transformaciones (rotaciones máximas y mínimas, probabilidad de añadir contexto, etc.) no tienen por qué coincidir para los conjuntos de evaluación y de entrenamiento, pues son dos instancias diferentes de `Dataset`, y por tanto sus transformaciones asignadas no tienen por qué coincidir.

A.4.2. Collators, callbacks y evaluador

Un *collator* es uno de los sistemas que se encarga de mediar entre el conjunto de datos y la GPU. Durante el entrenamiento, el collator recibe de otra estructura (el *data loader*) un subconjunto del conjunto de entrenamiento, el batch, les da el formato adecuado (o comprueba que lo tienen) y las envía a la tarjeta gráfica de forma que puedan procesarse correctamente. Asimismo, en Unsloth es el collator el encargado de recibir la información de respuesta de la GPU y prepararla para ser procesada, como la predicción del modelo durante la evaluación o la distribución de probabilidad que este genera durante el entrenamiento; en este último caso para calcular la entropía cruzada.

Puesto que media los cálculos durante el entrenamiento, es precisamente en el collator donde se manobra para implementar la solución propuesta en [Sección 3.5](#) para la contaminación por el contexto. Basta con activar el parámetro `train_on_responses_only`. Así, la pérdida se calcula únicamente sobre los tokens que se encuentran en la respuesta del modelo.

Por otro lado, un *callback* es un objeto de una clase concreta que, en caso de implementar algunos métodos concretos, se sincroniza con el proceso de entrenamiento para llevar a cabo alguna tarea al producirse algunos eventos du-

rante el entrenamiento. Por ejemplo, un callback puede implementar métodos `on_step_begin(...)` y `on_epoch_end(...)`, que se ejecutan, respectivamente, al inicio de una iteración y al final de una época.

A.4.2.1. Implementación del CL

Para la implementación del aprendizaje por currículum eran necesarias dos partes:

1. Un intermediario entre *trainer* y *dataset* que, conforme avanzase el entrenamiento y siguiendo un programa específico, ampliase progresivamente las muestras de entrenamiento a las que este segundo tiene acceso.
2. Una estructura que mediase con esta última, que tuviera acceso al progreso del entrenamiento, y le señalizara en qué punto del entrenamiento (y por tanto del programa) se encuentra el *trainer*.

Esto se ha conseguido con la implementación de dos clases, `CurriculumDatasetInterface` (CDI) y `CurriculumCallback` (CC). La primera de ellas, que hereda de `datasets.Dataset` y por tanto comparte todas sus funcionalidad, requiere para instanciarse 4 partes clave:

1. El «programa»: en nuestro caso, una lista de listas de enteros, cada una correspondiente a una época, que señalan las longitudes de qué muestras van a emplearse durante el entrenamiento en una época concreta.
2. El conjunto de entrenamiento completo.
3. La función de restricción de nuestro dataset, que recibe como argumentos el conjunto de entrenamiento completo y el programa de esa época (en nuestro caso, una lista de enteros).
4. La función de transformación del conjunto de entrenamiento, que debe reasignarse a los subconjuntos de entrenamiento empleados en cada época.

Por otro lado, `CurriculumCallback` simplemente implementa un método `on_epoch_end` que manda una señal a la interfaz de currículum indicando el indicándole que avance el programa. Tras esto, la interfaz actualiza el conjunto de datos «activo» siguiendo el programa: se llama a la función de restricción del dataset con los argumentos que corresponden a esa época siguiendo el programa.

Esta implementación tiene dos problemáticas principales:

1. El optimizador de todo el entrenamiento, pues es el mismo, arrastra el momento de los gradientes de una etapa del currículum al siguiente.
2. Puesto que *Unsloth* únicamente calcula el tamaño del conjunto de entrenamiento una vez, el cálculo de las épocas es completamente erróneo. Además, puesto que el dataloader recibe la información del tamaño del conjunto de datos y escoge las tandas de datos para la GPU en consecuencia, el número de muestras de entre los que se muestrean nunca es el conjunto de datos completo salvo cuando el conjunto de datos es tan grande como en la primera etapa.

A.4.2.2. CleanupCallback

Puesto que se empleó un servidor con *Windows Server* mientras que los paquetes como *Unsloth* y *Triton* están específicamente diseñados para *Linux*, existía un problema de compatibilidad parcial que se establecía únicamente con la GPU. El problema, que se resumía en una fragmentación irreparable de la GPU, resultaba en un llenado progresivo de la GPU conforme pasaban las iteraciones.

Para resolverlo, se implementó una primera clase *CleanupCallback*, que evaluaba el uso de memoria de la GPU y la vaciaba completamente al acabar una iteración si esta sobrepasaba un umbral de ocupación. Esta, bastante rígida, se sustituiría finalmente por otra, *ProactiveCleanupCallback*, que estima la cantidad de memoria a reservar durante la siguiente iteración y, si sobrepasa un umbral, vacía completamente la memoria.

Esta segunda implementación es más robusta y flexible puesto que nos permite establecer un umbral más alto sin perder seguridad. La estimación emplea una media móvil ponderada exponencialmente de la memoria alocada durante las últimas iteraciones en la misma época. Además, pues se tiene en cuenta el posible cambio en el conjunto de datos entre épocas siguiendo el currículum, vacía la memoria entre épocas y *olvida* la información sobre estimaciones anteriores.

A.4.2.3. Evaluador

Puesto que nos resulta de interés conseguir métricas tanto *con* contexto como *sin* contexto, necesitamos una manera de obtener ambas métricas separadas. Una vez que se envía una muestra (ya con formato de conversación) a la

GPU, el programa continúa y no es hasta que la GPU devuelve su predicción, únicamente de la respuesta, que se consigue una respuesta. Este proceso separa completamente la fase de envío de una muestra y la recepción de la respuesta, de forma que no es directo saber cuándo una respuesta procede de una muestra contextualizada de cuando no.

Así, para poder separarlas de forma más limpia, se separó el conjunto de evaluación en dos copias: una de ellas configurada con una transformación *on-the-fly* que nunca emplea contexto (para las métricas sin contexto), y otra que emplea una transformación *on-the-fly* que siempre incorpora contexto.

Nuestra intención, sin embargo, no es tener ambas métricas (CER con contexto, CER sin contexto) y emplear únicamente una de ellas para escoger el mejor modelo, sino que queríamos emplear una mezcla, la media, para discernir el mejor modelo. Para esto, sin embargo, nos es necesario definir una nueva métrica a partir de ambas, que tampoco es directo empleando *Unsloth*.

Para implementarlo se usó una clase independiente, `DualMetricsCalculator`, que se aprovecha de que los conjuntos de evaluación se revisan siempre en el mismo orden que se ha introducido. Así, en lugar de introducirse al *trainer* una función para calcular el CER como función de evaluación, se introduce una instancia de esta clase que:

1. Cuando se intenta ejecutar (como si fuera una función) sobre el primer conjunto de evaluación (siempre el que no contiene contexto), calcula el CER, lo guarda y lo devuelve. Este resultado devuelto como «cer» se formatea automáticamente como `no_context_cer`, puesto que se concatena el nombre de la métrica al del conjunto sobre el que se ha evaluado.
2. Tras esto, cuando se vuelve a llamar sobre el conjunto con contexto, realiza el mismo proceso y devuelve, además, la media entre el CER que acaba de calcular y el que tenía almacenado.

Así, contamos con todas las métricas necesarias.

A.4.3. Guardado de los parámetros

Para poder mantener un cierto orden durante los entrenamientos sería necesario llevar un registro de los parámetros empleados durante el entrenamiento. Estos parámetros proceden de cuatro fuentes:

- Configuración del *trainer* (*learning_rate*, *weight_decay*, número de épocas, etc.).

- Adaptadores LoRA (rango, qué capas tienen adaptadores, etc.).
- El programa empleado en el currículum.
- Parámetros de las transformaciones (*prompt*, rotación máxima, rotación mínima, zoom máximo, etc.).

Donde la información sobre la construcción del conjunto de datos ya se encuentra en Huggingface, puesto que se almacena directamente en esta plataforma en lugar de realizarse la división justo antes del entrenamiento. Se realiza de esta forma para evitar cambios en la división entre entrenamientos y asegurar homogeneidad.

Para organizar cada una de estas fuentes se implementarían sendas *dataclasses*, que se instancian antes del entrenamiento con los parámetros concretos del entrenamiento a realizar, y se suben conjuntamente al mismo repositorio de datos que el modelo. De esta forma es posible acceder de forma sencilla a todos ellos cuando se carga el modelo.

B.

Notas sobre LLMs

Este anexo expone los fundamentos teóricos del entrenamiento y funcionamiento que sustentan las arquitecturas presentes en LLM, aunque de forma algo superficial al ya haberse expuesto el transformer en el [Capítulo 2](#). Se presenta el tratamiento del lenguaje natural que efectúan, la representación de información sobre espacios vectoriales y los diferentes paradigmas empleados durante su entrenamiento. Se pretende justificar algunas decisiones tanto estructurales como metodológicas adoptadas en los capítulos principales de la memoria, extendiendo el análisis de la tokenización del texto hasta los algoritmos de aprendizaje por refuerzo y las estrategias de mitigación de alucinaciones.

Así, y como pieza fundamental, llamamos *entrenamiento* de un LLM, de igual forma que hablamos de entrenamiento en un modelo de *machine learning* habitual, al proceso de afinar sus parámetros, partiendo de pesos aleatorios, de forma automática o semiautomática.

B.1. Características de un LLM

Antes de describir el proceso de entrenamiento, establecemos la base del procedimiento y algunos conceptos fundamentales en el funcionamiento de un LLM.

B.1.1. Tokenización y entrada

Los LLM no procesan una secuencia de texto completa, sino que la dividen en fragmentos más pequeños y manejables. Llamamos a estos fragmentos *tókenes*,

y su delimitación viene de preprocesar el conjunto de preentrenamiento antes descrito.

Definición (Tokenización): Llamaremos Σ al conjunto de símbolos que queremos procese nuestro modelo (letras, números, *emojis*, etc.) y Σ^* su conjunto de cadenas. Llamamos *tokenización* a tres procesos similares pero no iguales:

1. Al proceso de asignar identificadores a cadenas de texto en el conjunto de entrenamiento. Formalmente, es el proceso de definir un $\mathcal{X} \subset \Sigma^*$, el *vocabulario* de nuestro modelo, y construir una función, posiblemente esto-cástica, $f : \Sigma^* \rightarrow \mathcal{X}^*$ llamada *tokenizador* que transforma toda secuencia de caracteres en una secuencia de tókenes de nuestro vocabulario⁴⁸. Estos tókenes suelen representarse, en lugar de mediante la cadena de símbolos subyacente, con un identificador único, un número natural.
2. Al proceso de aplicar la función f antes descrita a una entrada cualquiera (no necesariamente del conjunto de preentrenamiento), dividiéndola en tókenes.
3. Es, además, relativamente común en la literatura llamar también tokenización al proceso descrito en la segunda acepción seguido del proceso de embebimiento de estos tókenes en el espacio de entrada del modelo. No emplearemos esta terminología.

A lo largo de esta sección nos referiremos a la primera acepción. ○

La tokenización no es un proceso arbitrario, sino que responde a una característica de la tarea de embebimiento de texto en un espacio vectorial: la capacidad de los modelos de extraer significado del texto es limitada, por tanto existe un rango de dimensiones de embebimiento óptimas que optimizan la cantidad de significado representable siendo máximamente eficientes en memoria. Puesto que se quiere aprovechar la dimensión de embebimiento con vectores densos, compactar el texto permite una mayor diversidad en el conjunto de entrada y, en consecuencia, a vectores más variados.

Esta tokenización se realiza de la siguiente forma [64]:

1. A cada símbolo en Σ se le asocia un token, es decir, $\mathcal{X}_0 := \Sigma$.
2. Se analiza el corpus de texto para buscar las secuencias de símbolos más susceptibles a compactación (por ejemplo aquellas que, siendo sustituidas por un nuevo símbolo, reducen la longitud del corpus lo máximo posible). Se escoge la mejor según alguna métrica, a saber α , y se define $\mathcal{X}' = \mathcal{X} \sqcup \{\alpha\}$.

⁴⁸Nótese, entonces, que para que tal función pueda existir debe darse $\mathcal{X}^* = \Sigma^*$. Esto suele conseguirse imponiendo primero que $\Sigma \subset X$, donde si $\mathcal{X} = \Sigma$ el LLM se dice «*de nivel de carácter*» (*character-level LLM*).

3. Se repite el paso anterior hasta llegar al número de tokens deseado, por ejemplo $|\mathcal{X}| \approx 100\,000$.

Tras esto, el proceso descrito en el segundo punto de la definición de tokenización es simplemente sustituir en una cadena de caracteres los números (o, en general, identificadores) asociados a cada token:

pega	dos	los	unos	a	los	otros	,	los	sober	bios	al	cáza	res	de	la
865	23	86	589	42	86	122	99	86	542	582	73	872	892	424	85

↓

[865, 23, 86, 589, 42, 86, 122, 99, 86, 542, 582, 73, 872, 892, 424, 85]

La tokenización naturalmente no tiene por qué ser aditiva: si tenemos dos fragmentos con k_1 y k_2 tokens respectivamente, la tokenización de su concatenación tendrá $k_1 + k_2$ tokens como máximo, habitualmente menos.

Consideremos además que la tokenización es habitualmente un proceso estocástico[50]: puesto que una secuencia tiene más de una representación⁴⁹, los tokenizadores en ocasiones incorporan un factor de aleatorización para hacer a los modelos más robustos ante representaciones alternativas de los tokens.

B.1.1.1. Embebimiento de los tokens

Definición (Representación oculta): Llamamos *representación distribuida*, *latente* u *oculta* al resultado de embeber un token concreto mediante su matriz de *embebimiento semántico* en el espacio de entrada del modelo. Suelen ser vectores de muy alta dimensionalidad que codifican el significado (básico, sin contexto) de un token.

Estos vectores sufren tras esto la adición de un identificador de posición, un *embebimiento posicional*. ○

El embebimiento semántico se aprende durante el proceso de entrenamiento, y vienen a representar el significado de cada token de una forma coherente con la estructura de espacio vectorial de la representación oculta. Esto quiere decir que, en general, el significado de los tokens se transporta con las repre-

⁴⁹Naturalmente, al estar $\Sigma \subsetneq \mathcal{X} \subset \Sigma^*$, toda secuencia puede tokenizarse a nivel de carácter y, habitualmente, de otras muchas formas más eficientes en longitud. Existirán otras tokenizaciones (más eficientes) siempre que haya presente algún token multisimbólico de $\mathcal{X}$ en la secuencia, lo que es extremadamente común con vocabularios grandes.

representaciones vectoriales: siendo $r(t)$ la representación distribuida de un token t , poniendo un ejemplo clásico:

$$r(\text{mujer}) - r(\text{hombre}) \approx r(\text{reina}) - r(\text{rey})$$

de forma que el vector que produce esta diferencia puede asociarse, al menos aproximadamente, con el género. Este tipo de representaciones vectoriales preceden considerablemente a los LLM. [65]

B.1.1.2. Embebimientos posicionales

El bloque de autoatención es, por construcción, invariante ante permutaciones del conjunto de entrada, pues no hay diferenciación entre elementos por el orden de entrada. En caso de no explicitar el orden, el modelo procesa la secuencia como un bloque desordenado y no como una estructura ordenada linealmente. Para romper esta simetría y dotar a la red de una noción de orden entre tokens, se inyecta esta información posicional en las representaciones ocultas antes de su entrada en los bloques de decoder.

Originalmente se empleaban *embebimientos originales absolutos* (se sumaba a las representaciones un vector diseñado de forma semiartesanal), que el modelo aprendía durante el entrenamiento de alguna forma simbolizaba orden, presentándose también modelos en los que estos embebimientos eran también parámetros aprendibles.

Las arquitecturas contemporáneas, sin embargo, al estar orientadas a la eficiencia en ventanas de contexto enormes optan por mecanismos de *posición relativa*, como es el caso de RoPE (empleado en Qwen3-VL, tanto en los tokens visuales, donde se usa una versión 2D, como en los tokens textuales [20]). En RoPE (que se aleja de la noción de embebimiento propiamente dicho), en lugar de aplicar una adición tensorial antes de la entrada al transformer se opera directamente sobre los vectores de *query* y *key* en cada cabeza de atención. Se modifica la geometría mediante una rotación, distorsionando la señal que reciben entre sí. [66]

En primer lugar se segmenta el espacio vectorial d -dimensional en $d/2$ subespacios bidimensionales ortogonales y se aplica a estos vectores rotación, cuyo ángulo depende del índice de posición t y la frecuencia asignada a cada par de dimensiones. Este sistema perturba el producto escalar entre consultas y claves como función de la distancia relativa entre los índices de posición de sus correspondientes tokens. De esta forma, se consigue una «penalización

semántica» (en el coeficiente de la matriz de atención) que se incrementa conforme los tókenes se alejan.

B.1.2. Contexto, ventana de contexto

Definición (Contexto de un LLM): Llamamos *contexto* a toda la información que se le introduce a un LLM para que produzca su respuesta. En una conversación con un chatbot, por ejemplo, el contexto puede ser un documento que se le ha introducido y toda la conversación que se ha tenido hasta ese momento.○

Definición (Ventana de contexto de un LLM): Llamamos *ventana de contexto* (medida en tókenes) a la cantidad total de información que un LLM puede procesar simultáneamente sin tener que prescindir de parte de ella. Cuando el contenido que le hemos introducido supera esa ventana de contexto, el LLM pierde la capacidad de prestar atención al inicio de la conversación gradualmente, conforme este se escapa de su ventana de contexto. ○

Independientemente de si hemos superado o no la ventana de contexto, es un fenómeno conocido en LLM que su capacidad para *prestar atención* a partes de la conversación se degrada considerablemente conforme aumenta la longitud del contexto [48]. Es por esto que existen las pruebas de tipo «aguja en un pajar» («*needle-in-a-haystack*»), en las que la tarea es extraer de un contexto largo y mayoritariamente irrelevante una pieza de información pequeña y precisa.

B.1.3. Inferencia

Los LLM son modelos cuya función es predecir la siguiente palabra en un texto, lo que consiguen produciendo para cada texto una distribución de probabilidad definida sobre su vocabulario, $\mathcal{X}$. Llamamos, entonces, *inferencia*, al proceso de generar texto a partir de una entrada al modelo:

1. Se calcula la distribución de probabilidad para el siguiente elemento de la secuencia.
2. Se escoge, siguiendo alguna regla, el siguiente elemento a partir de la distribución de probabilidad que genera el modelo.
3. Se concatena este último elemento a la entrada.
4. Se repite el proceso desde el primer punto hasta estar satisfecho con el resultado.

Por ejemplo, se realiza inferencia cuando se hace una solicitud a un LLM en la web, siendo el contexto la pregunta realizada.

Existen tres clases de parámetros que dominan la generación del modelo: una se refiere a la forma en la que se calculan las probabilidades a partir de las puntuaciones del modelo, otra a qué candidatos de token se emplean para generar el siguiente elemento de la secuencia, y la última a cuántos candidatos a token se emplean y cómo se explora la generación a futuro. Estos tres tipos de parámetros tienen diferentes utilidades e inconvenientes, como se expone en [67].

Definición (Temperatura): Llamamos *temperatura*, T , en un LLM al parámetro encargado de controlar cómo de «plana» (homogénea) es la distribución. Esto se consigue introduciendo T como escalar en la última función softmax, mediante la cual el modelo nos devuelve las probabilidades finales.

Siendo $\mathcal{X}$ el vocabulario del LLM y $S : \mathcal{X} \times \mathcal{X}^* \rightarrow \mathbb{R}$ la puntuación (lógit) asignada a cada posible siguiente token, la probabilidad calculada por el modelo a partir de los lógits según el parámetro T viene dada por

$$f_t(y) = \frac{e^{S_{t(y)}/T}}{\sum_{x \in \mathcal{X}} e^{S_{t(x)}/T}},$$

donde $S_t(x)$ es abuso de notación para $S(x \mid x_1 \dots x_t)$ y $f_t(x)$ lo es a su vez para $f(x \mid x_1 \dots x_t)$. Así, conforme $T \rightarrow \infty$, la distribución tiende a la homogeneidad, mientras que si $0 < T < 1$ se favorece enormemente los tokens con una puntuación alta. $\circ$

Definición (Top- k): Parámetro que controla de entre cuántos de aquellos tokens que el modelo ha puntuado como mejores se escoge el siguiente. Siendo f la función descrita en la anterior definición y X_t^k el conjunto de tokens k con mayor probabilidad según f_t , se define la probabilidad «operativa» o «funcional» del modelo asignado a cada token durante inferencia como

$$\hat{f}_t(y) = \begin{cases} 0 & \text{si } y \notin X_t^k \\ \frac{f_t(y)}{\sum_{x \in X_t^k} f_t(x)} & \text{si } y \in X_t^k. \end{cases}$$

En el caso $k = 1$ tenemos la generación codiciosa, mientras que un k alto hace que el modelo sea menos predecible. $\circ$

De forma alternativa a $\text{top-}k$ es posible emplear *muestreo nuclear* (*nucleus sampling*) (o más comúnmente, $\text{top-}p$), en el que las muestras se toman de aquel subgrupo de mayor probabilidad cuyas probabilidades combinadas suman al menos p siendo este un parámetro.

Estos dos parámetros interfieren en la forma en la que se ensambla la respuesta, pero hemos asumido tácitamente a lo largo de su descripción un ensamblado esencialmente lineal. Resulta, sin embargo, que este método de inferencia tiene sus desventajas.

Ejemplo : Supongamos que un modelo genera, empleando las técnicas antes descritas, la siguiente secuencia:

$$\begin{aligned} a_1 a_2 a_3 \dots a_t : f_0(a_1) &= 0.55, & f_i(a_i) &= 0.1 \forall i > 1 \\ b_1 b_2 b_3 \dots b_t : f_0(b_1) &= 0.30, & f_i(b_i) &= 0.45 \forall i > 1 \end{aligned}$$

Pese a que el modelo tiene mucha más *confianza* en que el primer elemento de la secuencia es a_1 , tiene mucha más confianza promedio en la segunda secuencia completa que en la primera. La única forma de explorar esta secuencia habría sido comenzar generando, en lugar de un único token y ensamblar la respuesta, ensamblar simultáneamente varias, arrastrando métricas de probabilidad media, etc. ○

Definición (Generación en haces): Llamamos *beam generation* o *generación en haces* al proceso de considerar simultáneamente varios «caminos» durante la generación. En lugar de tomarse un único token de entre a los que el modelo asigna mejores puntuaciones, se escogen varios de ellos.

Se ensamblan las secuencias y se emplea una métrica (probabilidad media, probabilidad total, probabilidad media penalizada por longitud, etc) para escoger los siguientes caminos que se siguen. Si se encuentra un camino que resulta en una mejor métrica que otro que se está explorando y ya se han alcanzado el máximo de caminos a explorar posibles, se sustituye este último por el nuevo encontrado. ○

B.2. Etapas del entrenamiento

El entrenamiento de un LLM está dividido en dos etapas claramente diferenciadas: un preentrenamiento y un postentrenamiento. A su vez, el postentrena-

miento se encuentra también subdividido en varias etapas, no siendo necesaria ninguna de ellas.

B.2.1. Preentrenamiento

El preentrenamiento es la etapa computacionalmente más exigente de todo el proceso, a menudo consumiendo en torno a un 95% del tiempo total de cálculo.

Durante esta etapa, cuyo objetivo es que el modelo aprenda la estructura estadística del lenguaje (aprenda a generar texto coherente) y, como efecto secundario, aprenda también algunas características del mundo, el modelo procesa enormes cantidades de texto. Este texto se conoce como *conjunto de preentrenamiento*.

Definición (Conjunto de preentrenamiento de un LLM): Llamamos *conjunto de preentrenamiento* de un LLM a una base de datos enorme (cientos de terabytes de información) compuesta únicamente de texto. Este conjunto de datos se emplea para que el modelo aprenda la estructura estadística del lenguaje. ○

Estas bases de datos a menudo se obtienen ingiriendo todo el texto presente en internet casi en su totalidad. La obtención de este corpus textual, pues se busca que el contenido sea de la mayor calidad posible, a menudo incurre en problemas de derechos de autor o *copyright* al escanearse obras protegidas.

Para la ingesta del contenido presente en internet, a menudo se emplea un proceso basado en *bots* de recopilación automática que, en el contexto de internet, se denominan *crawlers*. Estos bots a menudo tienen restringido el acceso a páginas webs marcadas como «no deseadas» (contenido ofensivo, machista, racista, sexual, etc). Pese a esto, el contenido conforme se obtiene de los *crawlers* se filtra para eliminar texto que pueda haber pasado el primer filtro físico. Tras este primer filtrado, se eliminan también referencias a personas, direcciones, números de teléfono, etc. e información que los desarrolladores consideren innecesaria o legalmente dudosa.

Puesto que se pretende enseñar la estructura estadística del lenguaje al modelo con este conjunto de datos, deberá ser diverso. A menudo se considera que es una muestra representativa del lenguaje.

El entrenamiento del modelo se realiza empleando variantes de descenso de gradiente según una función de error (como la entropía cruzada) entre la distribución que presenta el modelo en cada momento y la distribución deseada. En el caso de querer generar un token w , sería la distribución con $\mathbb{P}(w) = 1$.

El conjunto de preentrenamiento se subdivide en bloques de tókenes de longitud igual a la ventana de contexto. Como se explica en [Capítulo 2](#), el decodificador genera un vector de predicción para todas y cada una de las posiciones de entrada, por lo que es posible calcular de forma paralela todas sus predicciones para todas las palabras en el corpus (o, por lo menos, en el bloque de ventana que esté procesando). En consecuencia, es posible calcular el error que comete en bloques de gran tamaño del conjunto de preentrenamiento, evitando un proceso secuencial e iterativo extremadamente ineficiente.

El modelo que resulta del preentrenamiento se conoce como *modelo base*, y es, en esencia, un simulador de textos de internet. Esto, en sí mismo, no tiene por qué ser útil: habitualmente el modelo deseado tiene una función específica para la que el lenguaje es simplemente un vehículo, como funcionar como asistente, programador... o como transcriptor de textos de Grothendieck.

Nota (Modelo base como asistente): Es posible conseguir *engañar* a un modelo para que funcione de alguna manera concreta sin haber hecho ningún proceso para enseñarle ese comportamiento. Por ejemplo, si en su corpus de entrenamiento hay conversaciones de tipo asistenciales, incluso si son pocas, es posible conseguir que se comporte como uno estructurando la conversación de una forma que parezca una conversación entre un asistente y un usuario.

Esto, sin embargo, es un proceso completamente abierto: el modelo no sabe diferenciar qué parte es la que él debe cumplir. Tan pronto como haya terminado produciendo el texto que nosotros, como humanos, adscribimos al asistente, podría continuar produciendo una nueva pregunta del usuario, en el mismo formato que el dado, si es que juzga que esa es la continuación más probable.○

Es sobre el modelo base que se realiza el postentrenamiento

B.3. Ajuste fino supervisado

Se conoce como *ajuste fino supervisado* (*supervised fine tuning*, *SFT*, o simplemente *ajuste fino*, *fine tuning* o *FT*) a enseñar al modelo a realizar una tarea específica. Habitualmente el conjunto de ajuste fino (llamado *conjunto de ins-*

trucción) es mucho más pequeño que el de preentrenamiento, pues a menudo es un texto muy refinado y cuidadosamente escogido [68]. Puesto que ya la conoce, el modelo no necesita reaprender la estructura estadística del lenguaje.

Este proceso termina con el modelo SFT (supervised *fine-tuning* model), que está considerablemente más cerca de estar acabado que el modelo base.

Los modelos aprenden mediante ajuste fino a tareas tan variadas como funcionar como asistentes o a razonar primero ante problemas complejos (en lugar de producir la respuesta directamente). A menudo estos ajustes finos involucran que los modelos se familiaricen con nuevos tokens, introducidos por los desarrolladores, para diferenciar diferentes comportamientos (uso de herramientas, modos de razonamiento, diferenciación entre preguntas y respuestas, etc.).

Ejemplo (Modelos asistenciales tokenización de la conversación): La primera de las diferencias con el entrenamiento primero es que se le asigna al modelo una estructura conversacional mucho más rígida que aquella con la que contaba. Se introducen, de forma esencial pero no exclusivamente, los tokens de turno.

Estos son únicamente delimitadores de la forma `<im_start>`, `<im_sep>` e `<im_end>` [69], que marcan la estructura de turnos propia de una conversación:⁵⁰

```
<im_start>user<im_sep>¿Cuánto es 2+2?<im_end>
<im_start>model<im_sep>2 + 2 = 4.<im_end>
```

Nótese que todos estos tokens no son conocidos por el tokenizador (por lo menos el tokenizador como antes se ha descrito), sino que se inyectan en las interacciones con el modelo por un proceso externo, aparte de la tokenización de la entrada. Esto supone una barrera externa, impuesta al modelo no estructuralmente, para que diferencie preguntas de respuestas.

Esta separación no es completamente estanca: el modelo aprende solo de forma «difusa» aquello que son instrucciones y aquello que es contenido, lo que puede suponer un riesgo en algunas situaciones donde el contenido no viene generado expresamente por el modelo (véase [Subsección B.5.2](#)). ○

La pretensión es, por tanto, superponer a la capacidad del modelo una «personalidad» o un «saber hacer» específico. Sirve para condicionar a nuestro

⁵⁰Esta notación no es común a todos los modelos, cada empresa (o incluso modelo) emplea su propia notación y forma de estructurar las conversaciones.

modelos a actuar de una forma que le hemos especificado, pues aprenden que, bajo la tokenización de conversación o aquella para la que han sido ajustados, el comportamiento más probable es el que se le muestra durante este postentrenamiento.

B.3.1. Obtención de la base de datos de postentrenamiento

Para obtener el conjunto de instrucción es habitual contratar empresas con trabajadores especializados (llamados *etiquetadores* o *labelers*) que se dedican profesionalmente a crear conversaciones para entrenar estos modelos. Para guiar a los etiquetadores en este proceso, a menudo el desarrollador del modelo crea una «guía para etiquetadores» donde se expresa el carácter general y cómo se desea que sean estas conversaciones⁵¹.

Así, cuando se solicita una información sobre, por ejemplo, París, la IA no está buscando todo internet y condensándolo, o buscando toda la información de su conjunto de entrenamiento y componiendo una respuesta, sino que está tomando muestras de una distribución de probabilidad que se basa tanto en su conjunto de entrenamiento como en su instrucción.

Notemos que los LLM se están comenzando a involucrar incluso en el proceso de obtención de esta, tan delicadamente generada, base de datos. En ocasiones la mayor parte de la respuesta es la dada por otro LLM, con un poco de cambio por parte del etiquetador. En consecuencia, puesto que un LLM aprende a imitar a sus etiquetadores, también aprende a imitar a otros LLM que han sido empleados para crear su conjunto de instrucción.

B.4. Entrenamiento de refuerzo

El entrenamiento por refuerzo (RL, del inglés *Reinforcement Learning*) tiene como objeto promover algunos comportamientos en el LLM y evitar otros: en general con modelos asistenciales se quiere que el modelo cometa menos errores, sea más preciso y más fehaciente. Para esto, sin embargo, resulta un

⁵¹Este mismo proceso fue el seguido a la hora de etiquetar los resultados, pues se suministró una guía para etiquetadores y una guía para revisores.

obstáculo casi insalvable que el modelo y los investigadores no siguen las mismas rutas para aprender ni «pensar».

Puesto que el equipo dedicado a su entrenamiento no sabe realmente cómo piensa (el modelo es, a todos los efectos, una caja negra), escoger sobre qué tipos y rutas de razonamientos entrenarlo para mejorar sus resultados resulta una tarea compleja.

El RL se plantea como solución circunvalar en la medida de lo posible la interacción humana. En lugar de una terna (problema, proceso de resolución, solución), se parte únicamente de un par (problema, solución). A partir de este par se permite que el modelo realice inferencia en paralelo sobre muy diversas líneas de pensamiento, y se recogen aquellas que llegan a la solución deseada⁵². Una vez se tiene esta información, se premia aquellas rutas de razonamiento que han llevado a la respuesta correcta mediante un nuevo afinado. En éste, el conjunto de ajuste finos son sus propias respuestas «correctas» al problema de RL, de forma que el modelo aprende a favorecer los razonamientos que le han sido eficaces [68].

Pese a que el RL aumenta considerablemente la precisión de nuestros modelos, presenta también el fallo de aumentar considerablemente el número de tokens que emplea para llegar a sus conclusiones[70].

B.4.1. Dominios no verificables, RLHF

En RL según se ha expuesto resulta imprescindible tener una «respuesta correcta» a cada pregunta para poder evaluar las del modelo. Existen, sin embargo, problemas donde no existe tal cosa como una «respuesta correcta» y, puesto que el RL es una técnica extremadamente fructífera, incluso se quiere aplicar en estos campos. Llamamos a aquellos campos donde no existe tal «respuesta correcta» (como la escritura creativa, comedia, inspiración, etc.) *dominios no verificables*.

Una idea en estos campos sería, en lugar de poder contrastar sencillamente si la última respuesta es correcta (como en un problema de física o ingeniería), emplear un revisor humano que puntúe la calidad de la respuesta. Sin embargo,

⁵²Es posible que se apliquen criterios extra: se puede premiar la capacidad expositiva, la sucintez, el rigor... Es un criterio abstracto, y es precisamente este criterio, así como el corpus de prompts al modelo para RL, lo que define la utilidad y dificultad del proceso.

puesto que en RL se emplean miles o millones de ejemplos, es inviable tener un equipo de humanos que revise la producción, más aún cuando se quiere que este criterio sea consistente.

Como alternativa a la calificación humana, por tanto, se emplea un *modelo de recompensa* [70], [71] creado con la intención de simular el criterio humano. Este modelo abstrae el criterio humano a partir de unas pocas respuestas de revisores humanos, y se emplea más tarde en lugar de un revisor humano para calificar las respuestas del modelo. Esta técnica se conoce como RLHF, por sus siglas en inglés, *Reinforcement Learning from Human Feedback*, o, en castellano, aprendizaje de refuerzo mediante retroalimentación humana.

Prima facie esta técnica simplemente mueve la necesidad de revisores humanos de un sitio (evaluación de respuestas para el entrenamiento del modelo generador) a otro (evaluación de respuestas para el entrenamiento del modelo evaluador). Entrenar una red neuronal secundaria para replicar el criterio humano puede parecer una forma más indirecta y más propensa al ruido que entrenar directamente al modelo original. La ventaja, sin embargo, radica en que ambos modelos están aprendiendo a realizar tareas fundamentalmente diferentes: la tarea de *discriminar* y *evaluar* respuestas, resulta ser mucho más sencilla de aprender y realizar correctamente para los modelos que la de *generar* [68]. Mientras que la red secundaria aprende a simular el criterio humano rápidamente y con pocos ejemplos, aquella que debe crear contenido que satisfaga este criterio tarda más tiempo.

Emplear un modelo intermedio, sin embargo, no está exento de desventajas. El modelo generador ya no aprende a satisfacer el criterio humano, sino una versión simulada, y por tanto de peor calidad, del mismo, pues el criterio que impone el modelo evaluador no tiene por qué estar completamente alineado con el criterio humano. Pasadas suficientes iteraciones, el modelo generador aprende a producir *ejemplos adversariales*, que se aprovechan de defectos en el modelo discriminador (fallos en su símil al criterio humano real) para recibir puntuaciones más altas [72]. Cuando esto ocurre, la calidad de las respuestas generadas comienza a descender, siendo óptimo parar el proceso antes de que comience esta dinámica adversarial.

B.5. Alucinaciones

El ajuste fino de instrucción tiene un enorme beneficio para modelos de tipo asistente, pues adquieren un tono serio, seguro y formal propio de esta labor. El proceso, sin embargo, instala también en el modelo una «personalidad» decidida, incluso soberbia, en la que posee una respuesta para todo independientemente del campo; independiente, incluso, de si realmente conoce la respuesta. [73]

Puesto que en sus conjuntos de instrucción no suele haber cabida a vacilación ni duda, el modelo aprende a no hacerlo [74]. Este comportamiento no supone un problema, por supuesto, hasta que el modelo se encuentra con preguntas cuya respuesta no puede ser otra que un sencillo «no sé». Ante una pregunta suficientemente similar en forma pero suficientemente alejada en contenido del conjunto de instrucción, el modelo genera una respuesta estadísticamente consistente con su conjunto de entrenamiento, pero con la posibilidad de que sea completamente errónea. Esta respuesta se presenta, además, con la misma seguridad que aquella completamente correcta, pues el modelo tradicional no tiene forma de discriminar cuándo algo es cierto de cuando no. Llamamos *alucinaciones* a este tipo de respuestas incorrectas).

Asimismo, es posible que durante el ajuste fino el modelo aprenda un dato, pero no se le proporcione información aledaña –como *por qué* dicho dato es cierto–, en cuyo caso el modelo, siendo consistente con su entrenamiento, produce la respuesta correcta y alucina una justificación. [75]

B.5.1. Reconocer desconocimiento

Una de las formas de lidiar con las alucinaciones es realizar un proceso de prospección del conocimiento del LLM, muestreando respuestas a preguntas factuales y consiguiendo así una nube de puntos donde es posible delinear una *frontera de conocimiento* (*knowledge boundary*) [74]. Una vez se tiene constancia de qué información ignora, se confecciona un conjunto de instrucción con ejemplos donde el modelo reconoce no saber la respuesta a estas preguntas. Este proceso, conocido como *honesty fine-tuning*, le devuelve al modelo la capacidad de admitir desconocimiento cuando sea el caso, que hasta cierto punto pierde durante la instrucción.

Estos *interrogatorios* pueden generarse de forma automática a partir de una fuente de conocimiento (como Wikipedia) y pidiéndole a otro modelo que genere preguntas factuales en base a ella. Cuando el modelo evaluador considera que se ha fallado una pregunta,

Usuario: ¿Cuándo se nombra a Filipinas en "Tiempo de Silencio"?

Modelo: Cuando Pedro y Amador viajan allí para encontrarse con Muecas.

Se marca como fallida, y se añade al conjunto de evaluación un ejemplo (o más, si se añaden variantes) en el que el modelo reconoce no conocer la respuesta.

Usuario: ¿Cuándo se nombra a Filipinas en "Tiempo de Silencio"?

Modelo: No conozco la respuesta exacta a esa pregunta.

Este postentrenamiento no basa su utilidad simplemente en que el modelo aprenda a decir que «no sabe» ante preguntas suficientemente difíciles (lo que limitaría enormemente la capacidad del modelo) sino que se aprovecha de la presencia de procesos análogos a la duda, presentes en modelos suficientemente complejos [73], [76]. Así, estos modelos son capaces de distinguir preguntas complejas cuya respuesta saben de aquellas en que no [77], [78].

B.5.2. Herramientas de apoyo

Otra posible opción para aumentar la precisión del modelo es un cambio de paradigma: en lugar de emplear al modelo simultáneamente como fuente de información y comunicador, es posible mover su labor esencialmente a que sea solo la segunda. Para esto es necesario dotar al modelo de la capacidad de acceder a una fuente de conocimiento.

Esto se consigue entrenando al modelo a *pedir ayuda*, a menudo empleando un buscador de internet o una base de datos externa, lo que se consigue con un ajuste fino. Un posible ejemplo de una de estas muestras es:

Usuario ¿Dónde está Macondo?

Modelo <SEARCH_START> Where is "Macondo"? <SEARCH_END>

Search Macondo is a fictional town described in Gabriel García Márquez's novel One Hundred Years of Solitude (as well as several others of his works). It is the hometown of the Buendía family.

Modelo [...]

Por supuesto, debe también implementarse un sistema que actúa como intermediario entre modelo y usuario, que debe incluir una funcionalidad que añade automáticamente la información de las búsquedas del modelo [79]. Así, en el momento en el que se cierra el espacio de búsqueda, este sistema completa los siguientes tokens del contexto con información resumida que encaja con la búsqueda del modelo [80]. Es habitual que esta información hallada por el intermediario no se muestre al usuario, o se haga solo parcialmente.

El modelo puede, entonces, conseguir información para responder de forma efectiva una gran variedad de preguntas sin apenas aumentar el coste computacional o su complejidad [79]. La complejidad de este proceso, similar al RAG aunque no igual en concepto, reside en la forma en la que se busca y suministra al modelo la información.

■ Bibliografía

- [1] J. A. Navarro, «Archivo de imágenes de Grothendieck». 2026.
- [2] A. Vaswani *et al.*, «Attention Is All You Need», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/1706.03762>
- [3] J. Alammari y M. Grootendorst, *Hands-On Large Language Models: Language Understanding and Generation*. Sebastopol, CA: O'Reilly Media, 2024. [En línea]. Disponible en: <https://www.oreilly.com/library/view/hands-on-large-language/9781098150952/>
- [4] C. An *et al.*, «Training-Free Long-Context Scaling of Large Language Models», 2024, [En línea]. Disponible en: <https://arxiv.org/abs/2402.17463>
- [5] A. Yang *et al.*, «Qwen3 Technical Report», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2505.09388>
- [6] L. Narváez Macarro, «En recuerdo de Alexander Grothendieck: Prólogo para una lectura de su vida y obra», *La Gaceta de la Real Sociedad Matemática Española*, vol. 20, n.º 2, pp. 297-324, 2017.
- [7] L. Alonso Tarrío y A. Jeremías López, «La obra de Alexander Grothendieck», *La Gaceta de la Real Sociedad Matemática Española*, vol. 20, n.º 3, pp. 445-467, 2017, [En línea]. Disponible en: <https://gaceta.rsme.es/abrir.php?id=183>
- [8] A. Grothendieck, *Cosechas y siembras: Reflexiones y testimonio sobre un pasado de matemático*. 2024. [En línea]. Disponible en: <https://csg.igrothendieck.org/wp-content/uploads/2024/06/G95ttt.pdf>
- [9] A. Grothendieck, «Sur quelques points d'algèbre homologique, I», *Tohoku Mathematical Journal*, vol. 9, n.º 2, pp. 119-221, 1957, doi: 10.2748/tmj/1178244839.
- [10] «Les inestimables Grothendieck papers». Accedido: 24 de junio de 2026. [En línea]. Disponible en: https://www.liberation.fr/debats/2016/06/08/les-inestimes-grothendieck-papers_1458196/

- [11] A. Grothendieck, «Déclaration d'intention de non-publication par Alexandre Grothendieck». Accedido: 21 de junio de 2026. [En línea]. Disponible en: <https://tqft.net/misc/Grothendieck%27s%20Declaration.pdf>
- [12] «Grothendieck : des gribouillis de génie en héritage». Accedido: 21 de junio de 2026. [En línea]. Disponible en: https://www.liberation.fr/futurs/2016/01/14/grothendieck-des-gribouillis-de-genie-en-heritage_1426478/
- [13] J. Lichfield, «Alexander Grothendieck: Legal battle over 'scribblings' of 20th century's 'greatest mathematician' and the secrets they may contain». Accedido: 21 de junio de 2026. [En línea]. Disponible en: <https://www.independent.co.uk/news/world/europe/alexander-grothendieck-legal-battle-over-scribblings-of-20th-century-s-greatest-mathematician-and-the-secrets-they-may-contain-a6815216.html>
- [14] A. Grothendieck, «Les Dérivateurs», 1991.
- [15] J. Memon, M. Sami, y R. A. Khan, «Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)», 2020, [En línea]. Disponible en: <https://arxiv.org/abs/2001.00139>
- [16] M. Li *et al.*, «TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models», *CoRR*, vol. abs/2109.10282, 2021, [En línea]. Disponible en: <https://arxiv.org/abs/2109.10282>
- [17] N. Parmar *et al.*, «Image transformer», pp. 4055-4064, 2018, [En línea]. Disponible en: <https://arxiv.org/abs/1802.05751>
- [18] A. Dosovitskiy *et al.*, «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale», 2021, [En línea]. Disponible en: <https://arxiv.org/abs/2010.11929>
- [19] J. Li, D. Li, S. Savarese, y S. Hoi, «BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2301.12597>
- [20] J. Bai *et al.*, «Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2308.12966>
- [21] S. Bai *et al.*, «Qwen3-VL Technical Report», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2511.21631>
- [22] J.-B. Alayrac *et al.*, «Flamingo: a Visual Language Model for Few-Shot Learning», 2022, [En línea]. Disponible en: <https://arxiv.org/abs/2204.14198>

- [23] R. M. Schmidt, «Recurrent Neural Networks (RNNs): A gentle Introduction and Overview», 2019, [En línea]. Disponible en: <https://arxiv.org/abs/1912.05911>
- [24] A. Ng, «Deep Learning Specialization», 2017, [En línea]. Disponible en: <https://www.coursera.org/specializations/deep-learning>
- [25] D. Bahdanau, K. Cho, y Y. Bengio, «Neural Machine Translation by Jointly Learning to Align and Translate», 2016, [En línea]. Disponible en: <https://arxiv.org/abs/1409.0473>
- [26] I. Sutskever, O. Vinyals, y Q. V. Le, «Sequence to Sequence Learning with Neural Networks», 2014, [En línea]. Disponible en: <https://arxiv.org/abs/1409.3215>
- [27] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, y I. Sutskever, «Robust Speech Recognition via Large-Scale Weak Supervision», 2022, [En línea]. Disponible en: <https://arxiv.org/abs/2212.04356>
- [28] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, y S. Sanghai, «GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2305.13245>
- [29] DeepSeek-AI *et al.*, «DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model», 2024, [En línea]. Disponible en: <https://arxiv.org/abs/2405.04434>
- [30] J. L. Ba, J. R. Kiros, y G. E. Hinton, «Layer Normalization», 2016, [En línea]. Disponible en: <https://arxiv.org/abs/1607.06450>
- [31] J. Devlin, M.-W. Chang, K. Lee, y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», 2019, [En línea]. Disponible en: <https://arxiv.org/abs/1810.04805>
- [32] A. Radford, K. Narasimhan, T. Salimans, y I. Sutskever, «Improving Language Understanding by Generative Pre-Training», technical report, jun. 2018. [En línea]. Disponible en: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- [33] J. Howard y S. Ruder, «Universal Language Model Fine-tuning for Text Classification», 2018, [En línea]. Disponible en: <https://arxiv.org/abs/1801.06146>
- [34] S. J. Pan y Q. Yang, «A Survey on Transfer Learning», *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, n.º 10, pp. 1345-1359, 2010, doi: 10.1109/TKDE.2009.191.

- [35] E. J. Hu *et al.*, «LoRA: Low-Rank Adaptation of Large Language Models», 2021, [En línea]. Disponible en: <https://arxiv.org/abs/2106.09685>
- [36] Y. Luo, Z. Yang, F. Meng, Y. Li, J. Zhou, y Y. Zhang, «An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2308.08747>
- [37] M. McCloskey y N. J. Cohen, «Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem», *The Psychology of Learning and Motivation*, vol. 24, pp. 109-165, 1989.
- [38] D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization», 2017, [En línea]. Disponible en: <https://arxiv.org/abs/1412.6980>
- [39] A. Aghajanyan, L. Zettlemoyer, y S. Gupta, «Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning», 2020, [En línea]. Disponible en: <https://arxiv.org/abs/2012.13255>
- [40] I. García-Redondo, «Topological and Geometric Deep Learning: Theory, Methods, and Applications», 2026, [En línea]. Disponible en: <https://ines.topology.rocks/teaching>
- [41] L. Valeriani, D. Doimo, F. Cuturello, A. Laio, A. Ansuini, y A. Cazzaniga, «The geometry of hidden representations of large transformer models», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2302.00294>
- [42] F. Meng, Z. Wang, y M. Zhang, «PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2404.02948>
- [43] C. H. Martin y M. W. Mahoney, «Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning», 2018, [En línea]. Disponible en: <https://arxiv.org/abs/1810.01075>
- [44] T. Dettmers, A. Pagnoni, A. Holtzman, y L. Zettlemoyer, «QLoRA: Efficient Finetuning of Quantized LLMs», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2305.14314>
- [45] B. Jacob *et al.*, «Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference», *CoRR*, vol. abs/1712.05877, 2017, [En línea]. Disponible en: <http://arxiv.org/abs/1712.05877>
- [46] X. Zhu, Z. Zhang, X. Chen, Y. Shi, M. Li, y J. Wu, «Connector-S: A Survey of Connectors in Multi-modal Large Language Models», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2502.11453>

- [47] J. Malgoire, «La Longue Marche à travers la théorie de Galois», 2007.
- [48] N. F. Liu *et al.*, «Lost in the Middle: How Language Models Use Long Contexts», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2307.03172>
- [49] P. Soviany, R. T. Ionescu, P. Rota, y N. Sebe, «Curriculum Learning: A Survey», 2022, [En línea]. Disponible en: <https://arxiv.org/abs/2101.10382>
- [50] T. Kudo, «Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates», 2018, [En línea]. Disponible en: <https://arxiv.org/abs/1804.10959>
- [51] I. Beltagy, M. E. Peters, y A. Cohan, «Longformer: The Long-Document Transformer», 2020, [En línea]. Disponible en: <https://arxiv.org/abs/2004.05150>
- [52] M. Tschannen *et al.*, «SigLIP 2: Multilingual Vision-Language Encoders with Improved Semantic Understanding, Localization, and Dense Features», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2502.14786>
- [53] N.-A. Ypsilantis, K. Chen, A. Araujo, y O. Chum, «Infusing fine-grained visual knowledge to Vision-Language Models», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2508.12137>
- [54] M. Levchenko, «Evaluating LLMs for Historical Document OCR: A Methodological Framework for Digital Humanities», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2510.06743>
- [55] J. Poznanski *et al.*, «olmOCR: Unlocking Trillions of Tokens in PDFs with Vision Language Models», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2502.18443>
- [56] J. Poznanski, L. Soldaini, y K. Lo, «olmOCR 2: Unit Test Rewards for Document OCR», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2510.19817>
- [57] G. Crosilla, L. Klic, y G. Colavizza, «Benchmarking Large Language Models for Handwritten Text Recognition», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2503.15195>
- [58] B. Wang *et al.*, «Image Over Text: Transforming Formula Recognition Evaluation with Character Detection Matching», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2409.03643>
- [59] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», 2016, [En línea]. Disponible en: <https://arxiv.org/abs/1506.02640>

- [60] N. Livathinos *et al.*, «Docling: An Efficient Open-Source Toolkit for AI-driven Document Conversion», 2025, [En línea]. Disponible en: <https://arxiv.org/abs/2501.17887>
- [61] B. Liang, H. Li, M. Su, X. Li, W. Shi, y X. Wang, «Detecting Adversarial Image Examples in Deep Neural Networks with Adaptive Noise Reduction», *IEEE Transactions on Dependable and Secure Computing*, vol. 18, n.º 1, pp. 72-85, ene. 2021, doi: 10.1109/tdsc.2018.2874243.
- [62] OpenAI, «Documentación de la API de OpenAi».
- [63] Google, «Documentación de la API de Gemini».
- [64] T. Kudo y J. Richardson, «SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing», 2018, [En línea]. Disponible en: <https://arxiv.org/abs/1808.06226>
- [65] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, y J. Dean, «Distributed Representations of Words and Phrases and their Compositionality», 2013, [En línea]. Disponible en: <https://arxiv.org/abs/1310.4546>
- [66] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, y Y. Liu, «RoFormer: Enhanced Transformer with Rotary Position Embedding», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2104.09864>
- [67] A. Fan, M. Lewis, y Y. Dauphin, «Hierarchical Neural Story Generation», 2018, [En línea]. Disponible en: <https://arxiv.org/abs/1805.04833>
- [68] L. Ouyang *et al.*, «Training language models to follow instructions with human feedback», 2022, [En línea]. Disponible en: <https://arxiv.org/abs/2203.02155>
- [69] R. Kumar, «Appendix A: ChatML Syntax Reference». [En línea]. Disponible en: <https://the-chatml-handbook.ranjankumar.in/chapters/AppendixA.html>
- [70] A. Karpathy, «[1hr Talk] Intro to Large Language Models». Accedido: 20 de junio de 2026. [En línea]. Disponible en: https://www.youtube.com/watch?v=zjkBMFhNj_g
- [71] A. Karpathy, «State of GPT». Accedido: 20 de junio de 2026. [En línea]. Disponible en: <https://www.youtube.com/watch?v=bZQun8Y4L2A>
- [72] L. Gao, J. Schulman, y J. Hilton, «Scaling Laws for Reward Model Overoptimization», 2022, [En línea]. Disponible en: <https://arxiv.org/abs/2210.10760>
- [73] A. Azaria y T. Mitchell, «The Internal State of an LLM Knows When It's Lying», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2304.13734>

- [74] L. Chen *et al.*, «Teaching Large Language Models to Express Knowledge Boundary from Their Own Signals», 2024, [En línea]. Disponible en: <https://arxiv.org/abs/2406.10881>
- [75] U. B. EECS, «John Schulman - Reinforcement Learning from Human Feedback: Progress and Challenges», 2023.
- [76] W. Labs, «The Dark Matter of AI [Mechanistic Interpretability]», 2024.
- [77] S. Kadavath *et al.*, «Language Models (Mostly) Know What They Know», 2022, [En línea]. Disponible en: <https://arxiv.org/abs/2207.05221>
- [78] S. Lin, J. Hilton, y O. Evans, «Teaching Models to Express Their Uncertainty in Words», 2022, [En línea]. Disponible en: <https://arxiv.org/abs/2205.14334>
- [79] T. Schick *et al.*, «Toolformer: Language Models Can Teach Themselves to Use Tools», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2302.04761>
- [80] S. Yao *et al.*, «ReAct: Synergizing Reasoning and Acting in Language Models», 2023, [En línea]. Disponible en: <https://arxiv.org/abs/2210.03629>



EPES

Escuela Politécnica
de Enseñanza Superior